INTERROS des LYCÉES

Collection dirigée par Éric MAURETTE

numérique et sciences informatiques

Stéphane PASQUET
Mikaël LEOPOLDOFF

Nathan

Remerciements

Les auteurs et l'équipe de Prepamath Édition tiennent à remercier Jean-Côme Charpentier, David Guénée et Bruno Mermet pour l'aide précieuse qu'ils ont apportée à cet ouvrage.

Illustrations : Prisca Baverey, Stéphane Pasquet

Coordination éditoriale : François Déliac

Conception couverture : Simon Geliot

Votre avis nous intéresse : `contact@prepamath.fr`

Retrouvez l'actualité des Interros des Lycées sur Facebook `facebook.com/interrosdeslycees/` et Instagram `@interrosdeslycees`.

© PREPAMATH Édition 2025
© Éditions NATHAN 2025- ISBN 9782095050184

Avant-propos

En classe de terminale générale, le programme de l'enseignement de spécialité de numérique et sciences informatiques (NSI) est conçu autour de quatre thèmes principaux :

- les données et leurs représentations,
- les bases de données,
- la programmation, les langages et les algorithmes,
- les architectures matérielles et réseaux.

À ces thèmes s'ajoutent des notions transverses telles que l'histoire de l'informatique ou les interfaces qui permettent la commande des systèmes.

Comme pour notre ouvrage de première NSI, nous avons mis l'accent sur la programmation et pour cela, proposé de télécharger la plupart des programmes mentionnés dans ce livre.

À partir de votre smartphone ou de votre tablette, les QRcodes vous permettront très facilement de télécharger les fichiers présentés dans l'ouvrage et de les envoyer par mail vers un ordinateur personnel. Gagnant ainsi un temps précieux, vous pourrez tester ces programmes chez vous et/ou les modifier pour exercer votre créativité et assimiler parfaitement leur fonctionnement.

Dans ce livre, les méthodes les plus utiles de Python 3 vous sont présentées. La liste exhaustive se trouve sur le site suivant :

`https://docs.python.org/fr/3/`

Des vidéos vous sont aussi proposées, exposant des explications détaillées sur le fonctionnement de certains algorithmes (voir la liste complète en fin d'ouvrage).

Même si l'informatique s'appuie sur une base théorique, nous insistons sur l'importance fondamentale de la pratique : chaque élève doit être capable de prendre des initiatives pour construire ses propres programmes, se poser des questions et ainsi les faire évoluer à sa guise.

Nombreux et variés, les exercices proposés dans cet ouvrage formeront un bon tremplin pour atteindre cet objectif.

Nous vous souhaitons à présent une bonne lecture et un bon apprentissage de cette matière passionnante, en constante évolution et pleine d'avenir.

Les auteurs

Table des matières

Méthodes de travail

Les conseils qui suivent ont été mis en pratique par un grand nombre de majors (sortis premiers) de Polytechnique ou de l'ÉNA, par des professionnels de l'organisation et sont également recommandés par de nombreux professeurs. Pour approfondir votre méthode de travail et obtenir les meilleurs résultats pendant vos études, nous vous recommandons la lecture du livre « Comment travailler plus efficacement », par F. Déliac, U. Hadrien, E. Matrullo et E. Maurette.

Faire des « feed-back »

Le « ***feed-back*** » est le conseil le plus important et le plus utilisé par ceux qui réussissent brillamment leurs études. Il consiste à contrôler systématiquement, sans s'aider de notes, ce que l'on vient d'apprendre (exercices et cours). Ce contrôle peut se faire mentalement, oralement ou par écrit.

- Dans les transports, essayez de vous rappeler mentalement, et sans vous aider de vos notes, le cours et les exercices vus le matin en classe (feed-back mental).
- Après avoir relu votre cours le soir, essayez de retrouver par écrit les principaux paragraphes et démonstrations sans regarder votre leçon (feed-back écrit).
- Après avoir résolu un problème, prenez 5 minutes pour contrôler par écrit que vous vous rappelez clairement l'énoncé ainsi que la démarche de résolution (feed-back écrit).
- Expliquez à des amis la leçon que vous venez d'apprendre ou l'exercice que vous venez de résoudre : c'est un excellent feed-back oral. Choisissez le type de « feed-back » qui vous convient le mieux et faites-en le plus régulièrement possible (après chaque cours et chaque série d'exercices). Pour être efficace, un « feed-back » doit se faire sans l'aide de vos notes. Ainsi, faire des fiches de résumés de cours à partir de vos cahiers ouverts ne constitue nullement un « feed-back ».

Miser sur la qualité

De nombreux témoignages démontrent que pour obtenir de bons résultats, il est préférable de faire un nombre limité d'exercices de manière approfondie plutôt que d'en survoler un grand nombre. Une tendance très répandue consiste à abattre une grande quantité d'exercices, à la chaîne, mais superficiellement, en espérant que le jour du contrôle, on aura déjà vu ce type de problème et que l'on saura s'en souvenir. Cette méthode est absolument inefficace car la seule manière de se souvenir d'un exercice de mathématiques ou de physique, c'est de l'avoir parfaitement compris et assimilé. Ainsi :

- À la fin d'un problème, prenez 5 à 10 minutes pour essayer de trouver un moyen de le généraliser ou de le compliquer (c'est ce que font souvent les professeurs pour concevoir leurs contrôles écrits) ; trouvez ce que cela pourrait changer dans la solution.

- Prenez également l'habitude, après chaque exercice, de faire un « feed-back » en faisant ressortir la démarche générale et en tissant des liens avec le cours. Bref, il ne faut pas vous contenter de résoudre l'exercice, mais il vous faut lui apporter de la valeur ajoutée et vous interroger sur son contenu.
- *Idem* pour le cours. Ne vous contentez pas de le parcourir de manière passive. Il vous faut avoir la rigueur d'effacer toutes les zones d'ombre. Pour chaque théorème, il faut vous demander quels types d'exercices son utilisation permettra de résoudre.

Travailler par « couches successives »

Cette méthode, très utile pour les étudiants préparant des examens ou des révisions, peut également être utilisée dès le lycée.

On observe que pour apprendre un gros volume de cours, rien n'est plus inefficace que de l'attaquer de front, de manière linéaire. La bonne manière consiste à d'abord survoler l'ensemble, en ne retenant que la structure, c'est-à-dire les grands titres, ainsi que les noms des paragraphes (première couche, étape devant durer 5 minutes). Dans l'étape suivante (deuxième couche, d'une durée de 10 minutes), on reprend son cours du début en retenant cette fois également les théorèmes et résultats importants. Après cette deuxième couche, on a déjà une idée claire de la structure de l'ensemble du cours. On peut alors aborder la dernière étape (troisième couche) : on reprend son cours au début pour, cette fois-ci, l'étudier en profondeur en apprenant le détail des démonstrations.

Il est à noter que cette méthode peut être également appliquée avec succès à des matières littéraires, ainsi qu'aux révisions du bac de français. Par exemple :

- Pour la préparation d'un contrôle, on commencera par passer en revue rapidement l'ensemble du cours et des exercices du chapitre précédemment étudié, avant de les réviser en détail. Ainsi aura-t-on développé une compréhension synthétique et claire.
- De même, avant d'aborder un problème volumineux (tel qu'un contrôle écrit), il est préférable d'en survoler l'ensemble avant de l'attaquer.

Travailler sa rapidité

Pour acquérir de la rapidité, trois voies sont possibles :

- Prenez l'habitude, en travaillant chez vous, de vous concentrer sur une seule chose à la fois, c'est-à-dire ne pas attaquer un problème ou une dissertation, en rêvassant à ce que vous pourriez trouver à manger dans le réfrigérateur ou en écoutant de la musique.
- Prenez l'habitude de travailler chez vous dans les mêmes conditions qu'en devoirs surveillés. Le minutage de chacun des exercices de ce livre est fait en ce sens. Cependant, cela ne devrait pas, une fois la résolution faite, vous empêcher d'y réfléchir plus calmement afin de vérifier la bonne assimilation du problème. Si les seuls moments où vous vous pressez sont les contrôles écrits, vous ne deviendrez jamais rapide.

- Essayez de contenir tout votre travail à la maison dans une plage horaire serrée. Engagez-vous, par exemple, à travailler chez vous tous les jours entre 18 h et 20 h et efforcez-vous de ne jamais déborder (quelle que soit votre charge de travail). En effet, si l'on ne se donne pas de limite de temps pour accomplir un travail, on a naturellement tendance à le laisser traîner en longueur et à rêvasser. L'étroitesse de la plage horaire vous obligera à ne pas vous endormir et à devenir efficace.

Spécial Baccalauréat

L'épreuve terminale obligatoire de spécialité est composée de deux parties : une partie écrite et une partie pratique, chacune notée sur 20. La note de la partie écrite a un coefficient de 0,75 et celle de la partie pratique a un coefficient de 0,25. La note globale de l'épreuve est donnée sur 20 points. Elle est à traiter en 3h30.

- Les exercices peuvent porter sur plusieurs notions (par exemple, bases de données relationnelles et programmation Python).
  Chaque exercice commence par une mention « Cet exercice porte sur . . . ».
  À l'aide de cette mention, vous pouvez choisir de commencer à traiter l'exercice qui porte sur vos points forts.
- L'épreuve écrite est composée de 3 exercices obligatoires : deux sur 6 points et un sur 8 points.
- Si vous êtes habitués à utiliser la méthode du feed-back exposée dans les premières pages de cet ouvrage, il est bon d'envisager un court feed-back mental sur les parties du cours concernées par l'épreuve du jour. Si vous êtes nerveux, cela vous aidera à vous mettre en confiance et à « démystifier » l'épreuve, que vous pourrez aborder avec la même sérénité qu'un devoir surveillé classique.
- Il faut absolument soigner la présentation. Au cours de l'année scolaire, votre professeur s'est habitué à votre style de rédaction, à votre écriture. Il sait ce que vous valez. Ce n'est pas le cas du correcteur qui lira votre copie. Il n'aura ni l'indulgence que pourrait avoir votre professeur, ni la patience d'essayer de vous déchiffrer si votre copie est présentée comme un brouillon. Par conséquent, il vous faut faire un réel effort de présentation le jour du bac, plus encore que pendant l'année scolaire. Vous devez notamment veiller à indiquer clairement le numéro des questions auxquelles vous répondez de façon à faciliter la correction de votre copie.
- Enfin, relisez-vous. Réservez-vous pour cela le temps nécessaire. Si vous vous laissez 60 minutes pour la résolution d'un exercice, il vous restera 10 minutes pour sa relecture. Celle-ci doit être très attentive (ce n'est pas toujours facile à l'issue de plus de trois heures d'épreuve). Enfin, souvenez-vous, si vous manquez de temps pour tout relire, qu'il vaut mieux une relecture partielle très attentive qu'une relecture « express » en diagonale, totalement inefficace pour détecter les erreurs résiduelles.

ECE
ÉCOLE D'INGÉNIEURS
ENGINEERING SCHOOL

Tu prévois quoi après ton Bac ?

Rejoins l'ECE :
l'école d'ingénieurs qui rend
l'Intelligence Artificielle indispensable

6 Campus Paris - Lyon - Bordeaux
Rennes - Marseille - Toulouse
3 Programmes Bachelor - Ingénieur - MSc

Retrouvez-nous sur

Inscris-toi
à nos prochains
événements

Chapitre 1

Structures indexées et dynamiques

Plan du chapitre

Exercice type — Lycée François Mauriac, Bordeaux

1 En Python, on considère une pile `P`.

(a) Quelle instruction permet d'empiler la valeur « 7 » dans `P` ?

(b) Quelle instruction permet de dépiler `P` ?

(c) Quelle instruction permet d'afficher les deux éléments au sommet de la pile sans les défiler ?

2 En Python, on considère une file `F`.

(a) Quelle(s) instruction(s) permet(tent) d'enfiler successivement les valeurs « 2 », « 7 » et « 3 » dans `F` ?

(b) Quelle instruction permet de défiler `F` ?

(c) Quelle instruction permet d'afficher l'élément en tête de file sans le dépiler ?

Voir corrigé page 7

1 Structures indexées

1.1 Notion de structure

À l'heure du *big data*, les quantités de données à traiter en informatique sont de plus en plus importantes. Le traitement de ces données (*data* en anglais) nécessite donc une organisation structurée et efficace. Plus les données seront structurées, plus leur traitement sera simple, et plus le risque d'erreur sera limité.

Exemple : un agenda contenant des personnes identifiées par leurs noms, prénoms, date de naissance adresse et numéro de téléphone est une structure de données.

COURS

Définition 1

Une *structure de données* est une manière d'organiser des données afin de pouvoir les traiter de façon simplifiée.

1.2 Structures indexées

1.2.1 - Tableaux

Définition 2

Un *tableau* est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder par leur position dans la séquence, appelée *indice*.

L'indexation d'un tableau de n éléments commence à 0 (premier élément) et se termine à $n-1$ (dernier élément).

Exemple :
En Python, on peut définir un tableau de 5 entiers de la manière suivante :

```
tab = [ 7 , 12 , 45 , 0 , 12 ]
```

Ceci est un tableau unidimensionnel. Il existe aussi des tableaux multidimensionnels **dont chaque élément est lui-même un tableau**.

Exemple : la matrice :

$$\begin{pmatrix} 1 & 0 & 3 \\ 5 & -2 & 8 \end{pmatrix}$$

peut-être représentée en Python par le tableau suivant :

```
tab = [
        [ 1 , 0 , 3 ]
        [ 5 , -2 , 8 ]
      ]
```

Ici, `tab` est un tableau dans lequel chaque élément est un tableau de trois entiers.

Un tableau se caractérise par :

- le type des éléments qu'il contient (entiers, caractères, tableaux,...) ;
- le nombre d'éléments qu'il contient (sa *taille*).

Exemple : dans cet exemple écrit en C, impossible de définir `tab[3]` car la taille du tableau a été fixée à 3 éléments uniquement.

```
double tab[3];
tab[0] = 8.7;
tab[1] = 2.3;
tab[2] = 1.5;
```

1.2.2 - Les dictionnaires

Définition 3

Un *dictionnaire*, aussi appelé *tableau associatif*, est une structure de données dans laquelle des *clés* sont associées à des *valeurs*.

Remarque : en mathématiques, l'équivalent du dictionnaire est une *application*, qui associe à chaque valeur de la variable une autre valeur.

Exemple :

En Python

```
capitale = { "la France" : "Paris",
             "l'Angleterre" : "Londres",
             "l'Allemagne" : "Berlin"}

for cle in capitale:
    print("La capitale de {} est {}.".format(cle,
  capitale[cle]))
```

Ce programme affiche :

```
La capitale de la France est Paris.
La capitale de l'Angleterre est Londres.
La capitale de l'Allemagne est Berlin.
```

2 Structures dynamiques

2.1 Listes

Définition 4

Une *liste* est une structure de données regroupant des éléments ordonnés qu'il est possible de manipuler individuellement (accès, ajout, suppression et modification).

Un exemple de définition et de manipulation de listes :

En Python

```
maliste = []
maliste.append(7.2)
maliste.append(9.7)
```

COURS

Remarques

- La taille d'un tableau est statique, contrairement à celle d'une liste.
- Le langage Python permet de confondre les deux notions car la notion de tableau n'est pas à proprement parler implémentée dans ce langage. Ainsi, quand on programme en Python, on parlera de tableaux ou de listes indifféremment.

2.2 Les piles

Retrouvez une explication des piles et des files en vidéo.

2.2.1 - Définition

Définition 5

Une *pile* est une liste remplie sur le principe du « dernier arrivé, premier sorti », abrégé en *LIFO*, de l'anglais « Last In, First Out ».

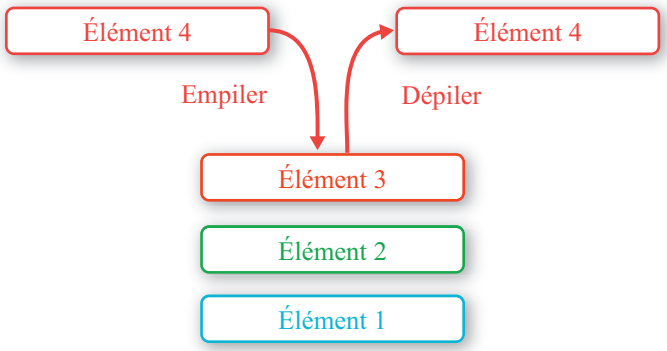


Figure 1.1 – Illustration d'une pile

Quand on ajoute un élément à une pile, on dit que l'on *empile* ; quand on retire un élément d'une liste, on dit que l'on *dépile*.

Exemples

- Dans un navigateur web, une pile sert à mémoriser les dernières pages visitées : la première à apparaître est la dernière à avoir été visitée (accessible en cliquant sur l'icône « Page précédente » par exemple).
- Dans un traitement de texte, la combinaison de touches « [Ctrl] + [Z] » annule la dernière action : cela repose sur le fait que les actions sont empilées.

2.2.2 - Les piles en Python

Dans les langages de programmation, il existe souvent de multiples moyens de représenter une même structure de données. Ainsi, en Python, on peut par exemple implémenter une pile à l'aide d'une liste dont le début représente le bas de la pile et la fin le sommet de celle-ci.

- `p.append(v)` ajoute `v` au sommet de la pile (à la fin de la liste) ;
- `p[-1]` est l'élément au sommet de la pile (c'est le dernier élément de la liste) ;
- `p.pop()` retire l'élément au sommet de la pile (le dernier élément de la liste). indexPython !pop() [pile]

Exemple :

```
pile = [] # pile vide
pile.append(3) # empile la valeur '3'
pile.append(7) # empile la valeur '7'
print(pile[-1]) # affiche : 7, le sommet de la pile
pile.pop()
print(pile) # affiche : [ 3 ]
```

2.3 Les files

2.3.1 - Définition

Définition 6

Une *file* est une suite ordonnée d'éléments remplie sur le principe du « premier arrivé, premier sorti », abrégé en *FIFO*, de l'anglais « First In, First Out ». Quand on ajoute un élément à une file, on dit que l'on *enfile* ; quand on retire un élément d'une file, on dit que l'on *défile*.

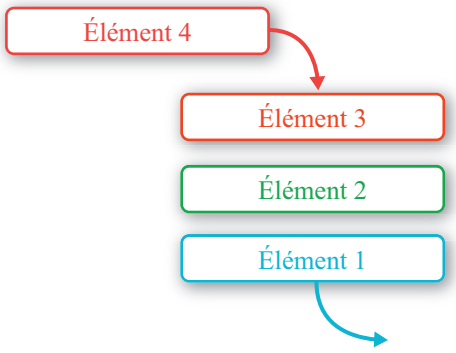


Figure 1.2 – Illustration d'une file

Exemple : lorsque l'on imprime plusieurs documents, les requêtes d'impression sont traitées selon une file. C'est le même principe qu'une caisse au supermarché.

COURS

2.3.2 - Les files en Python

Les listes sont mal adaptées pour représenter en Python des files car l'ajout d'élément se fait toujours en fin de liste, et non en début.

C'est la raison pour laquelle on pourra utiliser la structure de donnée `deque` (*double ended queue*) du module `collections`, que l'on pourra construire ainsi :

`file = deque([ liste de nombres séparés par des virgules ])`
ou : `file = deque((liste de nombres séparés par des virgules))`

Ainsi,

`file = deque( (3,2,1) )`

représente la file dont le premier élément (la tête de file) est « 1 » et dont le dernier élément enfilé est « 3 ».

On utilisera alors :

- `file.appendleft(élément)` pour enfiler un élément ;
- `file.pop()` pour défiler la tête de file. À noter toutefois qu'en mode console, écrire « `f.pop()` » affiche l'élément défilé, ce qui n'est pas le cas dans un programme.

Exemple :

```
from collections import deque
f = deque((3,2,1)) # initialise la file
f.appendleft(4) # ajoute la valeur 4
print(f) # affiche la file
print(f.pop()) # affiche la tête de file et défile
print(f)
```

À RETENIR

Tableaux	Séquence finie et de taille fixe d'éléments auxquels on peut accéder par leur indice.
Dictionnaires	Association dynamique de paires clé/valeur.
Liste	Groupe d'éléments ordonnés que l'on peut manipuler individuellement.
Pile	Liste remplie sur le principe « dernier arrivé, premier sorti » (LIFO).
File	Liste remplie sur le principe « premier arrivé, premier sorti » (FIFO).

Solution de l'exercice type

Lycée François Mauriac, Bordeaux

1 En Python une pile est représentée par une liste.

(a) Empiler une valeur revient à ajouter cette valeur à la fin de la liste.
L'instruction qui permet d'empiler la valeur « 7 » dans `P` est alors :

```
P.append(7)
```

(b) Dépiler `P` revient à enlever la dernière valeur de la liste.
Ainsi, l'instruction qui permet de dépiler `P` est :

```
P.pop()
```

(c) Les deux éléments au sommet de la pile sont les deux derniers éléments de la liste.
Ainsi, l'instruction qui permet d'afficher d'abord le 2e puis le 1er est :

```
P[-2] , P[-1]
```

2 En Python, une file `F` est aussi représentée par une liste.

(a) Pour enfiler une valeur dans `F`, on peut utiliser `deque` du module `collections`.
Dans ce cas, on pourra utiliser les instructions suivantes :

```
F = deque()
for i in [2,7,3]:
    F.appendleft(i)
```

À ce stade, `F = deque([3, 7, 2])`.

(b) Défiler `F` signifie enlever l'élément « le plus ancien » de la file, à savoir ici « 2 ».
Il s'agit du dernier élément de l'objet `F` (on ne peut pas réellement dire ici que `F` est une liste car quand on affiche `F`, ce n'est pas exactement une liste que l'on voit).
Ainsi, pour défiler, on utilise l'instruction :

```
>>> F.pop()
```

(c) Comme pour la pile, l'instruction qui permet de connaître ou de retrouver l'élément en tête de la file `F` est :

```
>>> F[-1]
```

Voir énoncé page 1

INTERROS

1 QCM Vérification de connaissances

5 min Corrigé p. 15

Pour chacune des questions suivantes, plusieurs propositions de réponses sont faites dont une seule est correcte. Laquelle ?

1 On souhaite insérer dans une structure de données les différentes hauteurs d'un arbre au fil des années sans y insérer les années. La structure de données la plus adaptée en Python est :

[a] une liste ;
[b] un dictionnaire ;
[c] une pile ;
[d] une file.

2 Dans une administration, on souhaite insérer dans une structure de données les doléances reçues afin de les traiter selon leur ordre d'arrivée. On aura alors plutôt intérêt à les enregistrer dans :

[a] une liste ;
[b] un dictionnaire ;
[c] une pile ;
[d] une file.

3 Dans une association, on souhaite enregistrer une liste d'informations pour chaque adhérent : taille, poids, etc. La meilleure structure de données pour cela est :

[a] une liste ;
[b] un dictionnaire ;
[c] une pile ;
[d] une file.

4 Pour un logiciel, on doit enregistrer la liste des actions dans une structure de données de sorte à voir la dernière action en priorité. On enregistrera donc les actions dans :

[a] une liste ;
[b] un dictionnaire ;
[c] une pile ;
[d] une file.

2 Fabricant de tee-shirts ★

5 min Corrigé p. 15

Lycée polyvalent Émile Zola, Aix-en-Provence

Un fabricant décide de créer des tee-shirts dont la taille peut-être : XS, S, M, L, XL, XXL. À chaque taille son prix ; il adopte le principe suivant : 8 € pour la taille XS et il ajoute 2 € en passant à la taille supérieure, jusqu'au XXL.

1 Implémenter en Python ces informations dans la structure de données la mieux adaptée.

2 Ce même fabricant décide de changer sa façon de fixer le prix de revente des tee-shirts. Ceux dont la taille est XS sont toujours à 8 €, mais cette fois-ci, pour passer d'une taille à la suivante, il ajoute au prix de la taille inférieure la moitié de sa racine carrée.

Par exemple, pour obtenir le prix des tailles S, il fait :

$$8 + \sqrt{8} \div 2 \approx 9{,}41.$$

Proposer un programme Python qui automatise ces calculs et les enregistre dans une structure de données adaptée.

3 Séparation opérations / nombres

★ — 5 min — Corrigé p. 16

Lycée Camille Jullian, Bordeaux

On considère une chaîne de caractères représentant une expression arithmétique `E` contenant des chiffres et les opérations élémentaires (+, -, *, et /).

Écrire une fonction Python `separe(E)` qui renvoie deux listes : la première, nommée **`nombres`**, contenant les chiffres de `E` et la seconde, nommée **`operations`**, contenant les opérations de `E`.

Par exemple, si `E` est la chaîne '3 + 5 * 7 - 2' alors `nombres = [3,5,7,2]` et `operations = ['+','*','-']`.

4 Nombre de parenthèses

★ — 10 min — Corrigé p. 17

Lycée François Mauriac, Bordeaux

On considère une expression arithmétique qui peut contenir des parenthèses.

1. Écrire un algorithme permettant de vérifier qu'il y a autant de parenthèses ouvrantes que de parenthèses fermantes.
2. Implémenter en Python une fonction `controle(E)` qui renvoie « True » si l'expression `E` comporte autant de parenthèses ouvrantes que fermantes, et « False » sinon.

5 Bon parenthésage ?

★ — 10 min — Corrigé p. 17

Lycée Sophie Germain, Paris

Une expression (algébrique ou arithmétique) est correctement parenthésée si d'une part, le nombre de parenthèses (et crochets) ouvrant.e.s et fermant.e.s est le même et si d'autre part les correspondances ne se croisent pas.

Par exemple, l'expression « $\big[3 + (5 - 7] * 3\big)$ » n'est pas correctement parenthésée car la parenthèse fermante ne peut pas venir après le crochet fermant.

1. Écrire un algorithme qui s'aide d'une pile pour contrôler le bon parenthésage d'une expression.
2. Écrire en Python une fonction `is_goodpar(E)` qui renvoie « True » si l'expression `E` est correctement parenthésée, et « False » dans le cas contraire.

INTERROS

6 Implémentation d'une file

★ 15 min Corrigé p. 18

Lycée Turgot, Paris

Le module `collections`, avec `deque`, permet d'implémenter une file. Ainsi, les instructions :

```
>>> f = deque()
>>> f.appendleft(1)
>>> f.appendleft(2)
>>> print(f)
```

enfile la valeur « 1 » puis la valeur « 2 », et affiche : « `deque([2,1]` », qui représente la file.

Écrire une fonction `enfile(liste,tuple)` qui admet pour arguments une liste `liste` (qui nous sert de file) et un tuple regroupant les éléments à enfiler dans l'ordre que l'on veut.

Par exemple,

```
>>> f = []
>>> f = enfile( f , (1,2) )
>>> f = enfile( f , 3)
```

retourne la liste `[3,2,1]` (on a enfilé « 1 », puis « 2 », et enfin « 3 »).

7 D'une liste à un dictionnaire

★ 15 min Corrigé p. 19

Lycée François Villon, Les Mureaux

Écrire en Python une fonction `list_to_dico(liste)` qui admet en argument une liste de la forme `[clé1,valeur1,clé2,valeur2,...]` et qui retourne le dictionnaire correspondant de la forme :

`{ clé 1 : valeur 1, clé 2 : valeur 2, ... }`.

8 Chiffrement

★★ 15 min Corrigé p. 20

Lycée Sainte-Anne, Brest

La fonction `ord(<caractère>)` renvoie le point de code Unicode du caractère spécifié. Par exemple, « `ord('f')` » renvoie la valeur 102.

La fonction inverse est `chr(<entier>)`. Par exemple, « `chr(102)` » renvoie « f ».

Pour calculer a modulo n, on utilise la syntaxe « a % n ».

Pour chiffrer un mot M, on décide de procéder ainsi : pour chaque caractère c de M,

- on calcule $3 \times \text{ord}(c) - 61$ modulo 91, puis on ajoute 33. Le résultat est noté r ;
- on trouve ensuite le caractère correspondant à r avec `chr(r)` : on note ce caractère c_1.
- si r est pair, on calcule $2 \times \text{ord}(c) - 31$ modulo 91 puis on ajoute 33, et on trouve le caractère correspondant à ce résultat, noté c_2. Dans ce cas, le caractère c est chiffré en c_1c_2 ;
- si r est impair, on calcule $2 \times \text{ord}(c) - 32$ modulo 91 puis on ajoute 33, que l'on transforme en son caractère c_2, puis on calcule $3 \times \text{ord}(c) - 60$ modulo 91 puis on ajoute 33, que l'on transforme en son caractère c_3. Le caractère c est alors chiffré en $c_1c_2c_3$.

Par exemple, pour chiffrer « P » :

- `ord('P') = 80`, donc on calcule :

$$3 \times 80 - 61 \mod 89 = 179 \mod 91 = 88,$$

  puis on ajoute 33 :

$$r = 88 + 33 = 121,$$

  qui est impair et qui correspond à `chr(121)` = « y ». Donc c_1 correspond au caractère : « y » ;
- r est impair donc on calcule :

$$2 \times 80 - 32 \mod 91 = 37,$$

  puis on ajoute 33 :

$$37 + 33 = 70,$$

  qui correspond à $c_2 = \text{chr}(70) =$ « F » ;
- ensuite, on calcule :

$$3 \times 80 - 60 \mod 91 = 89$$

  puis on ajoute 33 :

$$89 + 33 = 122$$

  qui correspond à $c_3 = \text{chr}(122) =$ « z » ;
- « P » est donc chiffré en « yFz ».

Afin d'optimiser le temps de cryptage, on souhaite implémenter une structure de données nous permettant d'obtenir le cryptage de tous les caractères dont le point de code Unicode est compris entre 32 et 122 compris.

1. Implémenter la structure de données adéquate.
2. Chiffrer le message : « informatique ».

9 Labyrinthe

★★ 15 min Corrigé p. 21

Lycée Camille Claudel, Blois

Une manière d'implémenter un labyrinthe rectangulaire est de découper le rectangle en cases, puis de définir pour chaque case s'il y a un mur en haut, en bas, à gauche et à droite avec des booléens.

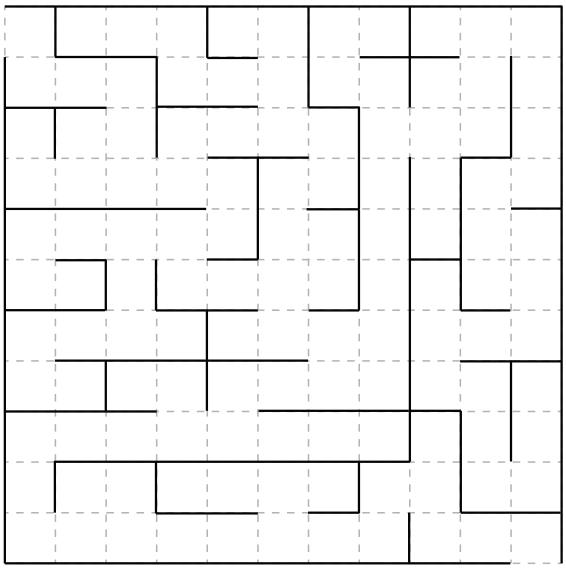


Figure 1.3 – Labyrinthe à implémenter en Python

Utiliser une structure de données adaptée afin d'implémenter ce labyrinthe selon cette façon de concevoir les choses.

On ne demande pas d'implémenter tout le labyrinthe, mais d'expliquer comment on ferait, en donnant l'exemple de la première case (en haut à gauche).

10 Dictionnaire et recettes

★★ 15 min Corrigé p. 21

Lycée Henri Alain-Fournier, Bourges

Vous souhaitez créer une petite application pour gérer une collection de recettes de cuisine. Chaque recette a un nom, une liste d'ingrédients, et des instructions de préparation. Vous devez permettre aux utilisateurs d'ajouter de nouvelles recettes, de rechercher une recette par son nom, et d'afficher les détails d'une recette.

Votre objectif est d'utiliser un dictionnaire pour stocker les recettes et implémentez les fonctions nécessaires pour manipuler cette collection.

Instructions :

1. Créez un dictionnaire appelé `recettes` où chaque clé est le nom de la recette et chaque valeur est un autre dictionnaire contenant les informations suivantes :
   - `ingredients` : Une liste des ingrédients nécessaires.
   - `instructions` : Une chaîne de caractères décrivant les étapes de préparation.
2. Implémentez les fonctions suivantes :
   - `ajouter_recette(recettes, nom, ingredients, instructions)` : Ajoute une nouvelle recette au dictionnaire.
   - `rechercher_recette(recettes, nom)` : Recherche une recette par son nom et retourne ses détails.
   - `afficher_recettes(recettes)` : Affiche les noms de toutes les recettes disponibles.
3. Testez votre application en ajoutant quelques recettes, en recherchant une recette spécifique, et en affichant la liste des recettes.

11 Gestion d'une bibliothèque

★★ 20 min Corrigé p. 22

Lycée François Mauriac, Bordeaux

Vous devez concevoir un système de gestion pour une bibliothèque. La bibliothèque possède plusieurs livres, et chaque livre a des informations associées telles que le titre, l'auteur, l'année de publication, et le nombre de copies disponibles. Vous devez permettre aux utilisateurs d'ajouter des livres, d'emprunter des livres, de rendre des livres, et de vérifier la disponibilité des livres.

Votre objectif est d'implémenter un système de gestion de bibliothèque en utilisant des dictionnaires imbriqués pour stocker les informations sur les livres et les opérations associées.

Instructions :

1. Créez un dictionnaire principal appelé `bibliotheque` où chaque clé est un identifiant unique pour un livre (par exemple, un ISBN ou un numéro d'identification).
2. Chaque valeur dans le dictionnaire principal doit être un autre dictionnaire contenant les informations suivantes pour chaque livre :
   - `titre` : Le titre du livre.
   - `auteur` : L'auteur du livre.
   - `annee` : L'année de publication du livre.
   - `copies_disponibles` : Le nombre de copies disponibles pour l'emprunt.

3 Implémentez les fonctions suivantes :

- `ajouter_livre(bibliotheque, identifiant, titre, auteur, annee, copies)` : Ajoute un nouveau livre à la bibliothèque.
- `emprunter_livre(bibliotheque, identifiant)` : Emprunte un livre si des copies sont disponibles.
- `rendre_livre(bibliotheque, identifiant)` : Rend un livre emprunté, augmentant ainsi le nombre de copies disponibles.
- `verifier_disponibilite(bibliotheque, identifiant)` : Vérifie combien de copies d'un livre sont disponibles.

4 Testez votre système en ajoutant plusieurs livres, en empruntant et en rendant des livres, et en vérifiant la disponibilité.

12 Expressions postfixées

★★★ 20 min Corrigé p. 23

Lycée Henri Alain-Fournier, Bourges

On ne considère dans cet exercice que les expressions numériques contenant les quatre opérations élémentaires (addition, soustraction, multiplication et division).

La notation postfixée consiste à mettre les opérations arithmétiques après leurs deux arguments. Ainsi, l'opération $a \bigstar b$, où $\bigstar \in \{+; -; \times; \div\}$, a pour notation postfixée : $ab\bigstar$.

1er exemple : l'expression 3 * ((4 + 5) + 6) peut s'écrire « 3 4 5 + 6 + * » (mais aussi : « 4 5 + 6 + 3 * » ou encore « 3 6 4 5 + + * »).

2e exemple : (4 + (5 + 6)) * 3 peut s'écrire « 4 5 6 + + 3 * ».

1 Écrire un algorithme utilisant une pile pour effectuer une opération en notation postfixée.

2 Implémenter l'algorithme en Python en définissant une fonction `calcul(E)`, où `E` est l'expression en notation postfixée dans laquelle chaque nombre ou symbole est séparé par une espace.

1 QCM Vérification de connaissances

Énoncé p. 8

1 Réponse [a]. Quand on observe l'évolution d'une quantité numérique, ce n'est en général pas pour sortir le premier ou le dernier élément en premier, donc pile et file sont à exclure.

De plus, il n'y a pas d'association donc un dictionnaire est impossible. Seule la liste convient pour ce genre de problème.

2 Réponse [d]. On enregistre les doléances selon le principe : « première arrivée, première traitée » (« first in, first out », donc FIFO) ce qui correspond à une file.

3 Réponse [b]. Il s'agit ici d'enregistrer des associations, c'est-à-dire des éléments qui sont rattachés à d'autres (adhérents → informations) donc le dictionnaire est le plus adapté. Ici, les clés seront les adhérents et les valeurs seront les informations.

4 Réponse [c]. Il s'agit en effet d'enregistrer les actions selon le principe : « dernière action faite, première action affichée » (« last in, first out », donc LIFO), ce qui correspond à une pile.

À RETENIR

- Liste : succession de données accessibles sans contrainte.
- Pile (LIFO) : on imagine une pile de pièces où la dernière mise est la première à pouvoir être prise.
- File (FIFO) : on peut penser aux files d'attente, où la première personne arrivée est la première à être servie.

2 Fabricant de tee-shirts

Énoncé p. 8

Lycée polyvalent Émile Zola, Aix-en-Provence

Comme toujours, quand il s'agit d'associer deux entités (ici, taille et prix), on utilisera un dictionnaire.

1 On implémente le dictionnaire ainsi :

```
prix = {
    'XS': 8, 'S': 10, 'M': 12, 'L': 14, 'XL': 16,
    'XXL': 18
}
```

2 Dans cette question, il faut imaginer un programme qui effectue les calculs au fur et à mesure. On peut donc par exemple stocker les tailles dans une liste, puis créer le dictionnaire élément par élément à partir de cette liste comme dans la proposition page suivante.

CORRIGÉS

```
dico = {}
tailles = ['XS', 'S', 'M', 'L', 'XL', 'XXL']

for i in range(len(tailles)):
    if i == 0:
        dico[ tailles[i] ] = 8
    else:
        # round(a,n) : arrondi du nombre 'a' à n
    chiffres après la virgule
        dico[ tailles[i] ] = round(dico.get(tailles
    [i-1]) + dico.get(tailles[i-1]) ** 0.5 / 2, 2)

print(dico)
```

Ce qui donne :

```
{ 'XS': 8, 'S': 9.41, 'M': 10.94, 'L': 12.59,
  'XL': 14.36, 'XXL': 16.25 }
```

Remarque : la racine carrée d'un nombre x peut s'écrire `x ** 0.5` car $\sqrt{x} = x^{1/2}$.

3 Séparation opérations / nombres

Énoncé p. 9

Lycée Camille Jullian, Bordeaux

Il suffit ici de parcourir l'expression `E` est d'effectuer des tests pour savoir si le caractère rencontré est un nombre (avec la méthode `isdigit` par exemple), une opération, ou autre chose (en l'occurrence, une espace).

```
def separe(E):
    nombres, operations = [], []
    for i in E:
        if i.isdigit():
            nombres.append(i)
        if i in '+-*/':
            operations.append(i)
    return nombres, operations

print(separe('3+5*9-6'))
```

4 Nombre de parenthèses

Énoncé p. 9

Lycée François Mauriac, Bordeaux

1 Il s'agit ici de compter le nombre de parenthèses ouvrantes et fermantes. Rien de bien compliqué.

```
E est une expression arithmétique
f ← 0 (nombre de parenthèses fermantes)
o ← 0 (nombre de parenthèses ouvrantes)
Pour chaque élément i de E:
   Si i = "(":
     o ← o + 1
   Sinon:
     Si i = ")":
       f ← f + 1
   Fin du Si
   Si f = o:
     Renvoyer True
   Sinon:
     Renvoyer False
```

2 Une implémentation possible est alors :

```
def controle(E):
    o, f = 0, 0
    for i in E:
        if i == '(':
            o += 1
        if i == ')':
            f += 1
    return o == f
```

5 Bon parenthésage ?

Énoncé p. 9

Lycée Sophie Germain, Paris

1 Il s'agit ici d'empiler les crochets et parenthèses ouvrant.e.s et de dépiler quand on rencontre son équivalent fermant.

```
P est une pile vide
E est une expression
Pour chaque élément e de E:
   Si e = "(" ou e = "[":
     e est empilé
```

...

```
   Si e = ")":
     Si la tête de P est "(":
       On dépile "("
     Sinon:
       Renvoyer False
     Fin du Si (tête de P)
   Fin du Si e = ")"
   Si e = "]":
     Si la tête de P est "]":
       On dépile "["
     Sinon:
       Renvoyer False
     Fin du Si (tête de P)
   Fin du Si e = "]"
Si P est vide:
   Renvoyer True
```

2 Une implémentation de cet algorithme en Python est :

```python
def is_goodpar(E):
    pile = []
    for e in E:
        if e == '(':
            pile.append(e)
        elif e == ')':
            if len(pile) != 0:
                if pile[-1] == '(':
                    pile.pop()
                else:
                    return False
            else:
                return False

    if len(pile) == 0:
        return True
```

6 Implémentation d'une file

Énoncé p. 10

Lycée Turgot, Paris

Une implémentation possible est celle proposée page suivante.

Ainsi, on aura par exemple :

```
>>> f = [2,1] # On initialise la file comme une liste
>>> f = enfile(f,(3,4)) # On enfile 3, puis 4
```

```
def enfile(liste,t):
    # on suppose que 't' est un tuple
    file = [i for i in liste]
    for i in t:
        file = [i] + file
    return file

f = [2,1]
f = enfile(f,(3,4))
print(f)
```

7 D'une liste à un dictionnaire

Énoncé p. 10

Lycée François Villon, Les Mureaux

Il s'agit ici de parcourir la liste donnée en argument en sautant un élément sur deux afin de mettre l'élément sauté en valeur.

Une fonction possible est alors la suivante :

```
def liste_to_dico(liste):
    d = {}
    for i in range(0,len(liste),2):
        d[ liste[i] ] = liste[i+1]

    return d
```

Ce qui donne par exemple :

```
>>> L = [ 'France' , 'Paris' ,
... 'Angleterre' , 'Londres' ,
... 'Espagne' , 'Madrid' ,
... 'Allemagne' , 'Berlin' ]
>>> print( liste_to_dico(L) )
{'France': 'Paris',
'Angleterre': 'Londres',
'Espagne': 'Madrid',
'Allemagne': 'Berlin'}
```

CORRIGÉS

8 Chiffrement

Énoncé p. 10

Lycée Sainte-Anne, Brest

1 **MÉTHODE**

On peut ici créer un dictionnaire où les clés seront les caractères dont le point de code Unicode est compris entre 32 et 122 et où les valeurs seront les résultats des chiffrements.

On obtient une implémentation comme celle-ci :

```
crypt = {}

for i in range(32,123):
    r = (3 * i - 61) % 91 + 33
    c1 = chr(r)
    if r % 2 == 0:
        c2 = chr((2 * i - 31) % 91 + 33)
        crypt[ chr(i) ] = c1 + c2
    else:
        c2 = chr((2 * i - 32) % 91 + 33)
        c3 = chr((3 * i - 60) % 91 + 33)
        crypt[ chr(i) ] = c1 + c2 + c3
```

Ici, `crypt` est un dictionnaire auquel on pourra faire appel lors de tout chiffrement.

2 On peut utiliser le programme suivant :

```
def cryptage(M):
    r = ''
    for c in M:
        r = r + crypt[c]
    return r

M = "informatique"

print(cryptage(M))
```

On obtient alors le résultat suivant :

```
ixjx(`s{)!)/*u%vQhR/30ixj&.26]p^
```

9 Labyrinthe

Énoncé p. 12

Lycée Camille Claudel, Blois

MÉTHODE

On peut représenter la grille par une matrice `L`, où `L[i][j]` représente la case de la $(i+1)$-ième ligne en partant du haut (par exemple) et $(j+1)$-ième colonne en partant de la gauche (par exemple).
Ensuite, la valeur de chaque cellule est un dictionnaire dont les clés seront par exemple quatre booléens `top`, `bottom`, `left` et `right` représentant la présence ou l'absence de murs respectivement en haut, en bas, à gauche et à droite.

On aura ainsi :

```
M[0][0] = { 'top' : True , 'bottom' : False ,
            'left' : False , 'right' : True }
```

Cette implémentation a le mérite d'être claire, mais peu pratique car assez longue à faire. Ainsi, pour alléger, on peut opter pour une liste de booléens dans chaque cellule, en convenant de garder le même ordre (mais l'ordre peut différer selon la logique de la personne qui programme). On aura alors :

```
M[0][0] = [ True , False , False , True]
```

ce qui est bien plus simple et rapide à écrire, mais cette redondance introduit un risque de discordance des informations, souvent à l'origine de bugs dans la pratique.

Remarque : nous verrons plus tard qu'il existe d'autres façons, peut-être plus pratiques, de représenter un labyrinthe, notamment à l'aide d'une *classe* (voir le chapitre sur la Programmation Orientée Objet).

10 Dictionnaire et recettes

Énoncé p. 12

Lycée Henri Alain-Fournier, Bourges

Voici un programme répondant aux instructions :

```
# Initialisation du dictionnaire des recettes
recettes = {}

def ajouter_recette(recettes, nom, ingredients,
    instructions):
    recettes[nom] = {
        'ingredients': ingredients,
        'instructions': instructions
    }
```

```
def rechercher_recette(recettes, nom):
    return recettes.get(nom, "Recette non trouvée.")

def afficher_recettes(recettes):
    print("Recettes disponibles :")
    for nom in recettes:
        print(f"- {nom}")
```

En scannant le QR Code, vous aurez un exemple d'utilisation en plus du programme ci-dessus.

11 Gestion d'une bibliothèque

Énoncé p. 13

Lycée François Mauriac, Bordeaux

Voici un programme répondant aux instructions données (vous aurez en plus un exemple d'utilisation en scannant le QR Code) :

```
# Initialisation de la bibliothèque
bibliotheque = {}

def ajouter_livre(bibliotheque, identifiant, titre,
    auteur, annee, copies):
    bibliotheque[identifiant] = {
        'titre': titre,
        'auteur': auteur,
        'annee': annee,
        'copies_disponibles': copies
    }

def emprunter_livre(bibliotheque, identifiant):
    if identifiant in bibliotheque and bibliotheque[
    identifiant]['copies_disponibles'] > 0:
        bibliotheque[identifiant]['copies_disponibles']
     -= 1
        print(f"Livre emprunté: {bibliotheque[
    identifiant]['titre']}")
    else:
        print("Livre non disponible pour l'emprunt.")
```

```
def rendre_livre(bibliotheque, identifiant):
    if identifiant in bibliotheque:
        bibliotheque[identifiant]['copies_disponibles']
     += 1
        print(f"Livre rendu: {bibliotheque[identifiant
    ]['titre']}")
    else:
        print("Identifiant de livre invalide.")

def verifier_disponibilite(bibliotheque, identifiant):
    if identifiant in bibliotheque:
        print(f"Copies disponibles pour '{bibliotheque[
    identifiant]['titre']}': {bibliotheque[identifiant
    ]['copies_disponibles']}")
    else:
        print("Identifiant de livre invalide.")
```

12 Expressions postfixées

Énoncé p. 14

Lycée Henri Alain-Fournier, Bourges

1 Un algorithme possible est le suivant :

```
P est une pile
E est une expression en notation postfixée
Pour chaque élément e de E:
   Si e est un nombre:
     Empiler e dans P
   Sinon:
     Dépiler les deux premières valeurs v1 et v2 de P
     Effectuer l'opération e sur v1 et v2
     Empiler le résultat dans P
   Fin du Si
```

2 Une implémentation possible de cette fonction est celle présentée page suivante.

- On parcourt la liste créée à partir de l'expression E de gauche à droite : `E.split()`.
- Si on rencontre un nombre, on l'ajoute à la file.
- Sinon, on retire les deux derniers nombres de la file, on applique l'opérateur, et on ajoute le résultat à la file.
- À la fin de l'expression, le résultat final sera le seul élément restant dans la file.

CORRIGÉS

```python
def calcul(E):
    file = []
    for i in E.split():
        if i not in ['+','-','*','/']:
            # Si ce n'est pas une opération, l'
    ajouter à la file
            file.append(float(i))
        else:
            # Sinon, retirer les deux derniers
    nombres de la file
            b = file.pop()
            a = file.pop()

            # Appliquer l'opérateur
            if i == '+':
                result = a + b
            elif i == '-':
                result = a - b
            elif i == '*':
                result = a * b
            elif i == '/':
                result = a / b

            # Ajouter le résultat à la file
            file.append(result)

    # Le résultat final est le seul élément restant
     dans la file
    return file.pop()
```

Chapitre 2

Programmation : généralités

Plan du chapitre

Exercice type 1 — **Lycée polyvalent Emile Zola, Aix-en-Provence**

Le *problème de l'arrêt* est une démonstration par l'absurde. proposée par Alan Turing.
Supposons qu'il existe un algorithme H qui décide le problème de l'arrêt, c'est-à-dire que, quel que soit l'algorithme A qu'on lui passe en paramètre,

$$\begin{cases} H(A) \to 1 & \text{si } A \text{ s'arrête;} \\ H(A) \to 0 & \text{si } A \text{ ne s'arrête pas.} \end{cases}$$

Posons alors $\overline{H}$ l'algorithme admettant l'algorithme A en paramètre qui exécute $H(A)$ tel que :

$$\begin{cases} \text{si } H(A) = 1 \text{ alors une boucle infinie est créée ;} \\ \text{si } H(A) = 0 \text{ alors il s'arrête.} \end{cases}$$

Montrer que le problème de l'arrêt est indécidable.

Voir corrigé page 32

1 Introduction

En informatique, la programmation est l'art de créer des programmes, qui sont des suites d'instructions ordonnées destinées à être exécutées par une machine numérique. Pour cela, il est indispensable d'utiliser des langages, qui permettent à la machine de comprendre l'humain. Ces trois notions (machines, programmation et langages) ont évolué de manière parallèle. Commençons par un peu d'histoire...

COURS

2 Historique

1842	La comtesse Ada Lovelace crée des diagrammes pour la machine analytique de Charles Babbage, mathématicien anglais. Ils sont aujourd'hui considérés comme les premiers programmes informatiques.
1936	Le concept de programmation et de programme enregistré est émis par Alan Turing, mathématicien anglais, dans un article intitulé « *On Computable Numbers, with an Application to the Entscheidungsproblem : Proceedings of the London Mathematical Society* ». Lors de la seconde guerre mondiale, Turing contribue à déchiffrer le code secret utilisé par les Allemands pour chiffrer leurs messages avec la machine *Enigma*.
Années 1940	Les premiers ordinateurs sont créés. Ils fonctionnent à l'aide de cartes perforées en carton.
1954	Développement du premier système d'exploitation au MIT (Massachusetts Institute of Technology). Apparition des premiers assembleurs et du premier compilateur pour le langage Fortran (Formula Translator), premier langage de programmation de haut niveau.
1964	le langage BASIC rend l'informatique accessible aux étudiants non scientifiques
1970	Arrivée de la programmation structurée, permettant aux programmeurs de concevoir des programmes plus poussés.
1972	Dennis Richie et Ken Thompson, chercheurs aux Bell Labs, créent le langage C, permettant de développer le système d'exploitation multitâche multi-utilisateur UNIX.
1974	Vince Cerf et Bob Kahn inventent le protocole TCP/IP, qui servira de base à Internet
1981	arrivée de l'IBM-PC sous MS-DOS et développement de la micro-informatique en entreprise.
1983	Création du langage C++.
1989	Le programmeur néerlandais Guido van Rossum crée un langage de script multi-plateforme en s'inspirant des langages ABC, C, et d'UNIX. Fan des Monty Python, il nomme son invention *Python*. La première version publique sort en 1991 sous licence libre.
1994	Apparition du langage HTML (Hypertext Markup Language). Associé au navigateur web Netscape et au protocole www (World Wide Web), HTML rend le réseau Internet accessible à tous.
2014	Trois milliards d'internautes utilisent quotidiennement l'informatique sur un milliard de sites en ligne.

3 Différents types de langages

Au plus bas niveau, le processeur ne comprend que le langage machine, qui est une suite d'opérations codées en binaire, incompréhensibles pour un être humain normalement constitué. C'est pourquoi des langages de plus haut niveau ont été inventés, mais il est nécessaire de les traduire en binaire. Il existe pour cela trois méthodes, qui correspondent à trois types de langages de programmation.

- les langages *compilés*, dans lesquels le code source écrit par le programmeur est transformé en code binaire par un logiciel appelé *compilateur*.
  Exemples : langages C, C++, Pascal.

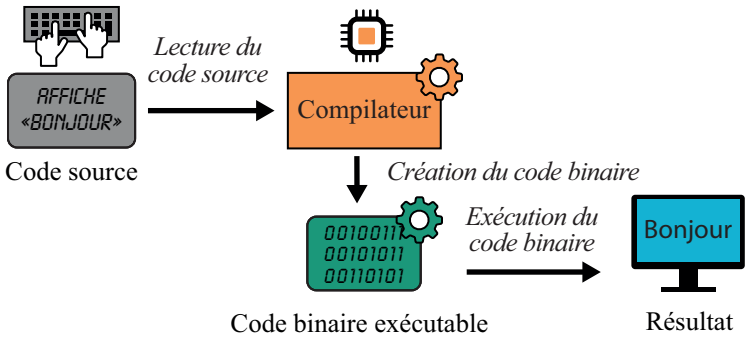


Figure 2.1 – Langage compilé

- les langages *interprétés*, dont le code source est utilisé en entrée d'un logiciel appelé *interpréteur*, qui l'exécute ligne par ligne en décidant à chaque étape ce qu'il va faire ensuite.
  Exemples : Python, JavaScript, PHP.

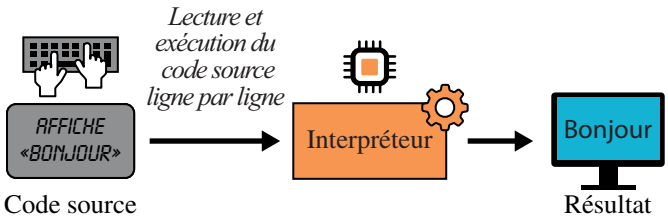


Figure 2.2 – Langage interprété

- les langages *semi-interprétés*, qui requièrent une compilation différente de celle vue précédemment, car elle ne transforme pas le code source en instructions binaires.
  Elle se contente de préparer le travail de l'interpréteur en réécrivant les instructions du code source dans un langage intermédiaire (bytecode), qui sera plus facile à traiter par l'interpréteur lors de son exécution.
  Exemple : Java, C#.

COURS

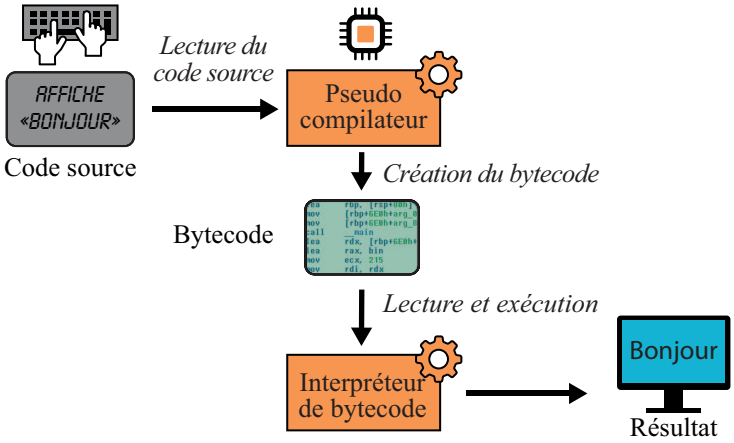


Figure 2.3 – Langage semi-interprété

4 Calculabilité et décidabilité

4.1 Calculabilité

La notion de calculabilité est apparue avec Turing en 1936. Pour l'appréhender, il est nécessaire d'en savoir un peu plus sur la machine de Turing.

4.1.1 - La machine de Turing

Retrouvez en vidéo une explication de la machine de Turing illustrée d'un exemple.

C'est un *modèle* abstrait de machine à calculer composé :

- d'un *ruban* infini composé de plusieurs cases dans lesquelles se trouve un élément de l'ensemble {0; 1} ; par exemple :

···	0	0	1	0	1	···

- d'une *tête de lecture et d'écriture* qui se trouve à chaque instant devant une des cases du ruban et qui peut être dans deux états possibles : 0 ou 1 ; par exemple :

0	0	1	0	1	···
		↑			
		0			

- d'une *table de transitions*, tableau contenant, comme son nom l'indique, des transitions (des transformations) ainsi que la valeur que l'on écrit, de la forme :

$$(t, v) \to (t', v')$$

où t et t' sont deux états de la tête de lecture, et où v et v' sont deux valeurs contenues dans une case ;
par exemple, la table :

(0,0)	→	(0,1)
(0,1)	→	(1,1)

signifie que :

→ si la tête de lecture est dans l'état « 0 » et pointe sur une case contenant la valeur « 0 » alors la tête de lecture reste sur l'état « 0 » mais la valeur passe à « 1 » et la tête de lecture avance d'une case ;

→ si la tête de lecture est dans l'état « 0 » et pointe sur une case contenant la valeur « 1 » alors la tête de lecture passe à l'état « 1 » et la valeur reste à « 1 » et la tête de lecture avance d'une case ;

→ pour les autres cas, rien ne change : l'état de la tête de lecture reste à sa valeur et il en est de même pour la valeur de la case vers laquelle pointe la tête de lecture. La tête de lecture reste où elle est.

Exemple (incrémentation) : Voyons comment modéliser l'opération $n \to n+1$ pour un entier n à l'aide d'une machine de Turing.

L'entier n est représenté par un ruban contenant n « 1 » suivis de plusieurs « 0 » ; prenons l'exemple de $n = 2$. Initialement, la tête de lecture pointe sur la première case et est dans l'état « 0 ».

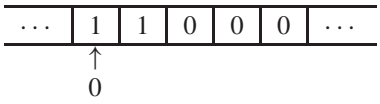


Figure 2.4 – Représentation du nombre « 2 » dans la machine de Turing

La table de transitions est la suivante :

(0,1)	→	(0,1)
(0,0)	→	(1,1)

Alors, comme la tête de lecture est initialement sur l'état « 0 » et que la valeur de la première case est « 1 », d'après la première transition, la tête de lecture restera à « 0 » et la valeur restera à « 1 ».

Une fois la transition prise en compte, la tête de lecture avance d'une case.

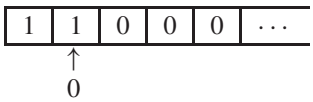


Figure 2.5 – Après la 1re transition

COURS

Là encore, la tête de lecture est à « 0 » et le nombre de la case vers laquelle elle pointe vaut « 1 », donc aucun changement n'est fait et la tête de lecture passe à la case suivante :

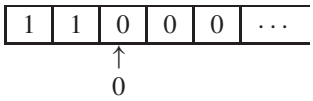


Figure 2.6 – Après la 2e transition

D'après la 2e ligne de la table de transitions, la tête de lecture passe à l'état « 1 » et la valeur de la case aussi, puis la tête de lecture avance d'une case :

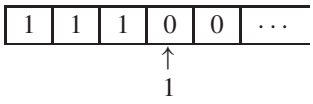


Figure 2.7 – Après la 3e transition

Comme aucune transition ne porte sur l'état (1,0), il n'y a plus de changement et la tête de lecture n'avance plus.

On aboutit ainsi à la valeur « 3 » (car il y a 3 « 1 »), ce qui correspond bien à l'incrémentation de 2.

4.1.2 - Thèse de Church-Turing et calculabilité

Cette thèse, qui n'est pas un théorème puisqu'elle n'a pas été démontrée, stipule que :

Si un problème est tel qu'il existe un algorithme le résolvant, alors il existe une machine de Turing qui résout le problème.

Cette thèse mène alors à la notion de calculabilité.

Définition 1

On dira qu'un problème est *calculable* s'il existe une machine de Turing le résolvant.

On peut ainsi définir un algorithme :

Définition 2

Un *algorithme* est une table de transitions d'une machine de Turing.

4.1.3 - La machine universelle

Généralisons la notion de machine de Turing et, pour bien comprendre, reprenons l'exemple de l'incrémentation.

Nous avons vu précédemment que nous avons passé en argument à la machine de Turing le nombre n afin d'obtenir son suivant. Appelons `Inc(n)` cette machine de Turing. `Inc` est définie par sa table de transitions, que l'on peut représenter simplement par la succession de binaires : « 01010011 ».

(0,1)	→	(0,1)
(0,0)	→	(1,1)

devient « 01010011 »

L'idée permettant de construire la machine U (la machine universelle) est de passer d'abord en argument la table de transitions sous forme binaire, puis le nombre n (pour notre exemple).

La machine universelle constitue les principes fondamentaux des ordinateurs d'aujourd'hui : on peut considérer un programme comme étant une table de transitions que l'on passe en paramètre.

Remarque : tous les problèmes ne sont pas calculables.

4.2 Décidabilité

Définition 3

Un problème est *décidable* si et seulement si il existe un algorithme qui le résout.

Cela sous-entend que l'algorithme se termine en un nombre fini d'opérations.

Exemple : un entier donné est-il un carré parfait ?

On peut aisément répondre à ce problème à l'aide de l'algorithme suivant :

```
n ← 0
x : nombre entier
Tant que (n² != x) et (n² < x):
   n ← n + 1
Retourner (n² == x)
```

COURS

Le code Python correspondant s'écrit :

```
def is_square(x):
    n = 0
    while (n**2 != x) and (n**2 < x):
        n += 1
    return (n**2 == x)

print(is_square(64))
```

Le problème est donc décidable.

Alan Turing a démontré en 1936 que tous les problèmes n'étaient pas décidables à l'aide d'un contre-exemple : le *problème de l'arrêt*. C'est l'objet de l'exercice type proposé au début de ce chapitre.

Solution de l'exercice type 1 — Lycée polyvalent Emile Zola, Aix-en-Provence

Nous avons posé $\overline{H}$ l'algorithme admettant l'algorithme A en paramètre qui exécute $H(A)$ tel que :

$$\begin{cases} \text{si } H(A) = 1 \text{ alors une boucle infinie est créée ;} \\ \text{si } H(A) = 0 \text{ alors il s'arrête.} \end{cases}$$

$\overline{H}$ existe si H existe.

Considérons alors $\overline{H}\left(\overline{H}\right)$.

- Si $\overline{H}$ s'arrête alors $H\left(\overline{H}\right) = 1$ et donc $\overline{H}\left(\overline{H}\right)$ crée une boucle infinie, donc ne s'arrête pas.
- Si $\overline{H}$ ne s'arrête pas alors $H\left(\overline{H}\right) = 0$ et donc $\overline{H}\left(\overline{H}\right)$ s'arrête.

Dans les deux cas, il y a contradiction. Ainsi, $\overline{H}$ n'existe pas, et donc H non plus.

Le problème de l'arrêt est donc indécidable.

Voir énoncé page 25

5 Récursivité

Exercice type 2 — Lycée Sainte-Anne, Brest

Implémenter en Python une fonction récursive `somme(n)` qui renvoie la valeur de $S_n = 1+2+3+\cdots+(n-1)+n$, en remarquant que $S_n = S_{n-1}+n$.

Voir corrigé page 37

Retrouvez une vidéo sur la récursivité.

5.1 Définition

Définition 4

Une fonction *récursive* est une fonction qui fait appel à elle-même.

Remarque : on parle aussi d'*algorithmes récursifs*.

Exemple (calcul d'une factorielle) : en mathématiques, on définit la factorielle d'un nombre entier n par :

$$n! = 1 \times 2 \times 3 \times 4 \times \cdots \times n.$$

Ainsi, si $f(n) = n!$ alors $f(n+1) = (n+1) \times f(n)$. On peut alors définir la fonction `factorielle` de façon récursive.

En Python, cela donne par exemple le code suivant.

```
def factorielle(n):
    if n == 0:
        return 1
    else:
        return n * factorielle(n-1)
```

En tapant :

```
print(factorielle(5))
```

le résultat « 120 » s'affiche, qui est bien égal à $5! = 1 \times 2 \times 3 \times 4 \times 5$.

Un autre exemple est celui du calcul du pgcd de deux nombres.

Remarque : le langage Python limite le nombre d'appels récursifs. Pour savoir le nombre maximal autorisé, on pourra taper :

```
import sys
print(sys.getrecursionlimit())
```

Par exemple, `factorielle(1000)` va afficher l'erreur :

```
RecursionError: maximum recursion depth exceeded in
comparison
```

Pour augmenter ce nombre limite, on pourra utiliser la syntaxe suivante :

```
sys.setrecursionlimit(1500)
```

On peut alors calculer `factorielle(1000)`, mais dans la mesure où le résultat comporte 2 568 (= `len(str(factorielle(1000)))`) chiffres, il ne figurera pas dans ce livre...

COURS

5.2 Complexité d'un programme récursif

Propriété 1

Un programme récursif a une complexité linéaire ou exponentielle.

Exemple : reprenons l'exemple précédent (algorithme récursif de la factorielle). Notons $C(n)$ la complexité de la fonction pour la variable entière n.

- Si $n = 0$, seul le test `n == 0` est effectué donc $C(0) = 1$;
- si $n > 0$, le test et le produit sont pris en compte en plus des opérations élémentaires de la fonction pour $n - 1$, donc $C(n) = 2 + C(n - 1)$.

On peut donc dire que la complexité de cette fonction est la suite (c_n) définie par $c_0 = 1$ et $c_{n+1} = c_n + 2 = 2n + 1$.

On dira alors que la complexité est :

- *linéaire* si $C(n) = O(n)$
- *quadratique* si $C(n) = O(n^2)$
- *exponentielle* si $C(n) = O(\mathrm{e}^x)$
- *logarithmique* si $C(n) = O(\ln x)$

où la notation $O(\ldots)$ est appelée *notation de Landau* : on prend le terme « dominant » à l'infini sans le coefficient dans l'expression de $C(n)$.

5.3 Correction d'un algorithme récursif

Pour garantir qu'un algorithme récursif fonctionne correctement, on doit vérifier :

- que lorsqu'il atteint un état final, il doit fournir la solution correcte au problème osé : c'est la *correction partielle* ;
- qu'il doit toujours parvenir à un état final, c'est-à-dire qu'il doit cesser de faire des appels récursifs après un nombre fini d'étapes. Cela implique de s'assurer que chaque appel récursif progresse vers une condition de base qui met fin à la récursion : c'est le *problème de la terminaison*

5.3.1 - Terminaison

Regardons sur l'exemple de la fonction récursive `factorielle` vue précédemment comment vérifier sa terminaison.

La fonction admet un argument entier positif. De plus,

- `factorielle(n)` fait appel à `factorielle(n-1)` ;
- `factorielle(n-1)` fait appel à `factorielle(n-2)` ;
- etc.
- `factorielle(1)` fait appel à `factorielle(0)` ;
- `factorielle(0)` renvoie la valeur « 1 ».

On est donc assurés que cette fonction se termine.

5.3.2 - Correction partielle

Définition 5

Dans une fonction récursive, un *appel interne* est un appel de la fonction déclenchée par elle-même (un appel *récursif*).

Pour démontrer la *correction partielle* d'un algorithme récursif, il est nécessaire de prouver que, si chaque appel récursif produit le résultat attendu, alors l'algorithme global produira également le résultat correct. En d'autres termes, si les sous-problèmes résolus par les appels récursifs sont corrects, alors la solution globale le sera aussi. Il en va de même pour une fonction.

Par exemple, pour la fonction `factorielle`, si on part du principe que `factorielle(n-1)` représente $(n-1)!$ alors `n * factorielle(n-1)` représente bien $n \times (n-1)! = n!$.

Cela ressemble au *raisonnement par récurrence* vu en mathématiques.

6 Modularité

6.1 Principe

Quand on doit élaborer un programme complexe, il est primordial de le penser en terme de modularité. Cela signifie qu'il est nécessaire de le découper en un maximum de blocs afin de le rendre plus lisible et fonctionnel.

Concevoir un programme avec plusieurs fonctions permet de faire appel à ces fonctions à plusieurs endroits dans le programme.

Les fonctions les plus utilisées pourront être placées dans des fichiers auxiliaires, autrement appelés *modules* ou *bibliothèques* (en Python par exemple).

Exemple : imaginons un module nommé `bienpratique` renfermant plusieurs fonctions. On peut alors écrire un programme principal faisant appel à ce module, c'est-à-dire le programme qui contient l'ensemble des instructions qui seront directement exécutées à son lancement.

L'exemple de la page suivante montre le contenu du fichier `bienpratique.py`, contenant deux fonctions `is_square` et `is_odd`, qui pourront être utilisées dans le fichier `main.py` dans lequel on importe `bienpratique.py`.

Remarque : quand on écrit un module, il est nécessaire de le documenter au mieux afin que toute personne puisse s'y retrouver et comprendre les blocs de code.

COURS

Module : bienpratique.py

```
"""
Fonction : is_square(entier n)
But : teste si la variable est un carré parfait
"""

def is_square(x):
    n = 1
    while (n**2 != x) and (n**2 < x):
        n += 1
    return (n ** 2 == x)

"""
Fonction : is_odd(entier n)
But : teste si la variable est impaire
"""

def is_odd(n):
    if (n % 2 == 0):
        return False
    else:
        return True
```

Programme principal : main.py

```
from bienpratique import *

n = int(input("Entrez un nombre entier : "))

if is_square(n):
    print("C'est un carré parfait !")

if is_odd(n):
    print("C'est un nombre impair.")
else:
    print("C'est un nombre pair.")
```

6.2 Ne pas réinventer la roue

Il existe déjà de nombreux modules libres et gratuits, fournis par Python, des entreprises et des particuliers. Il n'est donc pas nécessaire de créer des fonctions qui existent déjà.

Par exemple,

- le module `random` contient des fonctions qui ont un rapport avec la génération de nombre pseudo-aléatoires.
  Documentation : `https://docs.python.org/fr/3/library/random.html`.
- le module `numpy` concerne les calculs scientifiques.
  Documentation : `https://www.numpy.org`.
- le module `sympy` concerne les mathématiques symboliques (le calcul algébrique par exemple).
  Documentation : `https://www.sympy.org/en/index.html`.
- le module `pandas` concerne les statistiques et les manipulations de données.
  Documentation : `https://pandas.pydata.org`.
- le module `os` fournit des fonctionnalités dépendantes du système d'exploitation.
  Documentation : `https://docs.python.org/fr/3/library/os.html`.

Solution de l'exercice type 2 — Lycée Sainte-Anne, Brest

Comme dans toute fonction récursive, il faut penser à la terminaison, sans quoi le programme risque de ne jamais s'arrêter.
Ici, le premier terme de la somme est 1 donc il nous faut renvoyer « 1 » quand $n = 1$.
De plus, si on note $S_n = 1 + 2 + \cdots + n$, on voit que $S_n = S_{n-1} + n$, ce qui nous permet de dire que si $n \neq 1$ alors on doit retourner la somme de `somme(n-1)` et `n`.
On en déduit la fonction suivante :

```
def somme(n):
    # on suppose que n est un entier positif ou nul
    if n == 0:
        return 0
    return n + somme(n-1)

print(somme(100))
```

Voir énoncé page 32

INTERROS

1 QCM Vérification de connaissances

5 min

Corrigé p. 47

1 Un ordinateur comprend mieux :

- **a** des instructions en anglais ;
- **b** des instructions en langage binaire (0 et 1) ;
- **c** des instructions exprimées en Python ou en C (par exemple).

2 Lorsque l'on rédige un programme pour un ordinateur, on utilise :

- **a** un langage informatique comme C, Basic ou PHP ;
- **b** le langage binaire (0 et 1) obligatoirement ;
- **c** de préférence l'anglais.

3 Si on veut pouvoir modifier aisément un programme, il vaut mieux disposer :

- **a** du programme binaire ;
- **b** du programme source ;
- **c** du programme rédigé en anglais ou en français ;
- **d** la question n'a pas de sens : c'est impossible de modifier un programme.

4 La compilation, c'est :

- **a** la transformation du texte du programme binaire vers un langage informatique ;
- **b** la transformation du programme rédigé en anglais vers le français ;
- **c** l'ajout bout à bout de différents programmes ;
- **d** la transformation d'un programme source, écrit dans un langage informatique, en langage binaire ou en langage intermédiaire.

5 La compilation d'un programme doit se faire :

- **a** avant chaque utilisation du programme ;
- **b** régulièrement pendant le fonctionnement du programme ;
- **c** à chaque modification du code source ;
- **d** jamais.

6 Si je veux diffuser un programme, mais que je ne souhaite pas que les utilisateurs sachent comment il fonctionne,

- **a** je dois distribuer le code binaire uniquement ;
- **b** je dois distribuer le code source uniquement ;
- **c** je dois le compresser au format ZIP, sans nécessairement le protéger par un mot de passe.

7 Si je veux diffuser un programme et que je souhaite que les utilisateurs puissent l'améliorer...

- **a** je dois distribuer le code binaire uniquement ;
- **b** je dois diffuser le code source du programme ;
- **c** il n'y a aucun moyen de permettre aux utilisateurs d'améliorer un programme.

8 Un code source en langage interprété :

a peut être directement exécuté par le noyau et le processeur ;

b ne peut pas être directement exécuté par le noyau et le processeur.

2 Fonction mystère

★ 5 min Corrigé p. 47

Lycée François Mauriac, Bordeaux

Que fait la fonction suivante quand elle est appelée avec une seule liste en paramètre ?

```
def mystere(L, M = [ ]):
    # L est une liste
    if L is None or len(L) == 0:
        return M
    a = L.pop(0)
    if a not in M:
        M.append(a)
    return mystere(L, M)
```

3 Le retour de la fonction mystère

★ 5 min Corrigé p. 47

Lycée Henri Bergson, Angers

Que fait la fonction suivante :

```
def mystere(L):
    if len(L) == 1 :
        return L[0]
    if L[0] < L[1]:
        L.pop(1)
    else:
        L.pop(0)

    return mystere(L)
```

4 Calcul du pgcd de deux nombres

★ 10 min Corrigé p. 48

Lycée Camille Jullian, Bordeaux

Écrire en Python une fonction récursive `pgcd(a,b)` renvoyant le plus grand diviseur commun de deux nombres a et b.

Pour cela, on utilisera le résultat mathématique suivant :

« $\text{pgcd}(a,b) = \text{pgcd}(b,r)$, où $a = bq + r$. »

5 Nombre d'adhérents

★ 10 min Corrigé p. 48

Lycée Jean-Joseph Fourier, Auxerre

Une association a remarqué que d'une année à l'autre,

- elle perd 5 % de ses adhérents ;
- elle gagne 200 adhérents.

1 En admettant que le nombre d'adhérents de cette association était égal à 2 000 au 1er janvier 2019, écrire en Python une fonction récursive nommée `nombre(n)` affichant le nombre théorique d'adhérents après n années, $n \geqslant 1$.

2 Dans ce même programme, afficher le nombre théorique d'adhérents au cours des 20 prochaines années.

6 Suite de Fibonacci

★ 10 min Corrigé p. 49

Lycée Sophie Germain, Paris

La suite de Fibonacci est la suite numérique définie par :

$$F_0 = F_1 = 1 \quad , \quad \forall n \geqslant 0,\ F_{n+2} = F_{n+1} + F_n.$$

Écrire en Python une fonction récursive `Fibo(n)` permettant d'afficher le terme F_n, où $n \geqslant 0$, puis afficher les termes de F_0 à F_{20}.

7 Fonction à compléter

★ 15 min Corrigé p. 49

Lycée Richelieu, Versailles

La fonction `produit_des_chiffres(n)` prend en argument un entier positif `n` et calcule le produit des chiffres composant ce nombre entier.

Par exemple, `produit_des_chiffres(243)` doit renvoyer $2 \times 4 \times 3 = 24$. L'implémentation en Python peut utiliser les opérateurs division entière (`//`) et modulo (`%`).

1 Que doit retourner les instructions suivantes ?

- `produit_des_chiffres(8)`
- `produit_des_chiffres(222)`
- `produit_des_chiffres(90)`
- `produit_des_chiffres(1117)`

2 Compléter les lignes 2 et 5 de la définition de fonction ci-dessous afin de répondre à la spécification de la fonction.

```
def produit_des_chiffres(n):
    if ..... :
        return n
    else:
        return n % 10 * ..................
```

3 Réécrire cette fonction en ajoutant un test qui vérifie la précondition et qui fait en sorte que la fonction retourne `None` si la précondition n'est pas vérifiée.

8 La fonction compteur

★★ 10 min Corrigé p. 50

Lycée Jean-Joseph Fourier, Auxerre

Écrire une fonction récursive `compteur(chaine,caractere)` retournant le nombre de fois que l'on peut compter le caractère dans la chaîne.

Par exemple, `compteur('Bonjour Lucie','u')` renvoie le nombre 2 car il y a deux « u » dans la chaîne « Bonjour Lucie ».

Remarque : la méthode `count` permet d'obtenir ce résultat, mais le but de cet exercice n'est pas de se servir de cette méthode. On peut en effet avoir le nombre d'occurrences du caractère « p » dans une chaîne en tapant `chaine.count('p')`.

9 La vengeance de la fonction mystère

★★ 10 min Corrigé p. 51

Lycée François Villon, Les Mureaux

On considère le programme suivant :

```
def f(a,b):
    # a et b sont deux entiers positifs ou nuls
    if b == 0:
        return 0

    if b == 1:
        return a

    return a + f(a, b-1)
```

1. Quel résultat retourne `f(7,9)` ? (exécuter le programme à la main)
2. En déduire la signification de la valeur retournée par cette fonction pour deux entiers naturels non nuls a et b quelconques.
3. Pour une fonction récursive, un *cas de base* est un cas qui ne nécessite pas d'appel récursif à la fonction.

   Quel est le cas de base de cette fonction ?
4. Qu'est-ce qui garantit que le programme s'arrête ?
5. La complexité de cette fonction est-elle linéaire ? Quadratique ? Exponentielle ?

INTERROS

10 Diagonale de Cantor

★★ 15 min Corrigé p. 51

Lycée Jean Macé, Niort

Dans un repère, les points à coordonnées entières sont numérotés selon le principe suivant :

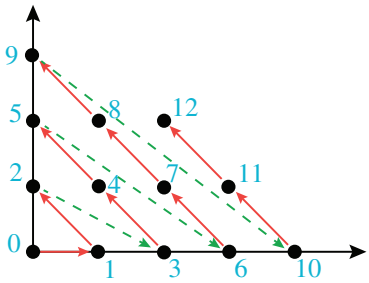


Figure 2.8 – Diagonale de Cantor

Écrire une fonction récursive `numero(x,y)` qui retourne le numéro du point de coordonnés $(x; y)$.

Par exemple, `numero(3,1)` doit retourner le nombre « 11 ».

11 Parcours de fichiers

★★ 15 min Corrigé p. 52

Lycée Richelieu, Versailles

Vous devez écrire une fonction récursive pour parcourir un système de fichiers et lister tous les fichiers présents dans un répertoire et ses sous-répertoires.

Votre objectif est d'écrire une fonction récursive `lister_fichiers(repertoire)` qui prend le chemin d'un répertoire en entrée et retourne une liste de tous les fichiers présents dans ce répertoire et ses sous-répertoires.

Instructions :

1. Écrivez une fonction récursive `lister_fichiers(repertoire)` qui prend le chemin d'un répertoire en entrée et retourne une liste de tous les fichiers présents dans ce répertoire et ses sous-répertoires.
2. La fonction doit utiliser les modules `os` et `os.path` pour naviguer dans le système de fichiers.
3. Pour chaque répertoire, la fonction doit :
   - Lister tous les fichiers et sous-répertoires présents.
   - Ajouter les fichiers à la liste des résultats.
   - Appeler récursivement la fonction pour chaque sous-répertoire.
4. Testez votre fonction avec un répertoire contenant plusieurs fichiers et sous-répertoires pour vérifier qu'elle fonctionne correctement.

12 Les tours de Hanoï

★★★ 20 min Corrigé p. 52

Lycée Sainte-Anne, Brest

Selon Wikipedia :

> Les tours de Hanoï (originellement, la tour d'Hanoï) sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles page suivante.

> - on ne peut déplacer plus d'un disque à la fois ;
> - on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.
>
> On suppose que cette dernière règle est également respectée dans la configuration de départ.

Ainsi, en notant n le nombre de disques, la configuration de départ pour $n = 3$ est la suivante :

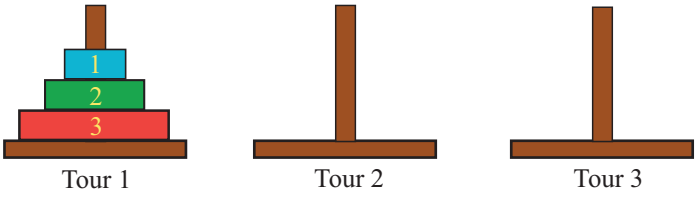


Figure 2.9 – Configuration initiale des tours de Hanoï

Pour résoudre le jeu, il faut déplacer tous les disques sur la Tour 3 en se servant de la tour 2 comme intermédiaire, ceci en commençant par déplacer le disque bleu sur la tour 3, puis le disque vert sur la tour 2, puis on peut mettre le disque bleu sur le disque vert, etc.

1. Pour $n = 3$, résoudre « à la main » le jeu. On notera (comme sur la figure 2.9) les disques avec les chiffres 1, 2 et 3 et on décrira les déplacements (par exemple, 1 va sur T3, 2 va sur T2, etc.)
2. Pour n quelconque, ramener le problème initial à un même problème en faisant intervenir les $n - 1$ premiers disques d'une part et le n-ième disque d'autre part.
3. En déduire une fonction récursive `hanoi(n,start=1,end=3,aux=2)` permettant d'afficher tous les mouvements jusqu'à résolution complète du jeu, où `start` désigne le numéro de la tour d'où provient un disque, `end` celui de la tour vers laquelle on déplace le disque et `aux` celui de la tour auxiliaire.

INTERROS

13 Preuve : incalculabilité

★★★ 30 min Corrigé p. 54

Lycée Turgot, Paris

Cet exercice a pour but de démontrer que tous les problèmes ne sont pas calculables.

Pour cela, nous aurons besoin des définitions suivantes :

- une *bijection* de $\mathbb{N}$ vers l'ensemble E est une fonction f telle que :
  - → quel que soit l'entier naturel n, il existe une unique image de n par f ;
  - → quel que soit l'élément e de E, il existe un unique antécédent dans $\mathbb{N}$ à e.

  La fonction $n \mapsto 2n$ est par exemple une bijection de $\mathbb{N}$ vers l'ensemble des entiers positifs pairs ;
- un ensemble E est *dénombrable* s'il existe une bijection de $\mathbb{N}$ vers E.

1 Un algorithme est une succession de caractères pris dans un alphabet fini, noté A. Montrer que l'ensemble $\mathcal{A}$ des algorithmes est dénombrable.

Aide : on pourra se servir de l'ensemble C des combinaisons des caractères de $\mathcal{A}$.

2 Un problème de décision peut être représenté par une fonction booléenne $f : n \to \{0; 1\}$ (« 0 » pour le cas où la propriété est fausse pour la variable et « 1 » dans le cas contraire).

Par exemple, le problème « Est-ce que l'entier est pair ? » se représente par la fonction $f : 2p \mapsto 1$ et $f : 2p+1 \mapsto 0$, pour tout entier naturel p.

On note $\mathcal{B}$ l'ensemble des fonctions booléennes.

On suppose que $\mathcal{B}$ est dénombrable, c'est-à-dire qu'il existe des fonctions f_1, f_2, ... qui peuvent être représentées par exemple comme ci-dessous :

Valeurs de n	0	1	2	3	⋯
f_0	1	0	0	1	⋯
f_1	1	1	0	0	⋯
f_2	0	1	0	0	⋯
⋮					

On pose alors g telle que : $g : \mathbb{N} \to \{0; 1\}$

$$k \mapsto 1 - f_k(k)$$

(a) Calculer $g(0)$, $g(1)$ et $g(2)$ pour les fonctions f_0, f_1 et f_2 données en exemple dans l'énoncé.

(b) Dans un cas général, montrer que, quel que soit l'entier naturel k, $g \neq f_k$.

(c) Justifier que $g \in \mathcal{B}$ et conclure à une contradiction.

Que cela permet-il de conclure sur $\mathcal{B}$?

3 Justifier alors que tous les problèmes ne sont pas dénombrables.

14 Grille de Sudoku

★★★ 50 min Corrigé p. 56

Lycée Sainte-Clotilde, Strasbourg

Un *Sudoku* classique est une grille de 9 lignes et 9 colonnes, donc composée de 9 blocs distincts de 3 lignes et 3 colonnes.

Remplir une grille de Sudoku consiste à utiliser tous les chiffres de 1 à 9 pour chacun des 9 blocs de sorte que chaque ligne et chaque colonne de la grille totale ne comporte aucun chiffre en double.

Dans cet exercice, on décidera de représenter une telle grille par une liste `S` de neuf listes à neuf éléments, chacune des listes représentant une ligne.

L'objectif de cet exercice est d'obtenir le remplissage de la grille ci-dessous (figure 2.10).

1 Compléter la variable :`S = [ [0,1,0,0,7,8,0,0,0] , ... ]`

2 Écrire une fonction `ligne(S,i)` qui retourne la liste des chiffres de 1 à 9 qui apparaissent sur la ligne i. On devra avoir pour notre grille :

```
>>> ligne(S,0)
[1,7,8]
```

	1			7	8			
	8			4		9		
		5	6				1	
1				6				5
	4		9	1	5		7	2
	6	7		8		4		
			3			1		
	7		8	9			2	3
					4			

Figure 2.10 – Grille de Sudoku à implémenter

3 Écrire une fonction `colonne(S,j)` qui retourne la liste des chiffres de 1 à 9 qui apparaissent dans la colonne j. On devra avoir pour notre grille :

```
>>> colonnes(S,0)
[1]
```

INTERROS

4 Écrire une fonction `bloc(S,i,j)` qui retourne la liste des chiffres de 1 à 9 qui apparaissent sur le bloc 3×3 auquel appartient la case de la ligne i et de la colonne j. On devra avoir pour notre grille :

```
>>> bloc(S,3,4)
[6,9,1,5,8]
```

5 Écrire alors une fonction `possibles(S,i,j)` qui retourne la liste des chiffres de 1 à 9 que l'on peut écrire dans la case de la ligne `i` et de la colonne `j` (en tenant compte des règles du jeu). On devra avoir pour notre grille :

```
>>> possibles(S,0,0)
[2,3,4,6,9]
```

6 On complète la grille de la ligne 0 à la ligne 8, de la colonne 0 à la colonne 8.

Écrire une fonction `suivante(i,j)` qui reçoit en paramètre la ligne `i` et la colonne `j` d'une case, et qui renvoie le tuple (ligne,colonne) de la case suivante. On devra avoir :

```
>>> suivante(0,0) , suivante(0,8) , suivante(8,8)
(0,1) , (1,0) , (9,0)
```

7 Compléter la fonction principale de résolution suivante :

```
def solve(S,i,j):
    if i == 9:
        return ...
    elif S[i][j] > 0:
        ...
        return ...
    for k in possibles(S,i,j):
        S[i][j] = k
        ...
        if solve(S,a,b):
            return ...
    S[i][j] = 0
    return False
```

1 QCM Vérification de connaissances

Énoncé p. 38

1. Réponse **b**. Un ordinateur ne sait interpréter que le langage binaire.
2. Réponse **a**. Les langages informatiques permettent d'obtenir (compilation) des programmes binaires (ou intermédiaires) qui sont compris par les ordinateurs (ou par un programme interpréteur).
3. Réponse **b**. Il est beaucoup plus facile pour un humain de comprendre et de modifier un code source qu'un programme binaire.
4. Réponse **d**. La compilation est précisément la traduction du programme source en programme binaire (ou intermédiaire).
5. Réponse **c**. Quand on modifie le code source d'un programme, il faut le compiler à nouveau pour remplacer le programme binaire déjà existant afin de prendre en compte les dernières modifications.
6. Réponse **a**. En ne disposant que du code binaire d'un programme, il est humainement difficile de le comprendre et de le modifier.
7. Réponse **b**. Un programme peut être modifié en ayant son code source.
8. Réponse **b**. Un code source en langage interprété ne peut pas être directement exécuté par le noyau et le processeur. C'est le cas de Python (contrairement au C).

2 Fonction mystère

Énoncé p. 39

Lycée François Mauriac, Bordeaux

La première instruction de la fonction consiste à tester si la liste `L` est absente, auquel cas la fonction retourne `M`, c'est-à-dire une liste vide.

Ensuite, on définit `a` comme étant le premier élément de `L` tout en enlevant cet élément de `L` (la méthode `pop(i)` supprime l'élément d'indice i).

Si `a` n'est pas dans la liste `M` alors on l'y ajoute.

En écrivant `return mystere(L,M)`, on exécute à nouveau la fonction `mystere` avec pour arguments la nouvelle liste `L` (donc sans l'élément `a`) et `M` (qui contient alors `a`).

Au final, on obtient une liste `M` des éléments distincts de `L`.

La fonction retourne donc la liste des éléments distincts d'une autre liste.

3 Le retour de la fonction mystère

Énoncé p. 39

Lycée Henri Bergson, Angers

La fonction commence par tester si la liste a un seul élément, auquel cas elle retourne cet élément. Cela nous donne donc un indice sur ce qu'elle fait : le retour est un nombre.

Ensuite, on compare les deux premiers éléments de la liste : si le premier est plus petit que le deuxième alors on supprime le deuxième sinon on supprime le premier ; on ne garde donc que le plus petit des deux premiers éléments et on écourte la liste d'un élément (le plus grand des deux premiers).

On termine par faire appel à la fonction elle-même.

Il s'agit donc ici de diminuer la liste initiale en ne conservant que le plus petit des deux premiers éléments à chaque fois. L'élément restant au final est donc le plus petit élément de la liste initiale.

4 Calcul du pgcd de deux nombres

Énoncé p. 39

Lycée Camille Jullian, Bordeaux

Une fonction récursive permettant de calculer et d'afficher le pgcd de deux nombres a et b est la suivante :

```
def pgcd(a,b):
    if b == 0:
        return a
    else:
        r = a % b
        return pgcd(b,r)
```

Rappelons que l'opérateur « % » permet d'obtenir le reste d'une division euclidienne. Ainsi, « `a % b` » représente le reste de la division euclidienne de `a` par `b`.

5 Nombre d'adhérents

Énoncé p. 40

Lycée Jean-Joseph Fourier, Auxerre

1 La fonction récursive calculant et donnant le nombre théorique d'adhérents est :

```
def nombre(n):
    if n == 0:
        u = 2000
    else:
        u = nombre(n-1)*0.95+200
    return u
```

En effet, on peut noter u_n le nombre d'adhérents lors de l'année $2019+n$. Ainsi, $u_0 = 2\,000$ et $u_{n+1} = 0{,}95u_n + 200$ (une perte de 5 % revient à conserver 95 % de la valeur précédente). Cette dernière égalité nous donne la formule récursive à mettre dans la fonction.

2 Pour afficher les 20 valeurs qui suivent la première, on utilise une boucle itérative :

```
>>> for i in range(1,21):
>>>     print(nombre(i))
```

6 Suite de Fibonacci

Énoncé p. 40

Lycée Sophie Germain, Paris

Voici une suggestion de fonction :

```
def fibo(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)

for i in range(21):
    print(fibo(i))
```

En effet, la relation $\forall n \geqslant 0,\ F_{n+2} = F_{n+1} + F_n$ peut aussi s'écrire :

$$\forall n \geqslant 2,\ F_n = F_{n-1} + F_{n-2}\ ,$$

d'où la formule : « `f = fibo(n-1)+fibo(n-2)` ».

Cependant, ce programme (naïf) n'est pas optimal. En effet, des mêmes termes sont calculés plusieurs fois, ce qui rend le programme quelque peu inefficace.

Nous verrons dans le chapitre suivant une autre solution utilisant la *programmation dynamique* (voir exercice **??** page **??**).

7 Fonction à compléter

Énoncé p. 40

Lycée Richelieu, Versailles

1
- `produit_des_chiffres(8)` doit retourner « 8 » ;
- `produit_des_chiffres(90)` doit retourner 9×0, soit « 0 » ;
- `produit_des_chiffres(222)` doit retourner $2 \times 2 \times 2$ soit « 8 » ;
- `produit_des_chiffres(1117)` doit retourner « 7 ».

2 Le programme dûment complété est le suivant :

```
def produit_des_chiffres(n):
    if n<10:
        return n
    else:
        return n % 10 * produit_des_chiffres(n//10)
```

3 Le code modifié est le suivant :

```
def produit_des_chiffres(n):
    if type(n) != int or n < 0:
        return None

    if n < 10:
        return n
    else:
        return n % 10 * produit_des_chiffres(n//10)
```

8 La fonction compteur

Énoncé p. 41

Lycée Jean-Joseph Fourier, Auxerre

Une première possibilité est la suivante :

```
def compteur(chaine, caractere):
    if chaine == '':
        return 0
    if chaine[0] ==  caractere:
        n = 1
    else:
        n = 0
    return n + compteur(chaine[1:], caractere)
```

L'idée consiste ici à comparer le premier caractère de la chaîne au caractère recherché ; s'ils coïncident alors on ajoute 1 au nombre d'occurrences du caractère dans la chaîne obtenue à partir de la chaîne principale en ayant enlevé le premier caractère. Sinon, on n'ajoute rien.

Une autre possibilité, bien moins naturelle, est la proposition suivante :

```
def compteur(chaine, caractere):
    L = list(chaine)
    if caractere in L:
        L.pop(L.index(caractere))
        return 1 + compteur(str(L), caractere)
    return 0
```

MÉTHODE

On transforme d'abord la chaîne de caractères en une liste.

Si le caractère est trouvé dans les éléments de cette nouvelle liste alors on l'enlève (avec la méthode `pop`) et on ajoute 1 au compteur qui correspond à la chaîne initiale privée du caractère enlevé (la liste a été transformée en chaîne avec la méthode `str`).

Enfin, il ne faut pas oublier de retourner « 0 » si le caractère n'est pas présent... sinon, un magnifique message d'erreur apparaît !

Mais cette façon de voir est légèrement tirée par les cheveux...

9 La vengeance de la fonction mystère

Énoncé p. 41

Lycée François Villon, Les Mureaux

1 La fonction calcule :
$$\begin{aligned} 7 + f(7,8) &= 7 + 7 + f(7,7) \\ &= 2 \times 7 + 7 + f(7,6) \\ &= 3 \times 7 + 7 + f(7,5) \\ &= 4 \times 7 + 7 + f(7,4) \\ &= 5 \times 7 + 7 + f(7,3) \\ &= 6 \times 7 + 7 + f(7,2) \\ &= 7 \times 7 + 7 + f(7,1) \\ &= 8 \times 7 + 7 = 9 \times 7 = 63. \end{aligned}$$

2 De l'exemple précédent, on peut remarquer que la fonction retourne $a \times b$.

3 Le cas de base de cette fonction est « $b = 1$ » car dans ce cas, on retourne une valeur (ici, la valeur de a).

4 Le fait d'appeler `f(a,b-1)` et le fait que b soit un entier naturel non nul nous assure que la valeur de b diminue de 1 en 1 et qu'elle va atteindre la valeur « 1 », auquel cas le programme s'arrête.

5 Ici, la complexité est ici linéaire car, au pire, on fait appel a fois à la fonction `f`.

10 Diagonale de Cantor

Énoncé p. 42

Lycée Jean Macé, Niort

Une fonction possible est la suivante :

```
def numero(x,y):
    if (x,y) == (0,0):
        return 0
    elif y == 0:
        return 1 + numero(0,x-1)
    else:
        return 1 + numero(x+1,y-1)
```

L'idée ici est de partir du point de coordonnées (x,y), qui initialise notre compteur à 1, et d'ajouter `numero(x+1,y-1)` tant que l'on ne se trouve pas sur l'axe des abscisses. Si l'ordonnée est nulle, il faut en effet revenir à une abscisse nulle et à une ordonnée qui est égale à l'abscisse du point duquel nous sommes partis diminuée de 1 : `numero(0,x-1)`.

On s'arrête pour le point de cordonnées (0,0). À chaque fois que l'on fait appel à la fonction récursive, on ajoute 1 (car un point est compté en plus).

11 Parcours de fichiers

Énoncé p. 42

Lycée Richelieu, Versailles

Un programme possible, répondant aux instructions, est le suivant :

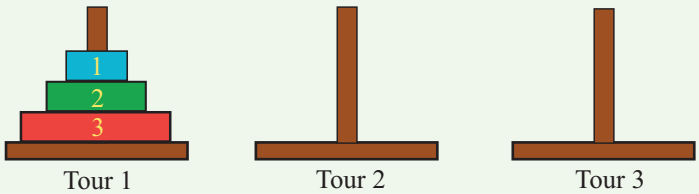
```
import os

def lister_fichiers(repertoire):
    fichiers = []
    for element in os.listdir(repertoire):
        chemin = os.path.join(repertoire, element)
        if os.path.isfile(chemin):
            fichiers.append(chemin)
        elif os.path.isdir(chemin):
            fichiers.extend(lister_fichiers(chemin))
    return fichiers

# Exemple d'utilisation
# Remplacez '/chemin/vers/repertoire' par le chemin ré
    el à tester
print(lister_fichiers('/chemin/vers/repertoire'))
```

12 Les tours de Hanoï

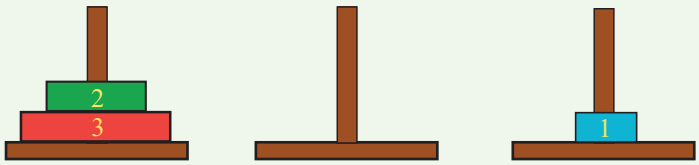
Énoncé p. 43

Lycée Sainte-Anne, Brest

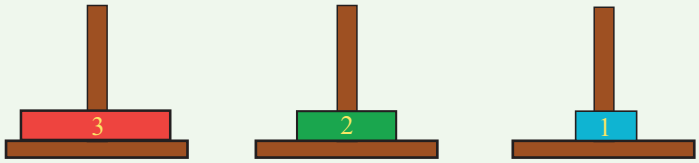
1 Déplacement 1 :



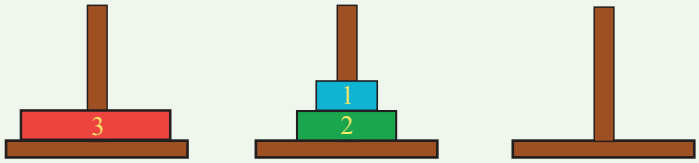
Déplacement 2 :



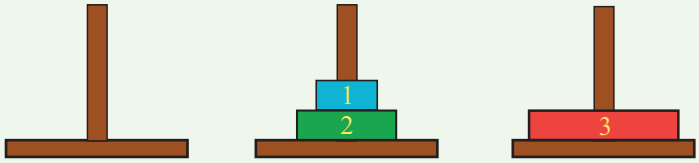
Déplacement 3 :



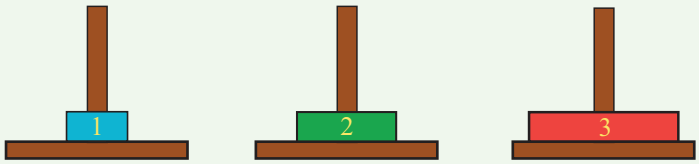
Déplacement 4 :



Déplacement 5 :



Déplacement 6 :



2 Afin de résoudre le jeu, il faut qu'à l'avant-dernier déplacement, tous les $n - 1$ plus petits disques soient empilés sur la tour 2 afin de pouvoir déplacer le disque n vers la tour 3.

Déplacer n disques de T1 vers T3 (en utilisant T2 comme intermédiaire) revient à :

- déplacer $n-1$ disques de T1 à T2 (en utilisant T3 comme intermédiaire) ;
- déplacer le n-ième disque de T1 vers T3 ;
- déplacer à nouveau les $n-1$ disques de T2 vers T3 (en utilisant T1 comme intermédiaire).

3 Dans la fonction récursive, il faut une condition d'arrêt (le cas de base) : ce sera le cas où $n = 1$, auquel cas on affichera : « Déplacement du disque de la tour X vers la tour 3 », qui marquera presque la fin de la récursivité (il faudra encore déplacer à nouveau les $n-1$ disques situés sur T2).

Si $n \neq 1$ alors le problème revient à déplacer $n-1$ disques vers la tour auxiliaire, donc on devra faire appel à `hanoi(n-1,start,aux,end)`, puis on devra déplacer les $n-1$ disques de l'auxiliaire vers la dernière tour, donc faire appel à `hanoi(n-1,aux,end,start)`.

```
def hanoi(n, start = 1, end = 3, aux = 2):
    if n == 1:
        hanoi(1, start, end, aux)
    else:
        hanoi(n-1, start, aux, end)
        print('T',start,' --> T',end)
        hanoi(n-1, aux, end, start)
```

```
>>> hanoi(3)
T 1 -> T 3
T 1 -> T 2
T 3 -> T 2
T 1 -> T 3
T 2 -> T 1
T 2 -> T 3
T 1 -> T 3
```

13 Preuve : incalculabilité

Énoncé p. 44

Lycée Turgot, Paris

1 Notons $A = \{a_0, a_1, \cdots, a_{n-1}\}$ l'ensemble des caractères nécessaires à la rédaction des algorithmes.

Considérons alors la fonction f définie comme suit :

$$\begin{array}{ccc} 0 & \mapsto & a_0 \\ \vdots & \mapsto & \vdots \\ n-1 & \mapsto & a_{n-1} \\ n & \mapsto & a_0a_0 \\ n+1 & \mapsto & a_0a_1 \\ \vdots & \mapsto & \vdots \\ 2n-1 & \mapsto & a_0a_{n-1} \\ 2n & \mapsto & a_0a_0a_0 \\ \vdots & \mapsto & \vdots \end{array}$$

f est manifestement une bijection de $\mathbb{N}$ vers l'ensemble C constitué des combinaisons des n caractères de A. Or, $\mathcal{A}$ (l'ensemble des algorithmes) est inclus dans C (car tout algorithme est une combinaison des caractères de A).

Il existe donc bien une bijection de $\mathbb{N}$ vers $\mathcal{A}$, donc $\mathcal{A}$ est dénombrable.

2 **(a)** $g(0) = 1 - f_0(0) = 1 - 1 = 0.$

$g(1) = 1 - f_1(1) = 1 - 1 = 0.$

$g(2) = 1 - f_2(2) = 1 - 0 = 1.$

(b) Dans un cas général,

- $g(0) = 1 - f_0(0)$ donc $g(0) \neq f_0(0)$, ce qui montre que $g \neq f_0$;
- $g(1) = 1 - f_1(1)$ donc $g(1) \neq f_1(1)$, ce qui montre que $g \neq f_1$;
- quel que soit l'entier k, $g(k) = 1 - f_k(k) \neq f_k(k)$, donc $g \neq f_k$.

(c) Quel que soit l'entier naturel k, $f_k \in \mathcal{B}$ donc $g : 1 - f_k \in \mathcal{B}$.

Or, nous venons de montrer à la question précédente que g n'était pas une fonction f_k, donc que $g \notin \mathcal{B}$. Il y a donc bien une contradiction.

Cela suppose donc que notre hypothèse de départ est fausse, à savoir que $\mathcal{B}$ est dénombrable.

3 D'une part, nous avons montré que l'ensemble des algorithmes était dénombrable, d'autre part que l'ensemble des fonctions représentant les problèmes de décisions ne l'était pas. Cela signifie que ce dernier ensemble a un cardinal supérieur à celui de l'ensemble $\mathcal{A}$, et donc qu'il existe plus de problèmes de décision que d'algorithmes.

14 Grille de Sudoku

Énoncé p. 45

Lycée Sainte-Clotilde, Strasbourg

1

```
S = [
        [0,1,0,0,7,8,0,0,0],
        [0,8,0,0,4,0,9,0,0],
        [0,0,5,6,0,0,0,1,0],
        [1,0,0,0,6,0,0,0,5],
        [0,4,0,9,1,5,0,7,2],
        [0,6,7,0,8,0,4,0,0],
        [0,0,0,3,0,0,1,0,0],
        [0,7,0,8,9,0,0,2,3],
        [0,0,0,0,0,4,0,0,0]
]
```

2

```
def ligne(S,i):
    return [S[i][j] for j in range(0,9) if S[i][j]
    != 0 ]
```

3

```
def colonne(S,j):
    return [S[i][j] for i in range(0,9) if S[i][j]
    != 0 ]
```

4

```
def bloc(S,i,j):
    r = []
    ligne = (i // 3) * 3
    colonne = (j // 3) * 3
    for l in  [ligne,ligne+1,ligne+2]:
        for c in [colonne,colonne+1,colonne+2]:
            if S[l][c] != 0:
                r.append(S[l][c])
    return r
```

Ici, la difficulté réside dans le choix de la méthode que l'on souhaite adopter. Dans la proposition faite, nous avons fait le choix de calculer les indices de la case haut-gauche du bloc auquel appartient la case (i, j). Pour se faire, on calcule le quotient de i par 3 (si $i = 4$ alors le quotient est 1) et on le multiplie par 3 car les lignes des cases haut-gauche sont toutes 0, 3 ou 6.

Il en est de même pour les colonnes.

5

```
def possibles(S,i,j):
    return [k for k in range(1,10)
            if k not in bloc(S,i,j) and
            k not in ligne(S,i) and
            k not in colonne(S,j)]
```

Il s'agit ici de trouver tous les nombres k compris entre 1 et 9 (`for k in range(1,10)`) qui ne sont pas encore dans le bloc auquel appartient la case (i,j) (`if k not in bloc(S,i,j)`), qui ne sont pas non plus sur la ligne i (`if k not in ligne(S,i)`) ni dans la colonne j (`for k not in colonne(S,j)`).

6

```
def suivante(i,j):
    if j == 8:
        i += 1
        j = 0
    else:
        j += 1

    return i,j
```

7

```
def solve(S,i,j):
    if i == 9:
        return True
    elif S[i][j] > 0:
        i,j = suivante(i,j)
        return solve(S,i,j)
    for k in possibles(S,i,j):
        S[i][j] = k
        a,b = suivante(i,j)
        if solve(S,a,b):
            return True
    S[i][j] = 0
    return False
```

Explications :

- on commence par le plus évident : si $i = 9$, c'est que l'on est arrivé à remplir la grille donc on retourne `True` ;
- sinon, si la case (i,j) est déjà remplie, on passe à la case suivante et on résout la grille ;
- le cas le plus difficile : si la case n'est pas remplie alors on teste chaque chiffre possible et on regarde si la résolution de la grille est possible en passant à la case suivante. Si tel est le cas, on renvoie `True` ;

- si on sort de la boucle, c'est que la grille n'est pas solvable.
  On remplace alors le chiffre de la case (i, j) par 0 (afin de revenir en arrière d'une case) et on retourne `False` pour signifier la non-résolubilité de la grille actuelle.

ANECDOTE

La méthode consistant à tester tous les nombres et à revenir en arrière si la solution n'est pas satisfaisante est appelée *backtracking*.

Téléchargez le programme complet en scannant ce QR code.

Chapitre 3

Méthodes de programmation

Plan du chapitre

Exercice type 1 — Lycée François Villon, Les Mureaux

1 Qu'affiche le programme suivant ?

```
global g
def calculSomme(nb):
  global g
  print("g+{} = {}".format(nb,g+nb))

def calculDiff(nb):
  global g
  print("g-{} = {}".format(nb,g-nb))

g = 200
print("g={}".format(g))
calculSomme(50)
calculDiff(5)
```

2 On modifie la première fonction de la manière suivante :

```
def calculSomme(nb):
  global g
  g = g + nb
  print("g + {} = {}".format(nb,g))
```

Qu'affiche alors le programme principal précédent ?
Qu'est-ce que cette modification a permis de mettre en relief ?

Voir corrigé page 61

COURS

1 Les différents types de variables

1.1 Les variables locales

Définition 1

Une *variable locale* est une variable qui n'existe que durant l'exécution de la fonction l'ayant créée.

Exemple : considérons le programme suivant :

```
def maFonction(x):
    try:
        print("'y' vaut {}.".format(y))
    except NameError:
        print("La variable 'y' n'existe pas encore.")
    y = x
    print("'y' vaut {}.".format(y))

maFonction(10)
print(y)
```

Quand on l'exécute, il s'affiche le message suivant :

```
La variable 'valeur' n'existe pas encore.
'valeur' vaut 10.
NameError: name 'valeur' is not defined
```

Dans la fonction `maFonction`, on *essaie* (« `try` ») d'afficher la valeur contenue dans la variable `y`. Si cette variable n'existe pas (« `except NameError` »), on affiche un message le spécifiant.
Ensuite, on affecte la valeur passée en paramètre à la variable `y`, puis on affiche cette valeur.
On appelle alors la fonction avec le paramètre « 10 ».

Pour finir, on demande d'afficher la valeur de la variable y à l'extérieur de la fonction, mais cette variable n'est pas reconnue (car nous sommes justement à l'extérieur de la fonction).
La variable y est *locale* : elle n'existe que dans la fonction définie, et pas ailleurs.

1.2 Les variables globales

Définition 2

Une *variable globale* est une variable déclarée à l'extérieur d'une fonction et pouvant être utilisée n'importe où dans le programme.

En Python, une variable `var` déclarée hors de toute fonction est une variable *globale*. Elle est alors accessible – en lecture uniquement – dans toute fonction. Pour qu'elle puisse être accessible également en écriture dans une fonction, il faut utiliser explicitement la déclaration `global var`.

Hors des fonctions, les variables globales sont accessibles en lecture et en écriture.

Exemples

Partie 1

```
def afficheA():
    print("a = {}".format(a))

a = 10
afficheA()
```

Partie 2

```
def modifieA():
    global a
    a+=1

a = 10
afficheA()
modifieA()
afficheA()
```

Dans la partie 1, la variable `a` est déclarée à l'extérieur de la fonction `afficheA()` ; pourtant, la fonction affiche bien sa valeur.

Dans la partie 2, `a` est toujours déclarée à l'extérieur des fonctions, mais il est nécessaire de la référencer avec le mot-clé `global` pour pouvoir modifier sa valeur.

2 Programmation impérative

Définition 3

La *programmation impérative* décrit les différentes opérations sous forme de suites d'instructions exécutées par la machine sur laquelle est lancé le programme pour modifier l'état du programme.

C'est le type de programmation (ou *paradigme* de programmation) le plus utilisé. La plupart des processeurs sont construits de manière à exécuter des suites d'instructions.

Définition 4 : effet de bord

Un *effet de bord* (*side effect*) est une modification provoquée par une fonction sur une dépendance partagée (variable ou périphérique par exemple) autre que celle explicitement spécifiée lors de l'appel de cette fonction.

Généralement, un effet de bord aboutit à des valeurs ou des comportements non prévus.

Solution de l'exercice type 1 — Lycée François Villon, Les Mureaux

1 Le programme affiche successivement : « 250 » (résultat obtenu avec l'instruction `calculSomme(50)`) et « 195 » (résultat obtenu avec l'instruction `calculDiff(5)`).

COURS

Solution de l'exercice type 1 (suite) — Lycée François Villon, Les Mureaux

2 Le même programme principal donnera après modification : « 250 » (résultat obtenu avec l'instruction `calculSomme(50)`) et « 245 » (résultat obtenu avec l'instruction `calculDiff(5)`).

On constate que la fonction `calculSomme()` continue de fonctionner et retourne le même résultat. En revanche, la fonction `calculDiff()`, qui n'a pourtant pas été modifiée, retourne une erreur de calcul !

Cela met en relief le fait que `calculDiff()` est *victime* d'un effet de bord suite à la modification de `calculSomme()`.

Voir énoncé page 59

3 Programmation fonctionnelle

3.1 Principe de base

Définition 5

La *programmation fonctionnelle* est un concept de programmation ayant pour but principal de considérer les fonctions comme des données, et donc de pouvoir les passer en paramètres à des fonctions, évitant ainsi les effets de bord.
Une fonction prenant en paramètre une autre fonction est appelée « fonction d'ordre supérieur ».

Python est un langage essentiellement impératif. Il existe des langages essentiellement fonctionnels (LISP, Ocaml, Haskell) mais il est possible de programmer en Python dans un « style fonctionnel ».

Exemple : voici deux programmes écrits en Python de deux manières différentes, mais dont le résultat est identique : incrémenter de 1 une variable. À gauche, la méthode de programmation est impérative, tandis qu'à droite, elle est fonctionnelle.

Impérative

```
variable = 0
def plus():
    global variable
    variable += 1
plus()
print(variable)
```

Fonctionnelle

```
def plus(variable):
    return variable + 1

print(plus(0))
```

3.2 Manipulation sur les listes

3.2.1 - Introduction

En programmation fonctionnelle, l'utilisation de `pop()` ou `append()` sur une liste ne serait pas tolérée car ces fonctions modifient l'état de la liste courante.

À RETENIR

- `liste = liste.append(element)` est remplacé par :

```
liste + [element]
```

- `element = liste.pop()` est remplacé par :

```
liste, element = liste[:-1], liste[-1]
```

3.2.2 - La fonction « map »

En Python, la fonction `map` est un exemple de fonction d'ordre supérieur. Cette fonction prend en argument une fonction `f` et une liste de données `L`.

Elle applique successivement la fonction `f` à chacune des données de la liste et « stocke les résultats obtenus dans une nouvelle liste `G` ».

Ainsi, `G = [ f(L[0]), f(L[1]), ..., f(L[-1]) ]`.

Exemples

- On souhaite connaître la longueur de plusieurs chaînes de caractères.

```
def longueur(liste):
    return list(map(len, liste))

print(longueur(["Bordeaux", "Paris", "Lyon"]))
```

Ce programme affiche : `[8, 5, 4]`.

- On souhaite calculer le carré des longueurs calculées précédemment.

```
def longueur(liste):
    return list(map(len, liste))

def carre(liste):
    return list(map(lambda x: x**2, longueur(liste)))

print(carre(["Bordeaux", "Paris", "Lyon"]))
```

Ce programme affiche : `[64, 25, 16]`.

3.2.3 - La fonction « reduce »

La fonction `reduce` (du module `functools`) est un autre outil de la programmation fonctionnelle. Elle fonctionne de la même façon que `map`, mais à partir d'une fonction `lambda` à deux paramètres : l'élément de la liste à traiter (`x`) et le résultat du traitement précédent (`a`). Elle ne retourne qu'un seul résultat, qui correspond au dernier calcul effectué.

COURS

ATTENTION

Il est à noter que la fonction `lambda` n'est pas exécutée sur le premier élément de la liste (en tant que `x`), pour lequel il n'existe pas encore d'antécédent ! C'est le premier élément de la liste qui sert d'antécédent lors de la première exécution de la fonction `lambda`. Cette toute première exécution de `lambda` par `reduce()` porte donc sur le deuxième élément de la liste (en tant que `x`) avec le premier élément de la liste comme antécédent (devenant `a`).

Exemple : on souhaite calculer le résultat de l'opération :

$$2^2 + 3^2 + 4^2 + 5^2.$$

On utilise alors le programme suivant :

```
from functools import reduce

def somme(liste):
    return reduce(lambda a, x: a + x**2, liste)

print(somme([2,3,4,5]))
```

Ce programme affiche « 52 », et non « 54 » (qui correspondrait au résultat de $2^2 + 3^2 + 4^2 + 5^2$). En effet, ce programme calcule le résultat de l'opération : $2 + 3^2 + 4^2 + 5^2$.

À RETENIR

- La fonction `reduce()`, tout comme la fonction `map()`, exécute de multiples fois la fonction `lambda` : `len(liste)-1` itérations.
- Contrairement à `map()`, les résultats des exécutions successives (hormis le dernier) ne sont pas conservés. Le résultat de l'itération $n - 1$ (de lambda) est utilisé en tant qu'antécédent (`a`) de l'exécution n (de lambda).

C'est cette « mécanique » qui permet d'agréger (de consolider) le résultat final, donc de *réduire* les itérations successives à un unique résultat.

4 Diviser pour régner

Exercice type 2 — Lycée Henri Bergson, Angers

Écrire un algorithme de *tri fusion* permettant de trier une liste de chiffres en la scindant en deux parties à peu près égales, puis en triant récursivement chacune de ces sous-listes.
Écrire le code Python correspondant.

Voir corrigé page 65

Pour trier une liste d'entiers, il vient naturellement à l'esprit de la scinder en deux sous-listes de même taille, qu'on triera séparément, puis de combiner le résultat de chacun de ces tris.

Si par exemple on veut trier la liste `[ 5 2 3 8 4 1 6 9 ]`, on peut diviser cette liste en :

`[ 5 2 3 8 ]` et `[ 4 1 6 9 ]`,

puis les trier séparément :

`[ 2 3 5 8 ]` et `[ 1 4 6 9 ]`,

et ensuite combiner ces résultats pour obtenir la liste complète triée :

`[ 1 2 3 4 5 6 8 9 ]`.

On aurait pu aussi scinder en deux chacune des sous-listes `[ 5 2 3 8 ]` et `[ 4 1 6 9 ]` pour parvenir de manière récursive au même résultat.

Cette méthode est appelée « Diviser pour régner ».

Définition 6 : diviser pour régner

Technique algorithmique consistant à diviser un problème initial en sous-problèmes que l'on résout récursivement ou directement, puis à combiner les résultats.

Elle comporte trois phases :

- *diviser* : on divise le problème initial en plusieurs sous-problèmes,
- *régner* : on résout récursivement ou directement chacun des sous-problèmes,
- *combiner* : on combine les différents résultats obtenus pour parvenir à la solution finale.

Solution de l'exercice type 2

Lycée Henri Bergson, Angers

Un algorithme possible est le suivant :

```
fonction tri(L):
   L : liste à trier
   N ← longueur de L
   Si N = 0 ou N = 1:
     Retourner L
   Sinon:
     M ← E(N/2)
     L1 ← liste des M premiers éléments de L
     L2 ← liste L privée de L1
     Retourner la fusion de tri(L1) et tri(L2)
```

COURS

Solution de l'exercice type 2 (suite) — Lycée Henri Bergson, Angers

En Python, cela donne :

Tri fusion en Python

```
def fusion(L1,L2):
    if L1 == []:
        return L2
    elif L2 == []:
        return L1
    elif L1[0] < L2[0]:
        return [L1[0]] + fusion(L1[1:],L2)
    else:
        return [L2[0]] + fusion(L1,L2[1:])

def tri(L):
    n = len(L)
    if n < 2:
        return L
    else:
        m = n // 2
        return fusion(tri(L[:m]),tri(L[m:]))
```

ATTENTION

Il ne faut jamais oublier d'écrire une condition d'arrêt à une fonction récursive, sous peine d'aboutir à une boucle infinie...

Voir énoncé page 64

5 Recherche textuelle

PROBLÉMATIQUE

Rechercher une chaîne de caractères dans une autre est un problème récurrent, que ce soit en informatique ou dans d'autres sciences comme par exemple en génétique, dès lors qu'il s'agit de localiser un motif dans une séquence d'ADN. Une approche naïve mènerait à un algorithme de complexité conséquente. Comment y remédier ?

5.1 Approche naïve

Nous disposons de la séquence d'ADN suivante :

ATAACAGAGAATAAGGCTAGGAAATACTGAA

Nous souhaitons savoir si le motif « AGGCTA » apparaît dans cette séquence.

L'approche naïve consiste à effectuer une comparaison du motif avec la séquence en partant de la droite et en décalant le motif vers la droite jusqu'à trouver une coïncidence.

Comparaison	Coïncidence
ATAACAGAGAATAAGGCTAGGAAATACTGAA AGGCTA	Non
ATAACAGAGAATAAGGCTAGGAAATACTGAA •AGGCTA	Non
ATAACAGAGAATAAGGCTAGGAAATACTGAA ••AGGCTA	Non
⋮	
ATAACAGAGAATAAGGCTAGGAAATACTGAA ••••••••••••AGGCTA	Oui

Dans cette approche, on aligne donc le motif avec la séquence sur la gauche et on compare les caractères de la gauche vers la droite.

Ainsi, pour la première comparaison, les premiers caractères (A) coïncident, mais pas les deuxièmes ; on décale donc le motif d'un caractère vers la droite.

Le « A » se retrouve donc sous un « T » : les deux caractères ne coïncident pas, donc on décale à nouveau.

On fait cela jusqu'à obtenir une parfaite coïncidence.

En Python, cela donne :

Approche naïve

```
def comparaison(sequence, motif):
    for i in range(len(sequence) - len(motif) + 1):
        if sequence[i:i+len(motif)] == motif:
            return sequence[:i]+'['+motif+']'+sequence[i+
    len(motif)+1:]

    return False

sequence = "ATAACAGAGAATAAGGCTAGGAAATACTGAA"
motif = "AGGCTA"

print(comparaison(sequence,motif))
```

qui retourne : `ATAACAGAGAATA[AGGCTA]GAAATACTGAA`.

La complexité de cet algorithme est $O\big(m(s-m)\big)$, où m et s représentent respectivement le nombre de lettres du motif et de la séquence.

COURS

5.2 L'algorithme de Boyer-Moore

5.2.1 - Approche à travers un exemple

Nous allons expliquer cet algorithme à travers l'exemple de la séquence et du motif précédent.

- On commence par aligner la séquence et le motif à gauche, comme précédemment, mais la comparaison se fait sur le dernier caractère du motif :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA

  Ce dernier caractère coïncide avec le 6e de la séquence. On compare alors les deux caractères précédents :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA

  Ils ne coïncident pas.
- Le caractère rencontré dans la séquence (C) apparaît tout de même dans le motif : nous allons donc décaler le motif de sorte que ce caractère (C) coïncide avec le même caractère du motif :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA
- On recommence alors la comparaison en partant de la droite du motif :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA

  Ils ne coïncident pas.
- Mais « G » apparaît dans le motif donc on décale ce dernier de sorte à faire coïncider le « G » de la séquence avec le premier « G » du motif rencontré en partant de la droite :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA
- On recommence alors la comparaison en partant de la droite du motif :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA

  Les caractères de droite coïncident, mais pas leurs précédents.
- On décale donc le motif de sorte que le « G » de la séquence coïncide avec le premier « G » du motif en partant de droite :

  ATAACAGAGAATAAGGCTAGGAAATACTGAA
  AGGCTA

  Les derniers caractères ne coïncident pas.

- On décale donc le motif de sorte que le « T » de la séquence coïncide avec le « T » du motif qui est juste avant :

ATAACAGAGAATAAGGCTAGGAAATACTGAA
AGGCTA

Les deux derniers caractères coïncident mais pas les troisièmes.

- On décale donc le motif de sorte à faire coïncider le « A » de la séquence avec le « A » du motif qui est à gauche des caractères qui coïncidaient (donc avec le « A » de gauche du motif) :

ATAACAGAGAATAAGGCTAGGAAATACTGAA
AGGCTA

Les derniers caractères ne coïncident pas.

- On décale donc le motif de sorte à aligner le « G » de la séquence avec celui du motif qui est le plus à droite :

ATAACAGAGAATAAGGCTAGGAAATACTGAA
AGGCTA

Il y a alors correspondance entre le motif et le bout de séquence. L'algorithme s'arrête donc.

5.2.2 - Table de sauts

À travers l'exemple vu précédemment, on peut facilement comprendre l'utilité de construire une première table qui indique la distance maximale de chaque caractère distinct au dernier dans le motif : on appelle cette table la « première table de sauts », ne dépendant que du motif et pas de la séquence.

On commence par l'avant-dernier caractère (en effet, le dernier caractère du motif n'est pas pris en compte pour cette table) : le motif étant « AGGCTA », le caractère « T » est à une distance de 1 du dernier « A », etc.

On obtient la table suivante :

Caractère	Distance
T	1
C	2
G	4
A	5
Autres	6

Tableau 3.1 – 1re table de sauts du motif « AGGCTA »

Remarque : pour un motif où les caractères sont tous distincts, on aura par exemple :

Caractère	Distance
C	1
B	2
A	3
Autres	4

Tableau 3.2 – 1re table de sauts du motif « ABCD »

Cette première table concerne la *séquence*, bien qu'elle soit construite sur le *motif*, car elle donne le décalage du motif à faire sur la séquence (on doit décaler le motif de 1, 2, 3,... sur la séquence).

COURS

5.2.3 - Programme en Python

La table de sauts précédemment vue nous donne l'idée de construire un dictionnaire contenant toutes les lettres de l'alphabet considéré ainsi que leur premier rang dans le motif à trouver, puis de s'appuyer sur ces valeurs pour décaler le motif. C'est le but de la fonction `table_sauts` du programme ci-dessous.

```
def table_sauts(motif, alphabet):
    distance_car = dict()
    longueur_motif = len(motif)
    for caractere in alphabet:
        try:
            distance_fin = longueur_motif -  motif.index(
    caractere) - 1
            if distance_fin > 0:
                distance_car[caractere] = distance_fin
        except ValueError:
            # si le caractère n'existe pas dans le motif,
    on peut décaler de tout le mot
            distance_car[caractere] = longueur_motif
    return distance_car

def boyer_moore(seq, motif, alphabet):
    lg_motif, lg_seq = len(motif) , len(seq)
    saut = table_sauts(motif, alphabet)
    print(saut)
    position = 0
    while(position <= lg_seq - lg_motif):
        indice_car = lg_motif - 1
        while (indice_car >= 0) and ( motif[indice_car] ==
    seq[position + indice_car] ):
            indice_car -= 1
        if indice_car < 0:
            print("Le motif a été trouvé en position {}".
    format(position))
            position += saut[seq[position + lg_motif - 1]]
        else:
            position += saut[seq[position + indice_car]]

sequence = 'ATAACAGAGAATAAGGCTAGGCTAAAATACTGAAGGCTA'
motif = 'AGGCTA'
alphabet = 'ACTGI'
boyer_moore(sequence , motif , alphabet)
```

Programmation fonctionnelle

1 Réécriture

★ — 5 min — Corrigé p. 76

Lycée André Theuriet, Civray

Réécrire de façon fonctionnelle le programme suivant :

```
L = [ 'Jean', 'Luc', 'Lucie', 'Marie' ]
for i in range(len(L)): L[i] = hash(L[i])
```

Remarque : la fonction `hash` renvoie la valeur de hachage d'un objet. Une fonction de hashage produit un condensé des données de l'objet. Ce condensé est de taille fixe, dont la valeur diffère suivant la fonction utilisée.

2 Fonction carré

★ — 5 min — Corrigé p. 76

Lycée Camille Claudel, Blois

Écrire en Python de manière fonctionnelle une fonction `carre(n)` qui affiche sous forme de liste le carré des entiers de 0 à n (compris).

3 Compteur d'occurrences

★★ — 10 min — Corrigé p. 76

Lycée Henri Alain-Fournier, Bourges

À l'aide de la fonction `reduce` du module `functools`, écrire en Python une fonction `compteur(liste,mot)` qui retourne le nombre d'occurrences de la chaîne `mot` dans la liste `liste`.

4 Taille d'une liste

★★ — 10 min — Corrigé p. 77

Lycée Sainte-Anne, Brest

Supposons que la fonction `len` n'existe pas.

Écrire, en Python et en utilisant le paradigme de la programmation fonctionnelle, une fonction récursive `taille(liste)` admettant une liste comme argument et retournant sa taille.

5 Calcul d'une taille moyenne

★★ — 15 min — Corrigé p. 77

Lycée Raoul Follereau, Nevers

On dispose d'une liste de dictionnaires `gens` :

```
gens = [
      'nom': 'Pierre' , 'taille': 178,
      'name': 'Marie' , 'taille': 181,
      'name': 'Paul' , 'taille': 157
]
```

INTERROS

Écrire en Python de manière fonctionnelle une fonction `average(gens)` admettant une liste de dictionnaires de la forme décrite précédemment et retournant la taille moyenne des personnes figurant dans cette liste.

Aide : on utilisera les fonctions :

`reduce` et `filter(fonction,dictionnaire)`,

cette dernière fonction renvoyant un dictionnaire contenant tous les éléments du dictionnaire d'origine pour lesquelles `fonction = True`.

On pourra aussi importer la fonction `add` du module `operator` de sorte que `reduce(add,liste)` retourne la somme des éléments de `liste`.

6 Une fonction « pipeline_each »

★★★ 10 min Corrigé p. 78

Lycée François Mauriac, Bordeaux

On souhaite écrire en Python une fonction `pipeline_each(liste,fonctions)` comme suit :

- son premier paramètre (`liste`) reçoit une liste de chaînes de caractères ;
- son second paramètre (`fonctions`) reçoit une liste d'une ou plusieurs fonctions qui, chacune, reçoit une liste de chaînes en argument et y effectue un traitement.

La fonction `pipeline_each(liste,fonctions)` doit appliquer, tour à tour, chacune des fonctions de la liste `fonctions` à la liste de chaînes de caractères `liste`. Puis elle doit renvoyer cette liste à l'issue des traitements successifs.

Compléter le programme suivant afin d'avoir en affichage :

`['Le Monde Est À Nous;', ' Il Ne Faut Pas Le Détériorer;']`

```
from functools import reduce

def pipeline_each(liste, fonctions):
    ...

def format1(chaine):
    return chaine.title()

def format2(chaine):
    return chaine.replace('.', ';')

print(list(pipeline_each(['Le monde est à nous.',
         ' Il ne faut pas le détériorer.'],
        [format2,format1])))
```

Diviser pour régner

7 Fonction « maximum » d'une liste

★ — 15 min — Corrigé p. 78

Lycée Jean-Joseph Fourier, Auxerre

On dispose d'une liste `L` de nombres.

En utilisant la méthode du « diviser pour régner », implémenter une fonction `maximum(L,debut,fin)` retournant le maximum de `L`.

8 Le tri fusion

★★ — 15 min — Corrigé p. 78

Lycée Richelieu, Versailles

Le *tri fusion* est un algorithme de tri efficace qui suit le paradigme « diviser pour régner ». Il divise la liste à trier en deux moitiés, trie chaque moitié récursivement, puis fusionne les deux moitiés triées pour obtenir la liste triée complète.

Notre objectif est d'implémenter l'algorithme de tri fusion en Python.

Instructions :

1. Écrivez une fonction `tri_fusion(liste)` qui prend une liste d'entiers en entrée et retourne une nouvelle liste triée.
2. La fonction doit suivre les étapes suivantes :
   - Si la liste est vide ou ne contient qu'un seul élément, elle est déjà triée. Retournez-la telle quelle.
   - Divisez la liste en deux moitiés.
   - Triez récursivement chaque moitié en appelant `tri_fusion` sur chaque moitié.
   - Fusionnez les deux moitiés triées pour obtenir la liste triée complète.
3. Écrivez une fonction auxiliaire `fusion(gauche, droite)` qui prend deux listes triées en entrée et retourne une seule liste triée contenant tous les éléments des deux listes.
4. Testez votre fonction `tri_fusion` avec plusieurs listes d'entiers pour vérifier qu'elle fonctionne correctement.

9 Recherche binaire récursive

★★ — 20 min — Corrigé p. 79

Lycée François Mauriac, Bordeaux

La recherche binaire est un algorithme efficace pour trouver un élément dans une liste triée. Elle fonctionne en divisant la liste en deux à chaque étape et en déterminant dans quelle moitié se trouve l'élément recherché.

INTERROS

Notre objectif est d'implémenter la recherche binaire de manière récursive en Python.

Instructions :

1. Écrivez une fonction `recherche_binaire(liste, element, debut, fin)` qui prend en entrée une liste triée `liste`, un élément à rechercher `element`, et les indices de début `debut` et de fin `fin` de la portion de la liste à rechercher. La fonction doit retourner l'indice de l'élément s'il est trouvé, sinon `-1`.
2. La fonction doit suivre les étapes suivantes :
   - Calculez l'indice du milieu de la portion de la liste.
   - Comparez l'élément au milieu avec l'élément recherché.
   - Si l'élément au milieu est égal à l'élément recherché, retournez l'indice du milieu.
   - Si l'élément recherché est inférieur à l'élément au milieu, appelez récursivement la fonction sur la moitié gauche de la liste.
   - Si l'élément recherché est supérieur à l'élément au milieu, appelez récursivement la fonction sur la moitié droite de la liste.
   - Si la portion de la liste est vide (c'est-à-dire que `debut` dépasse `fin`), retournez `-1`.
3. Testez votre fonction `recherche_binaire` avec plusieurs listes triées et éléments pour vérifier qu'elle fonctionne correctement.

10 Algorithme de Karatsuba

★★★ 30 min Corrigé p. 80

Lycée polyvalent Émile Zola, Aix-en-Provence

L'objectif de cet exercice est d'utiliser l'algorithme de Karatsuba afin de trouver le produit de deux polynômes P et Q.

On rappelle qu'un *polynôme en X de degré n* est une expression de la forme :

$$P(X) = \sum_{k=0}^{n} a_k X^k.$$

On représente alors en Python le polynôme P par la liste `[a0,a1,..,an]` des coefficients a_0, a_1, ..., a_n.

1. Écrire une fonction `standardise(P,Q)` qui retourne un couple de deux listes de dimension $\max\big(\deg(P); \deg(Q)\big)$, où $\deg(P)$ représente le degré de P.

   Par exemple, si $P(X) = 3 + 2X$ et $Q(X) = 5 - 7X + 4X^2$ (soit en Python : `P = [3,2]` et `Q = [5,-7,4]`), la fonction devra retourner : `([3,2,0] , [5,-7,4])`.

2 Écrire une fonction `add(P,Q)` qui retourne une liste dont les éléments sont les coefficients de la somme des deux polynômes P et Q, puis une fonction `sous(P,Q)` qui retourne sous forme de liste le résultat de $P(X) - Q(X)$.

Par exemple, si $P(X) = 3 + 2X$ et $Q(X) = 5 - 7X + 4X^2$ alors :

- `add(P,Q) = [8,-5,4]` ;
- `sous(P,Q) = [-2,9,-4]`.

3 L'algorithme de Karatsuba consiste à décomposer P et Q de la manière suivante :

$$\begin{cases} P(X) &= X^k P_1(X) + P_0(X), \quad \deg P_0 \leqslant k \\ Q(X) &= X^k Q_1(X) + Q_0(X), \quad \deg Q_0 \leqslant k \end{cases}$$

où $k = E\left(\frac{n}{2}\right)$ (partie entière de la moitié de `n`).

(a) On suppose que :

$$\begin{aligned}(PQ)(X) = {} & (P_0Q_0)(X) \\ & + \big((P_0 + P_1)(Q_0 + Q_1) - P_0Q_0 - P_1Q_1\big)(X)X^k \\ & + (P_1Q_1)(X)X^{2k}.\end{aligned}$$

Écrire une fonction `prodmonome(P,k)` qui renvoie sous forme de liste le produit du polynôme P par X^k.

(b) En déduire une fonction `prod(P,Q)` qui renvoie sous forme de liste le produit de deux polynômes P et Q.

(c) Tester cette fonction sur les polynômes :
$P(X) = 5-2X^4+3X^3-X^2+2X-1$ et $Q(X) = -3+5X+2X^2$.

11 Le problème du sac à dos

★★★ 30 min Corrigé p. 81

Lycée Camille Jullian, Bordeaux

On dispose d'un sac à dos ne pouvant supporter qu'un poids P. On dispose de plus d'une collection de n objets, chaque objet ayant un poids p_i et une valeur v_i, $1 \leqslant i \leqslant n$, avec $p_1 + p_2 + \cdots + p_n > P$.

On souhaite mettre dans le sac le plus d'objets possible tout en maximisant la valeur totale des objets.

En utilisant une approche récursive « diviser pour régner », donner une solution en Python qui affiche la valeur maximale.

Appliquer le programme au cas où l'ensemble des objets (poids,valeur) est :

$$E = \{(12,14); (1,2); (4,10); (1,1); (2,2)\}$$

et où $P = 15$.

CORRIGÉS

1 Réécriture

Énoncé p. 71

Lycée André Theuriet, Civray

Voici une possibilité :

```
L = [ 'Jean', 'Luc', 'Lucie', 'Marie' ]

list(map(hash, L))
```

2 Fonction carré

Énoncé p. 71

Lycée Camille Claudel, Blois

Une possibilité est la suivante :

```
def carre(n):
    return list(map(lambda x : x*x, range(n + 1)))

print(carre(5))
```

3 Compteur d'occurrences

Énoncé p. 71

Lycée Henri Alain-Fournier, Bourges

Une possibilité est la suivante :

```
from functools import reduce

def compteur(liste,mot):
    return reduce(lambda a, x : a + x.count(mot), liste,
     0)

print(compteur(['Le coq Hardi est le plus beau du
    poulailler ', 'Je pense que le voisin est un geek '
    ], 'le'))
```

Dans cet exemple, le programme retourne « 3 »; ce n'est pas un bug : le premier « Le » n'est pas compté (car il a une majuscule) mais dans le mot « poulailler », la chaîne « le » apparaît une fois. Il y a donc bien trois occurrences de « le » dans la liste.

4 Taille d'une liste

Énoncé p. 71

Lycée Sainte-Anne, Brest

Une première possibilité est la suivante :

Version itérative

```
def taille(liste):
    s = 0
    for x in liste: s += 1
    return s
```

Une deuxième possibilité est, en utilisant la récursivité :

Version fonctionnelle récursive

```
def taille(liste):
    if liste == []:
        return 0
    return 1 + taille(liste[1:])
```

5 Calcul d'une taille moyenne

Énoncé p. 71

Lycée Raoul Follereau, Nevers

Voici une possibilité :

```
from operator import add
from functools import reduce

def taille_moyenne(population):
    tailles = list(map(lambda personne: personne['
    taille'], gens))
    total_tailles = reduce(add, tailles)
    return total_tailles / len(tailles)

gens = [
{'name': 'Pierre' , 'taille': 178},
{'name': 'Marie' , 'taille': 181},
{'name': 'Paul' , 'taille': 157}
]

print(taille_moyenne(gens)) # affiche "172.0"
```

Ce programme affiche « 172 », qui correspond à la taille moyenne des trois personnes du dictionnaire.

CORRIGÉS

6 Une fonction « pipeline_each »

Énoncé p. 72

Lycée François Mauriac, Bordeaux

Une solution envisageable est la suivante :

```
from functools import reduce

def pipeline_each(liste, fonctions):
    return reduce(lambda a, x: map(x, a), fonctions,
    liste)

def format1(chaine):
    return chaine.title()

def format2(chaine):
    return chaine.replace('.', ';')

print(list(pipeline_each([ 'Le monde est à nous.',
          ' Il ne faut pas le détériorer.' ],
        [format2,format1])))
```

7 Fonction « maximum » d'une liste

Énoncé p. 73

Lycée Jean-Joseph Fourier, Auxerre

L'idée consiste à diviser en deux la liste L en deux parties et à trouver le maximum des deux parties. Le maximum global sera la plus grande des valeurs ainsi trouvées. On doit partager en deux jusqu'à obtenir des listes élémentaires (à 1 élément). On doit alors utiliser la récursivité.

Une fonction possible est alors la suivante :

```
def maximum(L,debut,fin):
    if debut == fin:
        return L[debut]
    milieu = (debut + fin) // 2
    x = maximum(L, debut, milieu)
    y = maximum(L, milieu + 1, fin)
    return x if x > y else y
```

8 Le tri fusion

Énoncé p. 73

Lycée Richelieu, Versailles

Le programme proposé page ci-contre répond aux instructions de l'énoncé.

```
def tri_fusion(liste):
    if len(liste) <= 1: return liste
    milieu = len(liste) // 2
    gauche = tri_fusion(liste[:milieu])
    droite = tri_fusion(liste[milieu:])

    return fusion(gauche, droite)

def fusion(gauche, droite):
    resultat = []
    i = 0
    j = 0
    while i < len(gauche) and j < len(droite):
        if gauche[i] < droite[j]:
            resultat.append(gauche[i])
            i += 1
        else:
            resultat.append(droite[j])
            j += 1
    resultat.extend(gauche[i:])
    resultat.extend(droite[j:])

    return resultat
```

9 Recherche binaire récursive

Énoncé p. 73

Lycée François Mauriac, Bordeaux

Voici une proposition répondant aux instructions :

```
def recherche_binaire(liste, element, debut, fin):
    if debut > fin:
        return -1

    milieu = (debut + fin) // 2

    if liste[milieu] == element:
        return milieu
    if liste[milieu] > element:
        return recherche_binaire(liste, element, debut,
     milieu - 1)

    return recherche_binaire(liste, element, milieu + 1,
     fin)
```

CORRIGÉS

10 Algorithme de Karatsuba

Énoncé p. 74

Lycée polyvalent Émile Zola, Aix-en-Provence

1 Une fonction possible est la suivante :

```
def standardise(P, Q):
    n, m = len(P), len(Q)
    if n < m: P = P + [0] * (m - n)
    else: Q = Q + [0] * (n - m)
    return P, Q
```

2 Une fonction possible est la suivante :

```
def add(P, Q):
    P, Q = standardise(P, Q)
    return [ P[i] + Q[i] for i in range(len(P)) ]

def sous(P, Q):
    P, Q = standardise(P, Q)
    return [ P[i] - Q[i] for i in range(len(P)) ]
```

3 **(a)** Une fonction possible est la suivante :

```
def prodmonome(P, k):
    return [0] * k + P
```

En effet,

$$P(X)X^k = X^k \sum_{i=0}^{n} a_i X^i = \sum_{i=0}^{n} a_i X^{i+k} = \sum_{i=k+1}^{n+k} a_{i-k-1} X^i.$$

Ainsi, a_0 n'est plus le coefficient de X^0 mais de X^{0+k}, a_1 est celui de X^{1+k}, etc.

(b)

```
def prod(P, Q):
    n = max(len(P), len(Q))
    if n == 1:
        return [P[0]*Q[0]]
    P, Q = standardise(P, Q)
    k = n // 2
    P0, P1 = P[0:k], P[k:n]
    Q0, Q1 = Q[0:k], Q[k:n]
    R0 = prod(P0, Q0)
    R1 = prod(P1, Q1)
    R2 = prod(add(P0, P1), add(Q0, Q1))
    R2 = sous(sous(R2, R1), R0)
    R1 = prodmonome(R1, 2 * k)
    R2 = prodmonome(R2, k)
    return add(add(R0, R1), R2)
```

(c) On définit dans cette question P et Q ainsi :

```
P , Q = [5,-2,3,-1,2,-1] , [-3,5,2]
```

En appelant la fonction `prod`, on obtient la liste :

```
[-15, 31, -9, 14, -5, 11, -1, -2, 0, 0, 0]
```

ce qui signifie que :

$$PQ(X) = -15 + 31X - 9X^2 + 14X^3 - 5X^4 + 11X^5 - X^6 - 2X^7.$$

Télécharger le programme complet.

11 Le problème du sac à dos

Énoncé p. 75

Lycée Camille Jullian, Bordeaux

Pour résoudre un tel problème, il faut avant tout passer par une formalisation mathématique : une traduction algébrique.

- Notons $E = \{(p_1, v_1); \cdots; (p_n, v_n)\}$ l'ensemble des n objets.
- La solution attendue sera un n-uplet $(x_1; x_2; \ldots; x_n)$ où $x_i = 0$ si l'on ne prend pas l'objet i et $x_i = 1$ dans le cas contraire.
- Il faut alors que $\sum_{i=1}^{n} x_i p_i \leqslant P$ et $\sum_{i=1}^{n} x_i v_i$ maximale.

Nous avons vu dans le livre de Première NSI qu'une approche gloutonne était peu recommandée car dans certains cas, la solution donnée n'est pas optimale.

On nous demande ici une approche récursive de type « diviser pour régner ». Il est alors important dans ce type d'exercices d'établir une formule de récurrence. Le raisonnement prend nécessairement en compte le poids et la valeur d'un objet.

Notons $V(i, p)$ la valeur maximale des objets que l'on peut mettre dans un sac de capacité p en ne prenant que les i premiers objets. Ce qui nous intéresse est alors $V(n, P)$. On a alors :

- $V(0, p) = 0$ car s'il n'y a pas d'objet, la valeur maximale est nécessairement 0 ;
- Si $p_i > p$ alors $V(i, p) = V(i-1, p)$ car si le poids de l'objet i est supérieure au poids maximal, on ne l'ajoute pas et la valeur maximale des objets reste inchangée par rapport à celle de l'étape précédente ;
- Enfin, si $p_i \leqslant p$ alors $V(i, p) = \max\big(V(i-1, p); V(i-1, p-p_i) + v_i\big)$ car il faut regarder la valeur obtenue en ajoutant la valeur de l'objet i (v_i) à la valeur maximale correspondant à $i-1$ objets dans un sac de capacité $p - p_i$.

Ces relations de récurrence nous mènent alors à l'implémentation suivante :

```
def sac(E,P,i):
    if i == 0: return 0
    if E[i-1][0] > P: return sac(E, P, i-1)
    else: return max(sac(E, P, i-1), E[i-1][1] +
                     sac(E, P-E[i-1][0], i-1))
def initSac(E,P):
    return sac(E, P, len(E))
```

Cela donne avec notre exemple :

```
>>> E = [ (12,14) , (1,2) , (4,10) , (1,1) , (2,2) ]
>>> P = 15
>>> initSac(E,P)
18
```

« 18 » est en effet la valeur maximale que l'on obtient en faisant $14 + 2 + 2$ (objets 1, 2 et 5).

Chapitre 4

Programmation orientée objet, programmation dynamique

Plan du chapitre

1 Programmation orientée objet (POO)

Exercice type 1 — Lycée Henri Alain-Fournier, Bourges

On considère une classe `Personnage` représentant un personnage de jeu.
Le plateau du jeu est représenté par un repère orthonormé à trois axes. La position du joueur dans le plateau est repérée par ses attributs x, y et z.
Écrire les méthodes `avance`, `droite` et `saute` permettant respectivement de faire avancer, aller à droite et sauter le personnage, c'est-à-dire d'augmenter de 1 respectivement x, y et z.
Implémenter de plus une autre méthode `coord` renvoyant les coordonnées sous forme d'un triplet.

Voir corrigé page 88

PROBLÉMATIQUE

Comment modéliser des éléments du monde réel sous forme d'objets informatiques afin de les manipuler de manière simple ?

1.1 Historique et définitions

La *programmation orientée objet* (en abrégé : POO) est un paradigme de programmation qui consiste à considérer un programme comme un ensemble d'objets en interaction.

COURS

Les *objets* peuvent représenter un concept du monde réel (comme un meuble, une personne ou encore un livre par exemple).

Pour faire de la POO, on peut utiliser un *langage à classes*, comme Python.

La POO trouve son origine en Norvège au début des années 1960. Elle fut élaborée par les informaticiens Ole-Johan Dahl et Kristen Nygaard.

Dans les années 1970, elle fut reprise par l'informaticien américain Alan Kay qui contribua à son essor. Cependant, il fallu attendre les années 1990 pour que ses principes soient clairement formalisés.

Retrouvez une présentation originale de la programmation orientée objet en vidéo.

1.2 Les classes sous Python

1.2.1 - Introduction

Nous allons considérer un objet `Individu` qui représente une personne.

Cette personne a plusieurs caractéristiques. Par exemple :

- sa date de naissance ;
- son genre ;
- sa taille ;
- sa masse ;
- etc.

qui vont représenter les *variables* de la personne.

De plus, cette personne peut faire plusieurs choses :

- parler ;
- manger ;
- marcher ;
- etc.

qui vont correspondre à des *fonctions* propres à elle-même.

Pour définir cette personne ainsi que les fonctions qui lui sont associées, on peut définir une *classe*.

1.2.2 - Définitions et vocabulaire

Définition 1

Une *classe* est un type qui définit en son sein :

- des variables d'instances (appelées aussi *attributs*) ;
- des méthodes (les fonctionnalités associées à la classe).

Exemple : voici un exemple en Python où l'on définit une classe `Personne` à l'aide de trois caractéristiques (année de naissance, taille en cm, masse en kg) et où l'on définit une fonction propre à la classe (grandir) qui admet un paramètre numérique (exprimé en cm).

```
class Personne:
    def __init__(self, birthyear, tall, weight):
        self.naissance = birthyear
        self.taille = tall
        self.poids = weight

    def grandir(self,h):
        self.taille += h

luc = Personne(2002,178,69)
init_tall = luc.taille
luc.grandir(8)
new_tall = luc.taille
print("Luc pèse", luc.poids, "kg.")
print("Luc mesure à présent", new_tall, "cm après avoir
    grandi de 8 cm en 1 an. Avant ça, il mesurait:",
    init_tall, "cm.")
```

Ce programme affiche :

```
Luc pèse 69 kg.
Luc mesure à présent 186 cm après avoir grandi de 8 cm en
1 an. Avant ça, il mesurait: 178 cm.
```

Définitions 2

Dans l'exemple précédent,

- `naissance`, `taille` et `poids` sont les *attributs* de la classe `Personne` ;
- `grandir` est une *méthode* de la classe `Personne`.

COURS

On accède aux attributs d'une classe à l'aide de la syntaxe :

```
<nom de la classe>.<nom de l'attribut>
```

On accède aux méthodes d'une classe à l'aide de la syntaxe :

```
<nom classe>.<nom méthode>(<arguments potentiels>)
```

1.2.3 - Constructeur d'une classe

Définition 3

Le *constructeur* d'une classe est une fonction :

- dont le nom est nécessairement `__init__` ;
- admettant au moins un paramètre nommé `self` placé obligatoirement en premier s'il y en a plusieurs.

Le paramètre `self` représente l'objet cible : c'est une variable qui contient une référence vers l'objet qui est en cours de création. Grâce à ce dernier, on va pouvoir accéder aux attributs et fonctionnalités de l'objet cible.

1.2.4 - Instanciation et variables d'instance

Une classe est une sorte de moule qui sert à fabriquer des objets.

L'*instanciation* est l'opération qui consiste à créer un objet. L'objet ainsi créé est alors appelé une *instance* de la classe.

Exemple : si on reprend la classe `Personne` de l'exemple précédent, en définissant :

```
luc = Personne(2002,178,69)
```

nous avons défini une instance de la classe `Personne`.

On peut ainsi définir plusieurs instances d'une même classe, chacune ayant des attributs qui lui sont propres.

Si on demande d'afficher la variable `luc`, on voit apparaître quelque chose de la forme :

```
<__main__.Personne object at 0x02CDDC50>
```

Cela signifie que `luc` est une référence vers l'objet créé, c'est-à-dire une indication sur son emplacement en mémoire. Ici, par exemple, l'objet `Personne` se trouve en mémoire à la position 0x02CDDC50.

Un attribut de la classe est représenté par une variable appelée *variable d'instance*, accessible à l'aide du paramètre `self`. L'idée, dans notre exemple, est que le constructeur initialise ces variables d'instances, qui seront alors stockées dans l'objet et en mémoire pour toute la durée de vie de l'objet.

Il est important de faire la différence entre les deux types de variables qui se trouvent dans le code de ce constructeur :

- la variable `self.taille` (par exemple) représente la variable d'instance, c'est-à-dire celle associée à l'objet, qui existe à partir de la création de l'objet jusque sa destruction ;
- la variable `taille` représente le paramètre reçu par le constructeur et n'existe que dans le corps de ce dernier.

Le paramètre `self` permet donc d'accéder aux variables d'instance, c'est-à-dire aux attributs de l'objet cible, depuis le constructeur.

1.2.5 - Méthodes d'une classe

Définition 4

Une *méthode* d'une classe est une fonction définie au sein de la classe et admettant au moins le paramètre `self` (obligatoirement en premier s'il y a plusieurs paramètres).

Le paramètre `self` représente l'objet cible sur lequel la méthode est appelée. Il permet notamment d'avoir accès aux variables d'instance de l'objet.

Dans l'exemple précédent, `grandir(self,h)` est une méthode admettant deux paramètres, dont le second est un nombre. Cette méthode redéfinit la variable d'instance `self.taille` en lui ajoutant la valeur passée en second paramètre.

Une méthode est appelée en spécifiant l'objet cible (ici, `luc`) suivi de la méthode en utilisant l'opérateur d'appel : le point. Cela donne ici :

luc.grandir(8)

1.2.6 - Redéfinition des opérations (hors-programme)

Maintenant que l'on sait définir un objet à l'aide d'une classe, on peut s'intéresser aux opérations que l'on peut faire sur ces objets.

Par exemple, si on souhaite définir un objet `vector` représentant un vecteur du plan, il serait utile de pouvoir définir la somme de deux de ces vecteurs, qui est aussi un objet de classe `vector` (car la somme de deux vecteurs du plan est aussi un vecteur du plan).

Python permet de définir une telle opération de la manière suivante :

```
class Vector:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def __add__(self,other):
        return Vector(self.x + other.x, self.y + other.y)
```

COURS

En mode console, cela donne par exemple :

```
>>> u , v = vector(-2,8) , vector(3,-5)
>>> w = u + v
>>> print("w({},{})".format(w.x,w.y))
w(1,3)
```

Cette capacité du langage Python s'appelle la *surcharge d'opérateur*.

1.2.7 - Encapsulation

La classe `vector` définie précédemment aurait pu être définie autrement :

```
class vector:
    def __init__(self,x,y):
        self.coord = (x, y)

    def __add__(self,other):
        return Vector(self.coord[0] + other.coord[0], self.
    coord[1] + other.coord[1])
```

On est passé de deux variables d'instance à une seule, mais cette différence ne doit pas être perçue par l'utilisateur : la manière dont on gère les valeurs passées en paramètres au sein de la classe ne doit pas être vue « de l'extérieur ». Ce principe fondamental de la POO est appelé *encapsulation*. Cela peut être réalisé grâce à la définition de méthodes adéquates.

Solution de l'exercice type 1 — Lycée Henri Alain-Fournier, Bourges

Une définition possible de la classe `Personnage` est la suivante :

```
class Personnnage:
    def __init__(self,x,y,z):
       self.x, self.y, self.z = x, y, z
    def avance(self):
        self.x += 1
    def droite(self):
        self.y += 1
    def saute(self):
        self.z += 1
    def coord(self):
        return (self.x, self.y, self.z)
```

...

Solution de l'exercice type 1 (suite) — Lycée Henri Alain-Fournier, Bourges

En utilisant le code, voici ce que donne un exemple d'utilisation :

```
>>> Lara = Personnnage(0,0,0)
>>> Lara.avance()
>>> Lara.saute()
>>> Lara.coord()
(1,0,1)
```

Voir énoncé page 83

2 Programmation dynamique

Exercice type 2 — Lycée André Theuriet, Civray

Étant donné un système de monnaie S comportant des pièces de valeurs différentes, écrire un programme qui affiche le nombre *minimal* de pièces à utiliser pour rendre une somme donnée, ainsi que leur énumération.

Voir corrigé page 94

2.1 Introduction

Il s'agit d'un problème d'optimisation, puisqu'on veut utiliser un nombre de pièces et de billets minimal. A travers l'étude de ce problème d'algorithmique classique, nous allons voir qu'il existe de multiples possibilités d'y répondre. La programmation dynamique est la plus efficace.

2.2 Approche gloutonne

C'est l'approche que font bon nombre de commerçants quand il rendent la monnaie à leurs clients.

Elle consiste à prendre le billet ou la pièce de valeur maximale en dessous de la somme à restituer, puis à recommencer ce geste par rapport à la différence. Cela donne l'algorithme suivant :

```
V ← valeur à atteindre
Tant que V n'est pas nulle:
   Choisir le billet ou la pièce de plus grande valeur v_i
   telle que v_i ⩽ V
   V ← V - v_i
Fin du Tant que
```

COURS

Exemple : comment rendre la somme de 14 € dans un système monétaire européen simplifié constitué uniquement de pièces de 1, 2, 5 et 10 € ?

On a alors $S = \{1; 2; 5; 10\}$.

Pour rendre 14 €, la plus grande valeur possible à déduire est 10 €, ce qui donne $14 - 10 = 4$.

La plus grande valeur possible à déduire de 4 € est 2×2 €, ce qui donne $4 - 2 \times 2 = 0$.

La solution renvoyée par notre algorithme est donc le quadruplet (0; 2; 0; 1), soit un total de 3 pièces. Les autres possibilités sont :

- (0; 2; 2; 0), soit 4 pièces,
- (0; 7; 0; 0), soit 7 pièces,
- (14; 0; 0; 0), soit 14 pièces.

L'algorithme glouton a ici donné la solution optimale. Mais cela ne serait pas forcément le cas pour d'autres systèmes.

Imaginons en effet un système composé uniquement de pièces de 1, 7 et 9 €, c'est-à-dire que $S = \{1; 7; 9\}$.

L'algorithme glouton nous amène à la solution (5; 0; 1), soit un rendu de 6 pièces. Alors que la solution optimale est (0; 2; 0), soit seulement 2 pièces.

Conclusion : l'algorithme glouton a le mérite d'être simple, mais il ne rend pas toujours la solution optimale. Il ne convient donc pas à la résolution de notre problème.

2.3 Approche récursive « diviser pour régner »

Notons à présent $S = \{v_1; v_2; \ldots; v_n\}$, v_i étant les valeurs des n pièces ou billets composant notre système monétaire S, et V la somme à rendre.

Nous allons calculer le nombre minimal de pièces à rendre que nous noterons `minimum`. Il servira ensuite à déterminer le n-uplet optimal recherché. La méthode précédente suggère de prendre une approche récursive.

- Si $V = 0$, on retourne « 0 »... Logique !
- Sinon, on fixe la valeur de la variable `minimum` (qui représentera au final le plus petit nombre de pièces) délibérément à une valeur assez grande (de toute évidence, le nombre de pièces est inférieur à la somme totale, donc fixer `minimum` à `V+1` nous assure que les valeurs de `minimum` qui seront calculées par la suite seront inférieures).
- On parcourt le système monétaire `S` et pour chaque valeur `S[i]`, on teste si elle est inférieure ou égale à `V` ; si tel est le cas, on prend en compte la valeur `S[i]` et on ajoute le nombre minimum de pièces ou billets nécessaires pour atteindre la valeur `V - S[i]`.

Si la valeur obtenue est plus petite que `minimum` alors cela signifie que l'on a trouvé une combinaison plus optimale que les précédentes et on change la valeur de la variable `minimum` pour la valeur de `n`.

- Une fois que l'on a parcouru le système monétaire, on retourne la valeur de `minimum`, qui correspond au plus petit nombre possible de pièces et de billets.

Le script ci-dessous est de type « diviser pour régner », puisque nous avons ramené le calcul du nombre recherché pour le montant V au même calcul pour des montants inférieurs (V − piece1, V − piece2, etc.). Nous avons divisé le problème initial, puis appliqué un traitement récursif. Notons cependant que pour cela, de nombreux calculs redondants ont été effectués. Cet algorithme n'est donc pas optimisé.

```
def Nombre(V, S = [1,7,9]):
    if V == 0:
        return 0
    else:
        minimum = V + 1
        for piece in S:
            if piece <= V:
                n = 1 + Nombre(V - piece)
                if n < minimum:
                    minimum = n
        return minimum

print(Nombre(14))
```

2.4 Approche dynamique Top Down

Nous allons améliorer l'algorithme précédent en mémorisant dans une liste nommée `L` les calculs effectués à chaque étape. `L` est une liste unidimensionnelle de $(V+1)$ éléments qui va contenir le nombre de pièces minimal `L[v]` que l'on doit utiliser pour rendre la monnaie, pour chaque montant `v` compris dans $[0;\ V]$. Cette liste est construite de manière récursive à partir du montant initial V. De cette manière, les redondances de l'algorithme précédent disparaissent et l'algorithme est optimisé.

C'est ce qu'on appelle l'approche dynamique *Top Down* (de haut en bas).

Définition 5 : Top Down

La forme récursive *Top Down* est une technique d'optimisation ayant pour but de diminuer le temps d'exécution d'une fonction récursive en mémorisant (dans une liste par exemple) les valeurs retournées par cette fonction après avoir vérifié qu'elles n'existaient pas encore dans la liste.

COURS

Cette technique est aussi appelée *mémoïsation*.

```
def construct_liste(V):
    L = [0]*(V+1)
    return nombre_top_down(V, L)

def nombre_top_down(value, L, S = [1,7,9]):
    if V == 0:
        return 0
    elif L[V] > 0:
        return L[V]
    else:
        minimum = V + 1
        for piece in S:
            if piece <= V:
                n = 1 + nombre_top_down(V - piece, L)
                if n < minimum:
                    minimum = n
                    L[V] = minimum
        return minimum

print(ConstructListe(14))
```

2.5 Approche dynamique Bottom Up

Dans cette approche, nous conservons notre liste `L` qui va être, cette fois, construite de manière itérative, en partant de 0 pour aller jusqu'à V. Il n'y a plus d'appel récursif comme c'était le cas pour la fonction `NombreTopDown`.

Cette technique de programmation dynamique est appelée *Bottom Up* (de bas en haut).

Définition 6 : Bottom Up

La forme itérative *Bottom Up* est une technique consistant à résoudre des problèmes en partant de celui qui a la taille la plus petite et en allant vers celui qui a la taille la plus grande, tout en stockant les résultats de chacun d'eux dans une liste (par exemple).

Les techniques d'optimisation « Bottom Up » et « Top Down » sont à la base de la programmation dynamique, ce qui nous amène à la définition page ci-contre.

Définition 7 : programmation dynamique

La *programmation dynamique* est une méthode de résolution de problèmes d'optimisation en scindant le problème en plusieurs sous-problèmes et en utilisant l'une des méthodes « Top Down » ou « Bottom Up ».

Ainsi, la programmation dynamique est un paradigme de programmation complémentaire à « diviser pour régner », ce dernier se révélant inefficace dans certains cas.

Voici le fonctionnement du programme final.

- On commence par initialiser deux listes `L` et `valeurs`, toutes les deux étant de dimension `V+1`, `V` étant la valeur à décomposer, les indices variant de 0 à `V`.
  `L` est une liste où `L[x]` contiendra le nombre minimum de pièces et billets qui compose la valeur `x` (comprise entre 0 et `V`) tandis que `valeurs[x]` contiendra l'indice `i` pour lequel `S[i]` $\leqslant$ `x` et pour lequel `1+L[x-S[i]]` est plus petit ou égal au nombre minimum de pièces et billets nécessaires pour la décomposition de la somme souhaitée.
- On remplit donc `L` avec la boucle : `for x in range(1,V+1)`.
- Dans cette boucle, on commence par initialiser le minimum à `V+1` ; nous sommes ainsi assuré.e.s que le minimum est plus petit que cette valeur.
- On parcourt ensuite tous les indices `i` de 0 à `len(S)`, et on effectue le test `if (S[i]<=x) and (1+L[x-S[i]]<minimum)` pour déterminer la valeur du nouveau minimum car si ces conditions sont remplies, le minimum devient `1+L[x-S[i]]`. Dans ce cas, on précise que l'indice à garder est celui sur lequel on est : on le stocke dans la variable `v` pour plus tard.
- Une fois à l'extérieur du test conditionnel, pour la valeur d'argent `x` sur laquelle on est, nous stockons le minimum obtenu dans `L[x]` ainsi que l'indice `v` obtenu dans `valeurs[x]`.
- À la sortie de la boucle itérative sur `x`, les listes `L` et `valeurs` sont ainsi remplies respectivement du minimum de pièces et billets qui composent chaque indice jusqu'à `V`.
  On initialise alors la variable `x` à `V` et une liste vide `R`, qui contiendra le rendu de monnaie, donc la décomposition.
- Pour remplir la liste `R`, il suffit de soustraire à `x` la valeur que l'on y insère, et cette valeur correspond à `S[valeurs[x]]`.
- On finit par retourner `L[V]`, à savoir le nombre minimum de pièces et billets nécessaires pour décomposer `V`, ainsi que la liste `R` nouvellement obtenue.

Le programme qui suit affiche `(2, [7, 7])`, qui représente bien la solution à notre problème :

- la solution optimale est composée de 2 pièces ;
- l'énumération des pièces à utiliser pour un montant de 14 € est : (7 €, 7 €).

COURS

Solution de l'exercice type 2

Lycée André Theuriet, Civray

```
def NombreBottomUp(V):
    S = [1,7,9]
    L = [0]*(V+1)
    valeurs = [0]*(V+1)
    for x in range(1,V+1):
        minimum = V + 1
        for i in range(len(S)):
            if (S[i] <= x) and (1+L[x-S[i]] < minimum):
                minimum = 1 + L[ x - S[i] ]
                v = i
        L[x] = minimum
        valeurs[x] = v
    x, R = V, [ ]
    while x > 0:
        R.append(S[ valeurs[x] ])
        x -= S[ valeurs[x] ]

    return L[V], R

print(NombreBottomUp(14))
```

Voir énoncé page 89

1 QCM Vérification de connaissances

10 min Corrigé p. 112

Pour chacune des questions suivantes, plusieurs propositions de réponses sont faites dont une seule est correcte. Laquelle ?

1 On définit la classe `identite` de la manière suivante :

```
class identite:
    def __init__(self, nom, prenom, an):
        self.nom = nom
        self.prenom = prenom
        self.an = an

    def age(self, a):
       return a - self.an
```

Si on tape :

```
>>> hugo = identite('Hugo' , 'Dupont' , 1999)
>>> print( hugo.age(2020) )
```

que s'affiche-t-il ?

[a] 'Hugo' [b] 'Dupont' [c] 1999 [d] 21

2 On définit la classe `voiture` de la manière suivante :

```
class voiture:
    def __init__(self, nom, couleur, puissance):
        self.nom = nom
        self.couleur = couleur
        self.puissance = puissance

    def nc(self):
        return nom + couleur
```

Si on tape :

```
>>> F430 = voiture('Ferrari' , 'Rouge' , 43)
>>> print(F430.nc())
```

que s'affiche-t-il ?

[a] 'Ferrari Rouge' [b] Une erreur

3 Qu'affiche le programme suivant ?

```
class toto:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def is_square(self):
        if self.a**2 + self.b**2 == self.c**2:
            return True
        else:
            return False

t = toto(3,4,5)

print(t.is_square())
```

[a] Une erreur [b] True [c] False

2 V/F Vérification des connaissances

10 min Corrigé p. 112

Les affirmations suivantes sont-elles vraies ou fausses ? Justifier la réponse.

1 *La programmation dynamique.*

(a) La mémoïsation est une forme itérative.

(b) La programmation dynamique consiste à résoudre des problèmes, de celui qui a la taille la plus petite vers celui qui a la taille la plus grande.

(c) Un problème d'optimisation où la variable peut être réelle peut être résolu en utilisant la programmation dynamique.

(d) Le problème du rendu de monnaie est un problème classique qui ne peut être résolu que par programmation dynamique.

2 *La Programmation Orientée Objet (POO).*

(a) Les variables d'une classe sont appelées des *méthodes*.

(b) Dans une classe, la définition d'une méthode `__init__` n'est pas obligatoire.

Classes

3 La classe « Piece »

★ 5 min Corrigé p. 113

Lycée Camille Jullian, Bordeaux

Voici un programme en Python :

```
import random

class Piece :
    def alea(self) :
        return random.randint(0,1)

    def moyenne(self, n):
        tirage = [ ]
        for i in range (n) :
            tirage.append(self.alea())

        return sum(tirage) / n

p = Piece()
print(p.moyenne(100))
```

Expliquez en détail ce qu'il permet d'afficher.

4 Écrire une classe

★ 10 min Corrigé p. 113

Lycée Sophie Germain, Paris

Pour chacune des questions suivantes, un constructeur initialisant les mesures devra être présent dans chacune des classes.

1. Écrire une classe nommée `Carre` admettant la mesure des côtés d'un carré en attribut, avec deux méthodes :
   - une méthode `perimetre` permettant de retourner le périmètre du carré ;
   - une méthode `aire` permettant de retourner son aire.
2. Écrire une classe `Triangle` représentant un triangle et admettant la mesure des trois côtés comme attributs, avec deux méthodes :
   - une méthode `aire` renvoyant l'aire du triangle à l'aide de la formule de Héron :
   $$\mathcal{A} = \sqrt{p(p-a)(p-b)(p-c)},$$
   où a, b et c sont les longueurs des trois côtés et où $p = \dfrac{a+b+c}{2}$;
   - une méthode `rect` qui renvoie `True` si le triangle est rectangle, et `False` sinon.

3 Définir alors un carré de côté 5, puis afficher son aire et son périmètre. Définir ensuite un triangle de côtés 5.4, 9 et 7.2 pour afficher son aire et regarder s'il est rectangle.

5 Devinette et remplissage ★ 10 min Corrigé p. 115

Lycée François Villon, Les Mureaux

On considère la classe `mystere` définie de la manière suivante :

```
class mystere:
    def __init__(self):
        self.Liste = [ ]

    def add(self, x):
        if len(self.Liste) == 0 or x >= self.Liste[ len
    (self.Liste)-1 ]:
            self.Liste.append(x)
        elif x < self.Liste[0]:
            self.Liste = [x] + self.Liste
        else:
            for i in range(len(self.Liste)):
                if x > self.Liste[i] and x <= self.
    Liste[i+1]:
                    self.Liste = self.Liste[:i+1] + [x]
      + self.Liste[i+1:]
                    break
```

1 Que fait le programme suivant ?

```
L = mystere()

for i in [78,89,10,50,7]:
    L.add(i)

print(L.Liste)
```

2 Ajoutez à cette classe une méthode `ecc` qui crée et renvoie une liste dont les éléments sont les cumuls de ceux de l'attribut `Liste` de la classe.

Par exemple, si `Liste = [1,2,4,5]`, `ecc` doit retourner :

`[1, 3 ,7, 12]`.

6 Une classe « Livre »

★ 10 min Corrigé p. 116

Lycée François Mauriac, Bordeaux

On définit une classe `Livre` de la manière suivante :

```
class Livre:
    def __init__(self, titre, auteur):
        self.titre = titre
        self.auteur = auteur

    def afficher_infos(self):
        print(f"Titre : {self.titre}, Auteur : {self.
    auteur}")

    def meme_auteur(self, autre_livre):
        return self.auteur == autre_livre.auteur

# Exemple d'utilisation
livre1 = Livre("1984", "George Orwell")
livre2 = Livre("La Ferme des animaux", "George Orwell")
livre3 = Livre("Le théorème du Perroquet", "Denis Guedj
    ")

print(livre1.meme_auteur(livre3))
```

1. Dans le programme ci-dessus,
   - **(a)** Combien a-t-on défini d'instances ? Quelles sont-elles ?
   - **(b)** Combien la classe `Livre` a-t-elle d'attributs ? Citez-les.
   - **(c)** Combien la classe `Livre` a-t-elle de méthodes ? Citez-les en précisant leur rôle.
2. Qu'affiche le programme ?

7 Compte bancaire

★ 20 min Corrigé p. 117

Lycée Richelieu, Versailles

On souhaite implémenter une classe `Compte_bancaire` telle que les instances de `Compte_bancaire` possèdent un seul attribut `solde` (pouvant être positif ou négatif) ainsi que les méthodes suivantes :

- une méthode `transferer` pour transférer un montant X positif vers une instance B de `Compte_bancaire` ;
- une méthode `prelever` pour prélever un montant X positif depuis une instance B de `Compte_bancaire` ;
- une méthode `etat_solde` pour renvoyer « Positif », « Négatif » ou « Nul » lorsqu'on lui demande l'état du compte.

Dans ce qui suit, on considère que le code sera écrit et indenté à l'intérieur d'un bloc commençant par `class Compte_bancaire`.

1. Écrire la méthode `__init__` en précisant si nécessaire le ou les arguments à utiliser.
2. Écrire la méthode `etat_solde` en précisant si nécessaire le ou les arguments à utiliser.
3. Soit A une variable recevant une instance de `Compte_bancaire` : `A = Compte_bancaire(1000)`.

   Prévoir le résultat de l'instruction suivante : `A.etat_solde()`.
4. **(a)** Écrire la méthode `transferer`.

   (b) Écrire la méthode `prelever`.

   (c) Écrire les instructions permettant de modéliser toutes les étapes de la situation suivante :

   - un compte bancaire A contient 1 000 € ;
   - un autre compte bancaire B contient 5 000 € ;
   - le compte bancaire B prélève 1 100 € le compte A ;
   - le solde du compte A est affiché (avec un `print`).

8 La classe « Temps »

★★ — 25 min — Corrigé p. 118

Lycée Sainte-Clotilde, Strasbourg

En Python, écrire une classe `Temps` qui permet de définir un horaire au format hh : mm : ss et qui admet les méthodes suivantes :

- `affiche`, qui affiche l'horaire au format « 12 h 37 min 45 s » ;
- `add`, qui ajoute deux horaires de la classe `Temps` ;
- `sub`, qui calcule la différence entre deux horaires de la classe `Temps`.

9 Classes « Mot » et « Phrase »

★★ — 30 min — Corrigé p. 118

Lycée Raoul Follereau, Nevers

On donne page ci-contre la classe `Mot`.

1. Indiquer ce que fait la fonction `fctA()` définie par :

```
def fctA():
  m = Mot("Socrate")
  m.doReverse()
  print(m.value())
```

```
from random import shuffle

class Mot:
  def __init__(self,m):
    self.m = m

  def do_reverse(self):
        # renverse la chaîne 'm'
    self.m = self.m[::-1]

  def do_shuffle(self):
    L = list(self.m)
    shuffle(L)
    self.m = ''.join(L)

  def value(self):
    return self.m
```

2 Indiquer ce que fait la fonction `fctB()` définie par :

```
def fctB():
  m = Mot("Socrate")
  m.doShuffle()
  print(m.value())
```

On souhaite écrire une classe `Phrase`. Toutes les questions suivantes porteront sur cette classe.

3 Écrire un constructeur qui définit une liste `self.mots` remplie de tous les mots de la phrase passée en paramètre, chacun des mots devant être de classe `Mot`.

4 Écrire deux méthodes `doReverse` et `value` telles que la fonction `fctC()` suivante :

```
def fctC():
  p = Phrase("Tous les hommes sont mortels")
  p.doReverse()
  print(p.value())
```

affiche :

« mortels sont hommes les Tous »

INTERROS

5 Écrire une méthode `doShuffle` afin que la fonction suivante :

```
def fctD():
  p = Phrase("Tous les hommes sont mortels")
  p.doShuffle()
  print(p.value())
```

affiche les mots de la phrase « Tous les hommes sont mortels » dans un ordre aléatoire.

6 On définit la méthode `motAt` de la manière suivante :

```
def motAt(self,pos):
  return self.mots[pos]
```

Que fait la fonction `fctE()` suivante ?

```
def fctE():
  p = Phrase("Tous les hommes sont mortels")
  m = p.motAt(3)
  m.doReverse()
  print(p.value())
```

7 On définit la méthode `insert` de la manière suivante :

```
def insert(self,pos,chaine):
  self.mots.insert(pos,Mot(chaine))
```

Que fait la fonction `fctF()` suivante ?

```
def fctF():
  p = Phrase("Tous les hommes sont mortels")
  p.insert(3,"ne")
  p.insert(5,"pas")
  print(p.value())
```

8 On définit la méthode `remove` de la manière suivante :

```
def remove(self,pos):
    self.mots.pop(pos)
```

Que fait la fonction `fctG()` suivante ?

```
def fctG():
  p = Phrase("Tous les hommes sont mortels")
  m = p.motAt(4)
  m.doShuffle()
  p.remove(2)
  p.insert(2,m.value())
  print(p.value())
```

10 Personnage d'un jeu vidéo

★★★ 30 min Corrigé p. 121

Lycée André Theuriet, Civray

Vous êtes en train de développer un jeu vidéo et vous avez besoin de créer une classe pour représenter les personnages du jeu. Chaque personnage a un nom, un niveau, et des points de vie. Vous devez pouvoir créer des personnages, les faire gagner de l'expérience, et vérifier leur état de santé.

Instructions :

1. Créez une classe `Personnage` avec les attributs suivants :
   - `nom` : une chaîne de caractères représentant le nom du personnage.
   - `niveau` : un entier représentant le niveau du personnage.
   - `points_de_vie` : un entier représentant les points de vie du personnage.

   Ajoutez un constructeur `__init__` pour initialiser ces attributs.
2. Implémentez les méthodes suivantes.
   - **(a)** `gagner_experience(self, experience)` : une méthode qui prend un nombre d'expérience en paramètre et augmente le niveau du personnage en fonction de l'expérience gagnée. Supposons que chaque tranche de 100 points d'expérience augmente le niveau de 1.
   - **(b)** `est_vivant(self)` : une méthode qui retourne `True` si le personnage a des points de vie supérieurs à 0, sinon `False`.
   - **(c)** `afficher_etat(self)` : une méthode qui affiche les informations du personnage sous la forme « Nom : [nom], Niveau : [niveau], Points de vie : [points_de_vie] ».

INTERROS

Programmation dynamique

11 Découpe d'une chaîne

★ 15 min Corrigé p. 121

Lycée Sainte-Anne, Brest

Complétez le programme suivant :

```
def decouper_chaine(chaine, dictionnaire, memo={}):
    if chaine in memo:
        return ...

    if chaine == "":
        return ...

    resultats = []
    for i in range(1, len(chaine) + 1):
        prefixe = chaine[:i]
        if prefixe in dictionnaire:
            for suffixe_decoupe in decouper_chaine(
    chaine[i:], dictionnaire, memo):
                resultats.append([prefixe] +
    suffixe_decoupe)

    memo[chaine] = resultats
    return resultats
```

Le but de ce programme est de proposer une liste de plusieurs « découpes » d'une chaîne de caractère selon un dictionnaire de mots fourni.

```
>>> dictionnaire = {
"chat", "chapeau", "au", "lait", "laitue", "eau", "tue", "lai"
}
>>> chaine = "chapeaulaitueau"
>>> decoupes = decouper_chaine(chaine, dictionnaire)
>>> print("Découpes possibles :", decoupes)
Découpes possibles :
[['chapeau', 'lai', 'tue', 'au'],
['chapeau', 'laitue', 'au']]
```

---

12 Modélisation d'une suite numérique ★★

20 min Corrigé p. 122

Lycée François Mauriac, Bordeaux

On considère la suite numérique (u_n) définie par ses deux premiers termes $u_0 = 1$, $u_1 = 2$ et par la relation de récurrence suivante :

$$\forall\, n \in \mathbb{N}, \quad u_{n+2} = \frac{u_n}{u_{n+1}}.$$

Écrire un programme Python permettant de calculer u_n pour un entier n donné en utilisant la programmation dynamique (version Bottom Up et Top Down).

13 Somme dans une suite numérique ★★★

30 min Corrigé p. 123

Lycée polyvalent Émile Zola, Aix-en-Provence

On définit une suite (u_n) par son premier terme $u_0 = 1$ et par la relation de récurrence :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \frac{1}{n+1}\sum_{k=0}^{n}(k+1)u_k.$$

1. Calculer u_1, u_2 et u_3.
2. Proposer deux programmes permettant de calculer u_n pour tout entier naturel n en utilisant le paradigme de la programmation dynamique (version Bottom Up et Top Down).

14 Le retour du problème du sac à dos ★★★

30 min Corrigé p. 124

Lycée Jean-Joseph Fourier, Auxerre

Nous avons vu dans l'exercice 11 page 75 ce en quoi consiste le problème du sac à dos. Nous en avons d'ailleurs donné une solution récursive de type « diviser pour régner ».

1. Proposer une approche dynamique de type Bottom Up et Top Down de ce problème qui affiche uniquement la valeur maximale des objets.

   Vérifier les programmes avec l'exemple où le poids maximal est $P = 10$ et où la collection d'objets est $E = \{(5,7); (3,1); (3,4); (3,2)\}$.
2. Compléter l'approche Bottom Up (par exemple) afin d'afficher la combinaison des objets retenus.
3. Que dire de la complexité de l'approche Bottom Up par rapport à celle de l'approche récursive initiale ?

INTERROS

15 La civilisation Maya

★★★ 45 min Corrigé p. 127

Extrait du Bac 2024, Métropole, septembre

Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.

La civilisation Maya est une ancienne civilisation de Mésoamérique principalement connue pour ses avancées dans les domaines de l'écriture, de l'art, de l'architecture, de l'agriculture, des mathématiques et de l'astronomie.

La numération Maya est une numération positionnelle de base 20 (dite vigésimale) utilisant trois symboles pour former les « chiffres » :

- une coquille pour le zéro,
- un point pour l'unité,
- un trait pour la valeur 5.

Les « chiffres » sont les suivants. Ils utilisent une numération additive :

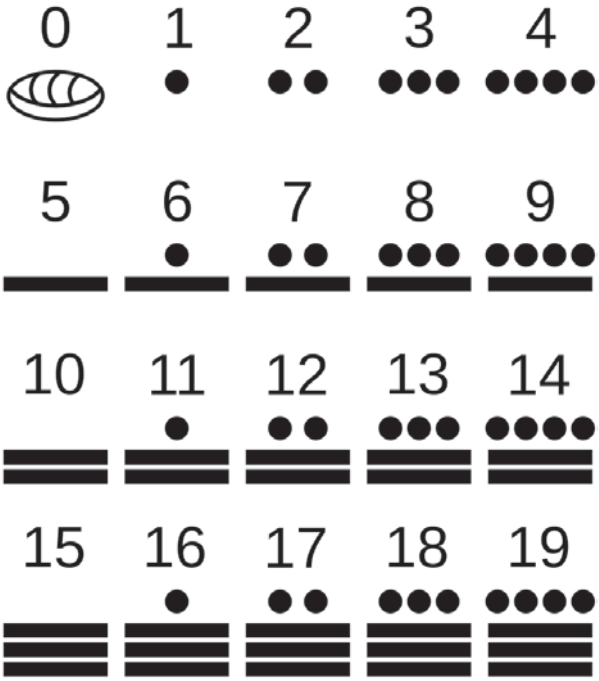
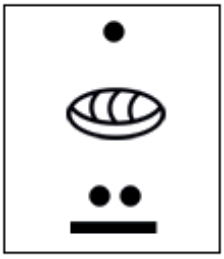


Figure 4.1 – Table des « chiffres » et valeurs correspondantes (source : Wikipédia)

Dans une version simplifiée de ce système, l'écriture d'un nombre se fait par empilement de « chiffres ». Chaque étage correspond à un chiffre de poids 20 fois supérieur au poids du chiffre de l'étage inférieur.

Ainsi la valeur du chiffre de l'étage le plus bas est multipliée par 20^0 soit 1, du second étage par 20^1, du troisième étage par 20^2, et ainsi de suite.

Exemple : la représentation Maya de l'entier 407 est la suivante.



1 Compléter le tableau donné en annexe à rendre avec la copie.

2 Justifier que l'écriture Maya de l'entier 3 435 est :

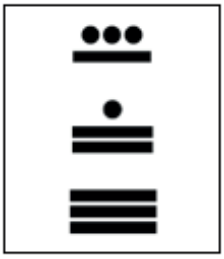


Figure 4.2 – Écriture Maya de l'entier 3 435

On modélise l'écriture d'un entier dans sa représentation Maya par une pile formée de listes. Chacune de ces listes est composée de trois entiers et modélise le « chiffre » d'un étage :

- le premier entier vaut 0 ou 1 suivant s'il y a ou non une coquille ;
- le deuxième représente le nombre de points ;
- le troisième représente le nombre de traits.

Ainsi, la modélisation Maya de l'entier 3 435 est :

```
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
```

et celle de l'entier 407 est :

```
[[0, 2, 1], [1, 0, 0], [0, 1, 0]].
```

Le sommet de la pile se situe en fin de liste. On dispose de la classe suivante :

```
class Maya:
    def __init__(self):
        self.nombre = []
    def ajouter(self, chiffre):
        """ chiffre est une liste de longueur 3.
        La méthode empile le chiffre au sommet de la
    pile """
        self.nombre.append(chiffre)
    def retirer(self):
        """ depile et renvoie le chiffre qui etait au
    sommet de
            la pile """
        if not self.estVide():
            return self.nombre.pop()
    def estVide(self):
        return self.nombre == []
    def nbEtages(self):
        """ renvoie le nombre de chiffres de la pile ""
    "
        ...
    def MayaToDec(self):
        """ renvoie le nombre entier correspondant a la
        modelisation Maya de l'instance courante """
        coeff = 20**...
        ch_Dec = 0
        while ... :
            ch_Maya = ...
            ch_Dec = ch_Dec + (valeurChiffre(ch_Maya))
    * coeff
            coeff = ...
        return ch_Dec
    def multiplie(self):
        """ renvoie le resultat de la multiplication
    par 20 d'un
        nombre en modelisation Maya. """
        ...
    def somme(self, maya2):
        """ ajoute maya2 à l'instance courante et
    renvoie le
        resultat en modelisation Maya """
        if self.nbEtages() == maya2.nbEtages()
            ...
```

3 Écrire une suite d'instructions permettant de créer une instance, nommée M, de la classe Maya qui modélise le nombre entier 3 435.

4 Écrire la méthode `nbEtages` de la classe `Maya`. Celle-ci renvoie le nombre de « chiffres » utilisés pour écrire le nombre correspondant en écriture Maya.

De l'écriture Maya à l'écriture décimale

5 Écrire une fonction `valeurChiffre` ayant pour paramètre une liste `L`. Celle-ci renvoie la valeur de l'entier associé à la liste `L = [c,p,t]` où `c` (de valeur 0 ou 1) indique la présence d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant un « chiffre » Maya.

Exemple :

```
>>> valeurChiffre([0, 2, 3])
>>> 17
>>> valeurChiffre([1, 0, 0])
>>> 0
```

6 Recopier et compléter les lignes 2, 4, 5 et 7 de la méthode `MayaToDec` suivante de la classe `Maya`. Cette méthode renvoie la valeur de l'entier associé à l'objet `Maya`. On pourra utiliser les méthodes `estVide`, `nbEtages` et `retirer`.

```
1  def MayaToDec(self):
2      coeff = 20**(self.nbEtages() - 1)
3      ch_Dec = 0
4      while not self.estVide() :
5          ch_Maya = self.retirer()
6          ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)
       ) * coeff
7          coeff = coeff // 20
8      return ch_Dec
```

De l'écriture décimale vers sa modélisation Maya

On considère que la fonction `DecToVige` est déjà écrite. Celle-ci prend en paramètre un entier n et renvoie la décomposition en base 20 de celui-ci sous la forme d'une liste $[a_0, a_1, \ldots, a_p]$ telle que :

$$n = a_0 \times 20^0 + a_1 \times 20^1 + a_2 \times 20^2 + \cdots + a_p \times 20^p.$$

Exemple :
```
>>> DecToVige(3435)
[15, 11, 8]
>>> DecToVige(407)
[7, 0, 1]
```

7 Écrire la fonction `decompChiffre` qui prend en paramètre un entier n compris entre 0 et 19 et renvoie la liste `[c, p, t]` où `c` vaut 0 ou 1 et indique la présence ou non d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant le « chiffre » Maya correspondant.

Exemple :
```
>>> decompChiffre(17)
[0, 2, 3]
>>> decompChiffre(0)
[1, 0, 0]
```

8 Écrire la fonction `DecToMaya` qui prend en paramètre un entier n et renvoie la modélisationMaya d'un objet `M` de la classe `Maya` correspondant.

Exemple :
```
>>> DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
>>> DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

Opérations sur les nombres en modélisation Maya

On souhaite additionner des nombres directement à partir de leur modélisation Maya.

9 Écrire la méthode `multiplie` de la classe `Maya` qui renvoie le résultat de la multiplication par 20 d'un nombre en modélisation Maya.

Exemple :
```
>>> M = Maya()
>>> M.ajouter([0, 0, 3])
>>> M.ajouter([0, 1, 2])
>>> M.multiplie().nombre
[[1, 0, 0], [0, 0, 3], [0, 1, 2]]
```

On donne la fonction `mystere` suivante :

```
def mystere(m1, m2, ret):
    c = 0
    p = (m1[1] + m2[1] + ret) % 5
    if m1[1] + m2[1] + ret >= 5:
        ret = 1
    else:
        ret = 0
    t = (m1[2] + m2[2] + ret) % 4
    if m1[2] + m2[2] + ret < 4:
        ret = 0
    else:
        ret = 1
    if (m1[0] == 1 and m2[0] == 1) or (p + t == 0 and
    ret == 1):
        c = 1
    return ([c, p, t], ret)
```

10 Donner les résultats renvoyés par les deux appels suivants :

- `mystere([0, 1, 1], [0, 3, 1], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)`

11 Écrire une méthode `somme` de la classe `Maya` permettant d'ajouter à l'instance courante un autre nombre `maya2` de même taille en modélisation Maya.

ANNEXE À RENDRE AVEC LA COPIE

Étage	Écriture Maya	Valeur du « chiffre » de l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3\,200$
2			
1			

CORRIGÉS

1 QCM Vérification de connaissances

Énoncé p. 95

1 Réponse **d**. En effet, la méthode calcule la différence entre la valeur de l'argument `a` et la valeur de `self.an`, qui correspond à l'année (sans doute de naissance) de Hugo. Il s'agit donc ici de $2020 - 1999 = 21$.

2 Réponse **b**. En effet, dans la déclaration de la méthode `nc`, les variables `nom` et `couleur` ne sont pas définies. Il aurait fallu écrire `self.nom + self.couleur` pour que cela soit correct.

Rappelons que la déclaration d'une méthode peut faire appel :

- à des variables locales (qui doivent être initialisées avant utilisation et qui sont « détruites » à la fin de la méthode) ;
- à des attributs de l'objet lui-même qui « appartiennent » à l'objet, qui sont partagés entre toutes les méthodes, et qui subsistent en mémoire jusqu'à ce que l'objet lui-même soit détruit. Dans ce cas, il faut y faire référence en utilisant le paramètre `self`, qui est le premier argument de toute méthode.

3 Réponse **b**. Ici, tout est correctement défini. Ainsi, la méthode `is_square` regarde si $a^2 + b^2$ est bien égal à c^2, ce qui est le cas pour notre exemple. « True » est donc affiché.

2 V/F Vérification des connaissances

Énoncé p. 96

1 *La programmation dynamique.*

(a) *Faux*. La mémoïsation, ou Top Down, est une forme récursive.

(b) *Faux*. La technique décrite dans la question est celle du Bottom Up. La programmation dynamique, quant à elle, consiste à découper un problème en plusieurs sous-problèmes de tailles inférieures et en utilisant l'une des techniques Bottom Up ou Top Down pour les résoudre.

(c) *Faux*. La programmation dynamique ne sert que lorsque la variable est entière.

(d) *Faux*. Nous avons vu dans le cours qu'il existait d'autres façons que la programmation dynamique pour résoudre le problème du rendu de monnaie, comme l'approche gloutonne ou diviser pour régner. Cependant, la programmation dynamique a pour objectif d'optimiser le code. C'est donc cette technique qui est préférable aux autres car elle nécessite moins d'opérations de calcul et de traitement que les autres.

2 *La Programmation Orientée Objet (POO).*

(a) *Faux*. Une *méthode* est une *fonction* d'une classe. Les *attributs* sont les variables de la classe.

(b) *Vrai*. On peut tout à fait définir une classe sans la méthode `__init__`. En voici un exemple :

```
class toto:
    b = 5
    def fct(self,a): return a * self.b
d = toto()
print(d.fct(10))
```

qui renvoie « 50 ».

Le constructeur `__init__` est rendu obligatoire dès lors qu'un paramètre est passé lors de l'appel de la classe (par exemple, `d = toto(7)`) ou qu'il est nécessaire d'initialiser certains attributs dès la création de l'objet.

3 La classe « Piece »

Énoncé p. 97

Lycée Camille Jullian, Bordeaux

La classe `Piece` est composée de deux méthodes.

- La méthode `alea` retourne simplement un nombre aléatoire de l'ensemble {0; 1}.
- La méthode `moyenne`, qui admet un attribut `n`, commence par définir une liste vide nommée `tirage`. Ensuite, une boucle itérative est créée, pour 100 itérations. Au cours de chacune d'elles, on insère dans la liste `tirage` le retour de la méthode `alea` (on insère donc « 0 » ou « 1 » n fois).
  Une fois la boucle finie, on retourne la valeur de « `sum(tirage) / n` », c'est-à-dire la somme des nombres contenus dans la liste `tirage` divisée par n : il s'agit ici du calcul du nombre moyen de « 1 » obtenus lors des n choix aléatoires.

Ainsi, dans ce contexte, si on décide d'associer « Pile » à « 1 » lors du lancer d'une pièce, le programme calcule le nombre moyen de « Pile » sur 100 lancers.

4 Écrire une classe

Énoncé p. 97

Lycée Sophie Germain, Paris

1

```
class Carre:
    def __init__(self,a):
        self.cote = a
    def perimetre(self):
        return self.cote*4
    def aire(self):
        return self.cote**2
```

2 Une classe possible est la suivante :

```
class Triangle:
    def __init__(self,a,b,c):
        self.a, self.b, self.c = a, b, c

    def aire(self):
        p = (self.a + self.b + self.c)/2
        return (p*(p-self.a)*(p-self.b)*(p-self.c))
    **0.5

    def rect(self):
        L = [self.a, self.b, self.c]
        L.sort()

        return L[2]**2 == L[0]**2 + L[1]**2
```

3 Définissons dans une console le carré et le triangle demandés.

```
>>> C = Carre(5)
>>> C.aire() , C.perimetre()
(25, 20)
>>> T = Triangle(5.4 , 9 , 7.2)
>>> (T.aire() , T.rect())
(19.44000000000001, True)
```

On voit ainsi que le triangle défini est rectangle et que son aire vaut 19,44.

Remarque : on pourrait améliorer légèrement l'affichage dans le cas où le triangle est rectangle.

En effet, dans ce cas précis, pour calculer l'aire du triangle, on peut utiliser la formule $\frac{b \times h}{2}$, où b et h sont respectivement une base du triangle et la hauteur relative à cette base.
Ainsi, dans la méthode `aire`, on pourrait écrire le code page ci-contre.

À RETENIR

Quel que soit le contexte d'un programme, il ne faut pas s'éloigner de ses règles. Si le programme concerne des mathématiques alors il faut utiliser les propriétés mathématiques pour optimiser le code.

```
class Triangle:
    def __init__(self,a,b,c):
        self.cotes = [a, b, c]
        self.cotes.sort()

    def rect(self):
        return self.cotes[2]**2 == self.cotes[0]**2
     + self.cotes[1]**2

    def aire(self):
        if self.rect():
            return self.cotes[0] * self.cotes[1] /
    2
        else:
            p = (self.cotes[0] + self.cotes[1] +
    self.cotes[2])/2
            return (p*(p-self.cotes[0])*(p-self.
    cotes[1])*(p-self.cotes[2]))**0.5
```

5 Devinette et remplissage

Énoncé p. 98

Lycée François Villon, Les Mureaux

1 `L.Liste` est *a priori* une liste (d'après la ligne : `self.Liste = [ ]`). Ainsi, l'affichage sera une liste. Regardons maintenant de plus près la définition de la fonction `add` :

- avant tout, si l'argument `x` est supérieur ou égal au dernier élément de la liste ou si la liste est vide, on ajoute la valeur de `x` à la liste ;
- sinon, si `x` contient une valeur strictement plus petite que le premier élément de la liste, on place la valeur de `x` en tête de la liste ;
- dans les autres cas, on parcourt la liste et on teste si la valeur contenue dans `x` est comprise entre deux valeurs de la liste. Si tel est le cas, on insère cette valeur entre les deux autres valeurs.

On peut alors dire que la fonction `add` ajoute la valeur de l'argument tout en faisant en sorte que la liste soit rangée dans l'ordre croissant.

Ainsi, le programme proposé affichera :

```
[7, 10, 50, 78, 89]
```

2 Voici une proposition pour la méthode `ecc` :

```
def ecc(self):
  E, prev = [], 0
  for nb in self.Liste:
    prev = prev + nb
    E.append(prev)
  return E
```

Ici, on commence par créer une liste `E` dont le seul élément est le premier de la liste `self.Liste`, puis on parcourt `self.Liste` à partir du deuxième élément : pour chaque nombre d'index `i`, on lui ajoute `E[i-1]` et on stocke le résultat à la fin de la liste `E`, donc dans `E[i]`.

La boucle terminée, on retourne `E`, qui est la liste de tous les cumuls.

Remarque : en statistiques, on appelle ces nombres les *effectifs cumulés croissants*.

6 Une classe « Livre »

Énoncé p. 98

Lycée François Mauriac, Bordeaux

1 **(a)** Dans le programme, il y a trois instances :

```
livre1 = Livre("1984", "George Orwell")
livre2 = Livre("La Ferme des animaux",
               "George Orwell")
livre3 = Livre("Le théorème du Perroquet",
               "Denis Guedj")
```

(b) La classe `Livre` a deux attributs : `titre` et `auteur`.

(c) La classe `Livre` a deux méthodes :

- `afficher_infos(self)` : une méthode qui affiche les informations du livre sous la forme « Titre : [titre], Auteur : [auteur] » ;
- `meme_auteur(self, autre_livre)` : une méthode qui prend un autre objet de type `Livre` en paramètre et retourne `True` si les deux livres ont le même auteur, sinon `False`.

2 Le programme affiche « False » car `livre1` et `livre3` n'ont pas le même auteur.

7 Compte bancaire

Énoncé p. 99

Lycée Richelieu, Versailles

1 Le constructeur de la classe peut être le suivant :

```
class Compte_bancaire:
    def __init__(self, solde_initial):
        self.solde = solde_initial
```

2 La méthode `etat_solde` peut être définie comme suit :

```
def etat_solde(self):
    if self.solde > 0:
        return "Positif"
    elif self.solde < 0:
        return "Négatif"
    else:
        return "Nul"
```

3 Le résultat est le suivant :

```
>>> A = Compte_bancaire(1000)
>>> A.etat_solde()
"Positif"
```

4 **(a)** La méthode `transférer` peut être définie ainsi :

```
def transferer(self, montant, destination):
    destination.solde += montant
    self.solde -= montant
```

(b) La méthode `prelever` peut être définie ainsi :

```
def prelever(self, montant, origine):
    origine.solde -= montant
    self.solde += montant
```

(c) La suite d'instructions peut être la suivante :

```
A = Compte_bancaire(1000)
B = Compte_bancaire(5000)
B.prelever(1100, A)
print(A.solde)
```

CORRIGÉS

8 La classe « Temps »

Énoncé p. 100

Lycée Sainte-Clotilde, Strasbourg

Une définition possible de la classe `Temps` est la suivante :

```
class Temps:
    def __init__(self,h,m,s):
        self.h, self.m, self.s = h, m, s

    def affiche(self):
        return '{} h {} min {} s'.format(str(self.h),
    str(self.m), str(self.s))

    def add(self,t):
        sec = self.s + t.s
        rs = sec % 60
        minute = self.m + t.m + sec//60
        rm = minute % 60
        rh = self.h + t.h + minute//60
        return Temps(rh,rm,rs)

    def sub(self,t):
        a = self.h*3600 + self.m*60 + self.s
        b = t.h*3600 + t.m*60 + t.s
        diff = abs(b-a)
        h = diff // 3600
        m = (diff-3600*h)//60
        s = diff-3600*h-60*m
        return Temps(h,m,s)
```

Avec cette définition, voici ce que l'on peut obtenir :

```
>>> t1, t2 = Temps(8,45,37), Temps(15,31,18)
>>> (t1+t2).affiche(), (t1-t2).affiche()
'24 h 16 min 55 s' , '6 h 45 min 41 s'
```

9 Classes « Mot » et « Phrase »

Énoncé p. 100

Lycée Raoul Follereau, Nevers

1 L'instruction « `m = Mot("Socrate")` » définit dans un premier temps un objet `Mot` comme étant la chaîne de caractères « Socrate ».

Ensuite, « `m.doReverse()` » fait appel à la méthode `doReverse` de la classe `Mot` qui, comme son nom l'indique, transforme la chaîne de caractères en inversant les lettres.

Enfin, « `print(m.value())` » affiche ce qui est contenu dans `self.m` de la classe `Mot`, donc affiche ici : « etarcoS » (le mot écrit à l'envers).

2 Comme dans la fonction précédente, on commence par définir un objet de classe « Mot », puis on fait appel à la méthode `doShuffle` qui retourne une anagramme du mot.

La fonction `fctB()` affiche donc une anagramme de « Socrate ».

3 Un constructeur possible est le suivant :

```
class Phrase:
  def __init__(self,phrase):
    L = phrase.split()
    self.mots = []
    for m in L:
      self.mots.append(Mot(m))
```

On commence par définir une liste `L` composée de tous les mots de la phrase passée en paramètre avec la méthode `split` : à ce stade, les mots se sont que des chaînes de caractères. Or, nous souhaitons que chaque mot soit de classe `Mot`. On parcourt alors cette liste et, à partir de chaque item, on définit un objet de classe `Mot` que l'on insère dans une liste `self.mots` (« self » est indispensable pour lier cette nouvelle liste à l'objet de classe `Phrase` ainsi défini).

4 Une proposition de méthodes est la suivante :

```
def doReverse(self):
  self.mots = self.mots[::-1]

def value(self):
  result  = ''
  for m in self.mots:
    result += m.value() + ' '
  return result
```

Dans la méthode `doReverse`, nous n'avons fait qu'inverser l'ordre des items de la liste `self.mots` avec l'opération `[::-1]`, et dans la méthode `value`, nous avons initialisé une chaîne de caractères `result` comme étant vide, puis parcouru chaque item de la liste `self.mots` pour concaténer à `result` la chaîne de caractères correspondant à l'item (`m.value()`) en y ajoutant un espace afin qu'au final, `result` représente la phrase.

5 Ici, nous pouvons utiliser à nouveau la fonction `shuffle` du module `random`, ce qui donne :

```
def doShuffle(self):
  shuffle(self.mots)
```

6 Dans la fonction `fctE()`, on commence par définir une phrase, puis un mot `m` comme étant l'objet `Mot` en position 3 dans la phrase définie. Ici, il s'agit donc de « sont » (attention : le premier mot est à la position 0, comme dans toute liste). Ensuite, on applique la méthode `doReverse` de la classe `Mot` au mot ainsi trouvé, ce qui a pour effet de mettre ses lettres dans le sens inverse.

On affiche ensuite la phrase ainsi obtenue, ce qui donne :

Tous les hommes tnos mortels

7 Dans la fonction `fctE()`, on commence par définir une phrase, puis on fait appel à la méthode `insert` qui insère le mot « ne » dans la phrase en position 3, c'est-à-dire après le 3e mot de la phrase. On fait de même en position 5, c'est-à-dire après le 5e mot : on y insère le mot « pas ». Et enfin, on affiche la phrase ainsi obtenue, ce qui donne :

Tous les hommes ne sont pas mortels

8 On commence par prendre le mot en position 4 dans la phrase, donc « mortels ». On mélange ensuite les lettres dans ce mot en faisant appel à la méthode `doShuffle` de la classe `Mot`. Ensuite, on fait appel à la méthode `remove` de la classe `Phrase` en passant en paramètre le nombre 2 : on supprime alors le mot « hommes ». Enfin, on insère en position 2 le contenu de la variable `m`, c'est-à-dire l'anagramme de « mortels » obtenu précédemment. On affiche alors le résultat, ce qui donne par exemple :

Tous les elmsort sont elmsort

Remarque : le dernier mot n'est plus « mortels » ; en effet, nous avons définit `m` comme l'objet en position 4 dans l'objet `p`. Dès lors que l'on écrit `m.doShuffle`, cet objet est modifié dans l'objet `p`. Ainsi, à ce stade, l'objet « mortels » n'existe plus mais il est remplacé par une anagramme de la chaîne de caractères qu'il représente.

Télécharger le programme complet à l'aide du QR code.

10 Personnage d'un jeu vidéo

Énoncé p. 103

Lycée André Theuriet, Civray

1 La classe `Personnage` peut être définie avec son constructeur de la manière suivante :

```
class Personnage:
    def __init__(self, nom, niveau, points_de_vie):
        self.nom = nom
        self.niveau = niveau
        self.points_de_vie = points_de_vie
```

2 **(a)**

```
def gagner_experience(self, experience):
        self.niveau += experience // 100
```

(b)

```
def est_vivant(self):
        return self.points_de_vie > 0
```

(c)

```
def afficher_etat(self):
        print(f"Nom : {self.nom}, Niveau : {self
    .niveau}, Points de vie : {self.
    points_de_vie}")
```

11 Découpe d'une chaîne

Énoncé p. 104

Lycée Sainte-Anne, Brest

Voici le programme dument complété :

```
def decouper_chaine(chaine, dictionnaire, memo={}):
    if chaine in memo:
        return memo[chaine]
    if chaine == "":
        return [[]]
    resultats = []
    for i in range(1, len(chaine) + 1):
        prefixe = chaine[:i]
        if prefixe in dictionnaire:
            for suffixe_decoupe in decouper_chaine(
    chaine[i:], dictionnaire, memo):
                resultats.append([prefixe] +
    suffixe_decoupe)
    memo[chaine] = resultats
    return resultats
```

CORRIGÉS

12 Modélisation d'une suite numérique

Énoncé p. 105

Lycée François Mauriac, Bordeaux

Une version Bottom Up du programme dynamique est la suivante :

Bottom Up

```
def u(n):
    U = [ 0 ] * (n+1)
    for i in range(n+1):
        if i == 0: U[i] = 1
        elif i == 1: U[i] = 2
        else:
            U[i] = U[i-2] / U[i-1]
    return U[n]
```

Une version optimisée (en espace mémoire) est la suivante :

Bottom Up (optimisée en espace mémoire)

```
def u(n):
    u,v = 1,2
    for i in range(2,n+1):
        if n == 0: return u
        elif n == 1: return v
        else:
            w = u / v
            u = v
            v = w
    return w
```

Une version Top Down est la suivante :

Top Down

```
def u(n, U = None):
    if n == 0: return 1
    elif n == 1: return 2
    else:
        U = [None] * (n+1)
        U[0], U[1] = 1, 2
        return u_cache(U,n)

def u_cache(U,n):
    if U[n] is None:
        U[n] = u(n-2, U) / u(n-1, U)
    return U[n]
```

Une manière plus simple peut aussi être envisagée :

Top Down

```
def u(n, U = None):
  if U is None or U == []:
    U = [1,2] + [None] * (n-1)
  if U[n] is None:
    U[n] = u(n-2, U) / u(n-1, U)
  return U[n]
```

13 Somme dans une suite numérique

Énoncé p. 105

Lycée polyvalent Émile Zola, Aix-en-Provence

1 Bien que mathématique, et non informatique, cette question permet de comprendre le mode de génération des termes de la suite définie.

- $u_1 = u_{0+1} = \dfrac{1}{0+1}\displaystyle\sum_{k=0}^{0}(k+1)u_k = u_0 = 1$;
- $u_2 = u_{1+1} = \dfrac{1}{1+1}\displaystyle\sum_{k=0}^{1}(k+1)u_k = \dfrac{1}{2}(1u_0 + 2u_1) = \dfrac{3}{2}$;
- $u_3 = u_{2+1}$

$$= \frac{1}{2+1}\sum_{k=0}^{2}(k+1)u_k$$
$$= \frac{1}{3}(1u_0 + 2u_1 + 3u_2)$$
$$= \frac{1}{3} \times \left(3 + \frac{9}{2}\right) = \frac{5}{2}.$$

2 La difficulté de ce genre de calcul réside dans la sommation. Il faut penser à la stocker au même titre que les termes de la suite.

Bottom Up

```
def u(n):
    U = [None] * (n+1)
    U[0] = (1,0)
    for k in range(1,n+1):
        S = U[k-1][1] + k * U[k-1][0]
        U[k] = (S/k, S)
    return U[n][0]

for n in range(20):
    print (u(n))
```

L'idée ici est donc de stocker non pas un terme mais un couple constitué du terme u_k et de la somme $u_0 + 2u_1 + \cdots + (k+1)u_k$ pour pouvoir calculer par la suite u_{k+1}.

Top Down

```
def u(n, U = None):
    if U is None or U == []:
        U = [(1,0)] + [(None,None)] * n
    if U[n][0] is None:
        val = u(n-1, U)[1] + n * u(n-1, U)[0]
        U[n] = (val / n, val)
    return U[n]

for n in range(20):
    print(u(n)[0])
```

14 Le retour du problème du sac à dos

Énoncé p. 105

Lycée Jean-Joseph Fourier, Auxerre

1 Rappelons avant tout la fonction récursive initiale du problème :

```
def sac(E,P,i):
    if i == 0:
        return 0
    if E[i-1][0] > P:
        return sac(E, P, i-1)
    else:
        return max(sac(E, P, i-1), E[i-1][1] + sac(
    E, P-E[i-1][0], i-1))

def initSac(E,P):
    return sac(E, P, len(E))
```

- *Approche dynamique de type « Top Down ».*

  Pour améliorer cette fonction récursive, il nous faut mémoriser les solutions aux sous-problèmes (mémoïsation). Nous allons donc stocker les valeurs de `V[i][p]`, avec $0 \leqslant i \leqslant n$ et $0 \leqslant p \leqslant P$, où P est le poids maximal autorisé, dans une liste nommée par exemple `temp`.
  La solution est très proche de l'initiale.

```
def sac(E,P,i,temp):
    if temp[i][P]>0: return temp[i][P]
    if i == 0: return 0
    if E[i-1][0] > P:
        temp[i][P] = sac(E, P, i-1,temp)
        return temp[i][P]
    else:
        temp[i][P] = max(sac(E, P, i-1, temp), E[i
    -1][1] + sac(E, P - E[i-1][0], i-1, temp))
        return temp[i][P]

def initSac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)
    ]
    return sac(E, P, len(E), temp)
```

Ce qui donne pour notre exemple :

```
>>> E, P = [ (5,7) , (3,1) , (3,4) , (3,2) ] , 10
>>> initSac(E,P)
11
```

Cela signifie que la valeur maximale des objets que l'on peut mettre dans le sac est égale à 11 ($7 + 4$, avec un poids de $5 + 3 = 8$).

- *Approche dynamique de type « Bottom Up ».*

```
def sac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)]
    for i in range(1,len(E)+1):
        for p in range(P+1):
            if E[i-1][0] > p:
                temp[i][p] = temp[i-1][p]
            else:
                temp[i][p] = max(temp[i-1][p], E[i
    -1][1] + temp[i-1][ p - E[i-1][0] ])
    return temp[len(E)][P]
```

Avec cette approche, il n'y a plus de récursivité mais une itération. On garde à peu près la même structure pour la fonction `sac`.

On obtient bien entendu le même résultat, à savoir « 11 ».

2 Comme nous pouvons le voir ici, ces solutions n'affichent que la valeur optimale. Le problème du sac à dos étant d'obtenir la combinaison des objets retenus, il nous faut compléter ces solutions.

Si on complète l'approche Bottom up, cela donne le programme suivant :

```
def sac(E,P):
    temp = [ [0] * (P+1) for i in range(len(E)+1)]
    garde  = [ [False] * (P+1) for i in range(len(E)
    +1) ]
    for i in range(1,len(E)+1):
        for p in range(P+1):
            if E[i-1][0] > p:
                temp[i][p] = temp[i-1][p]
            else:
                if E[i-1][1] + temp[i-1][p-E[i
    -1][0]] > temp[i-1][p]:
                    temp[i][p] =  E[i-1][1] + temp[
    i-1][ p - E[i-1][0] ]
                    garde[i][p] = True
                else:
                    temp[i][p] = temp[i-1][p]
    p,L = P,[]
    for i in range(len(E),0,-1):
        if garde[i][p]:
            L.append(i)
            p -= E[i-1][0]

    return temp[len(E)][P], L
```

Ici, on ajoute une liste `garde` qui stocke le numéro des objets que l'on met dans le sac.

3 Dans l'approche récursive initiale, la complexité est en $O\left(2^n\right)$. En effet, deux appels récursifs sont fait pour n objets.

Dans l'approche Bottom Up, elle est en $O(n \times P)$ car il y a deux boucles imbriquées portant sur n et P, et qu'il y a le même nombre d'opérations élémentaires dans chacune d'elles.

⚠ ATTENTION

Il ne faut pas conclure que la complexité est linéaire par rapport à P. En effet, la variable `P` est de taille $\log_2(P)$ et on peut alors écrire que le poids maximal est $P = 2^{\log_2(\mathrm{P})}$. La complexité du programme dynamique est alors exponentielle par rapport à la taille de `P`. Cependant, elle reste meilleure que celle du programme initial.

15 La civilisation Maya

Énoncé p. 106

Extrait du Bac 2024, Métropole, septembre

1 Le tableau complété est le suivant :

Étage	Écriture Maya	Valeur du « chiffre » de l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3\,200$
2		$2 \times 5 + 1 \times 1 = 11$	$11 \times 20^1 = 220$
1		$3 \times 5 = 15$	$15^2 0^0 = 15$

2 En additionnant les nombres trouvés précédemment, on a :

$$3\,200 + 220 + 15 = 3\,435.$$

3 Les instructions pour créer une instance de la classe `Maya` modélisant le nombre 3 435 sont :

```
M = Maya()
M.ajouter([0, 0, 3])
M.ajouter([0, 1, 2])
M.ajouter([0, 3, 1])
```

4 La méthode `nbEtages` de la classe `Maya` est :

```
def nbEtages(self):
    return len(self.nombre)
```

5 La fonction `valeurChiffre` est :

```
def valeurChiffre(L):
    return L[1] + 5 * L[2]
```

6 La méthode `MayaToDec` complétée est :

```
def MayaToDec(self):
    coeff = 20**...
    ch_Dec = 0
    while ... :
        ch_Maya = ...
        ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) *
     coeff
        coeff = ...
    return ch_Dec
```

7 La fonction `decompChiffre` est :

```
def decompChiffre(n):
    if n == 0:
        return [1, 0, 0]
    return [0, n % 5, n // 5]
```

8 La fonction `DecToMaya` est :

```
def DecToMaya(n):
    M = Maya()
    for coeff in DecToVige(n):
        M.ajouter(decompChiffre(coeff))
    return M
```

9 La méthode `multiplie` de la classe `Maya` est :

```
 def multiplie(self):
    result = Maya()
    result.nombre = [[1,0,0]] + self.nombre
    return result
```

10 Les résultats renvoyés par les appels de la fonction `mystere` sont :

- `mystere([0, 1, 1], [0, 3, 1], 0)` renvoie `([0, 4, 2], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)` renvoie `([1, 0, 0], 1)`

Cette fonction renvoie en effet la somme de deux « chiffres » Mayas.

11 La méthode `somme` de la classe `Maya` est :

```
def somme(self, maya2):
    if self.nbEtages() != maya2.nbEtages():
        return None
    result = Maya()
    ret = 0
    while not self.estVide():
        ch1 = self.retirer()
        ch2 = maya2.retirer()
        ch_somme, ret = mystere(ch1, ch2, ret)
        result.ajouter(ch_somme)
    return result
```

Téléchargez la classe complète.

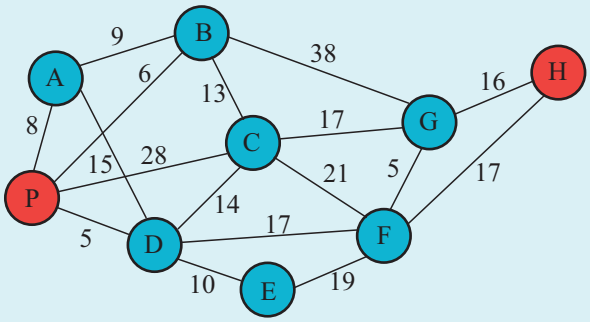
Chapitre 5

Les graphes : structures relationnelles

Plan du chapitre

Exercice type — Lycée Turgot, Paris

On considère le graphe pondéré suivant :



Implémenter ce graphe en Python, puis trouver le chemin le plus court de P à H.

Voir corrigé page 148

PROBLÉMATIQUE

La gestion des réseaux, qu'ils soient routiers, électriques ou sociaux, est un enjeu majeur de l'informatique.
Comment représenter de telles structures ?

COURS

1 Notion de graphes

Définition 1

Un *graphe* $G = (V, E)$ est la donnée :

- d'un ensemble V d'éléments, appelés *sommets* (appelés aussi *nœuds*) ;
- d'un ensemble E dont les éléments sont des parties à un ou deux éléments de V, nommés *arêtes* ou *arcs*.

Remarque : nous avons utilisé les lettres V et E pour respectivement *Vertex* (« sommet ») et *Edge* (« fil », sous-entendu : le fil reliant deux sommets).

Un graphe est utilisé pour représenter le lien entre divers objets ou entités.

Exemple : $V = \{A; B; C\}$ et $E = \big\{(A,B); (A,C); (C)\big\}$.
Dans ce cas, $G = (V, E)$ est le graphe de sommets A, B et C, où A et B sont reliés, A et C sont reliés et où C est connecté à lui-même.

Définition 2

La *représentation sagittale* d'un graphe $G = (V, E)$ est un schéma où les éléments de V sont représentés par des disques et où les éléments de E sont représentés par des traits (rectilignes ou courbés).

Exemple : une représentation sagittale du graphe défini dans l'exemple précédent est la suivante :

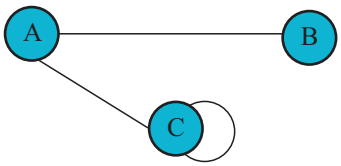


Figure 5.1 – Représentation sagittale du graphe $G = (V, E)$ avec $V = \{A; B; C\}$ et $E = \big\{(A,B); (A,C); (C)\big\}$

Par la suite, par souci de simplification, on confondra le graphe et sa représentation sagittale.

Définitions 3 : vocabulaire

- L'*ordre* d'un graphe est le nombre de ses sommets.
- Un graphe est dit *cyclique* s'il existe un chemin reliant n'importe quel sommet à lui-même.
- Un graphe est dit *connexe* si ses sommets peuvent être reliés deux à deux par une arête ou un arc.
- Un *chemin* est une suite de sommets reliés par une arête.

2 Les différents types de graphes

Il existe deux types de graphes : *orientés* et *non orientés*.

2.1 Graphes non orientés

Définition 4

Un graphe *non orienté* est un graphe dont les *arêtes* n'ont pas de sens.

Cela signifie que si $G = (V,E)$ est un graphe avec $V = \{A; B; C\}$ alors les éléments (A,B) et (B,A) de E sont identiques.

Exemple : trois villes A, B et C sont reliées par un réseau routier :

- A et B sont reliées par une route ;
- A et C sont reliées par une route ;
- B et C sont reliées par une route.

On peut alors représenter ce réseau routier par le graphe suivant $G = (V,E)$ où :

$$V = \{A; B; C\} \text{ et } E = \big\{(A,B); (A,C); (B,C)\big\}$$

dont une représentation sagittale est :

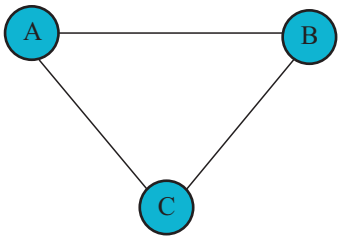


Figure 5.2 – Représentation sagittale d'un graphe non orienté

Ce graphe est d'ordre 3 (car il y a 3 sommets) ; il est de plus connexe et cyclique.

Remarque : la représentation sagittale d'un graphe n'est pas unique car on peut, par exemple, placer les sommets n'importe où.

Définitions 5

Dans un graphe non orienté,

- deux sommets reliés par une arête sont dits *adjacents* ;
- une arête ne possédant qu'un élément est une *boucle* ;
- un graphe qui ne comporte pas de boucle est qualifié de *simple*.

COURS

2.2 Graphes orientés

Définition 6

Un graphe *orienté* est un graphe dont les *arcs* ont un sens.

Remarque : cela signifie que dans le graphe $G = (V, E)$ où $V = \{A; B; C\}$, les éléments (A, B) et (B, A) sont distincts.

Exemple : imaginons un site internet composé de trois pages, nommées A, B et C.

- Sur A sont présents des liens vers B et C.
- Sur B figure un lien vers C.
- Sur C figure un lien vers A.

On peut représenter ceci à l'aide d'un graphe où les sommets sont A, B et C et où les arcs représentent les liens.

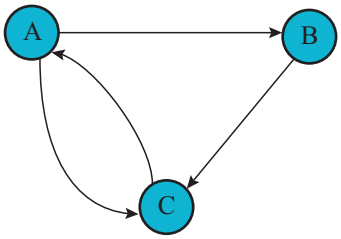


Figure 5.3 – Représentation sagittale d'un graphe orienté

À RETENIR

Un graphe non orienté a des *arêtes*.
Dans un graphe orienté, les arêtes sont souvent appelées *arcs*.

Définitions 7

Soit $G = (V, E)$ un graphe orienté. Si (A, B) est un arc de G alors :

- A est appelé le *prédécesseur* de B ;
- B est appelé le *successeur* de A.

Remarque : on dit aussi que pour un arc (A, B), A est son *origine* et B son *extrémité*.

2.3 Graphes pondérés

Définitions 8

On dit d'un graphe qu'il est *pondéré* (ou *valué*) quand ses arêtes ou ses arcs sont couplés avec des nombres. Ces nombres sont appelés des *poids*.

Exemple (graphe pondéré) : imaginons le cas d'un livreur dont le parcours peut être représenté par le graphe suivant :

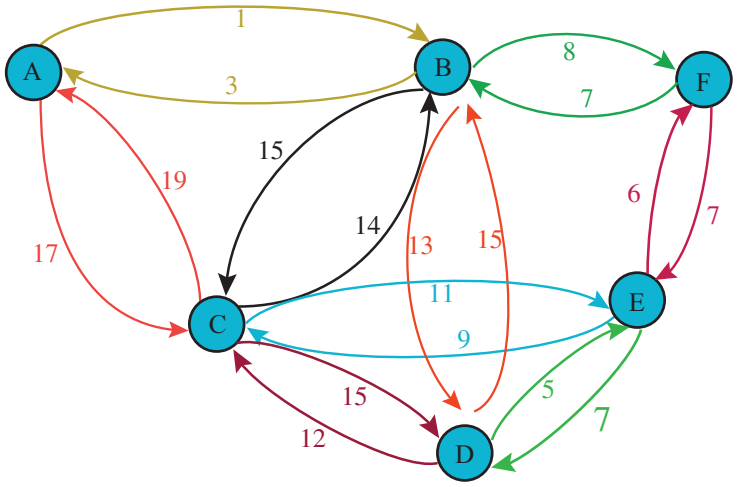


Figure 5.4 – Graphe du livreur

Les nombres attribués à chaque arc représentent ici le temps (en minutes) qu'il met pour aller d'un sommet à un autre.

Définition 9 : plus court chemin

Dans le cas d'un graphe pondéré, on appelle *plus court chemin* le chemin dont la somme des pondérations est minimale.

Exemple : dans l'exemple précédent, le plus court chemin de A à E est :

$$\text{A} - \text{B} - \text{F} - \text{E}$$

dont le poids total est : $1 + 8 + 7 = 16$.

3 Matrices de représentation

3.1 Matrice d'adjacence (graphe non pondéré)

Définition 10

Soit G un graphe d'ordre n. On appelle *matrice d'adjacence de G* la matrice carrée $M = (m_{ij})$ d'ordre n telle que :

- si G est non orienté,

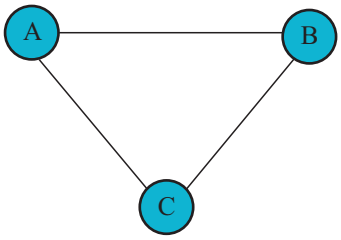
$$m_{ij} = \begin{cases} 1 & \text{si les sommets } i \text{ et } j \text{ sont adjacents} \\ 0 & \text{sinon} \end{cases}$$

COURS

- si G est orienté,

$$m_{ij} = \begin{cases} 1 & \text{si le sommet } i \text{ est l'origine d'un arc} \\ & \text{dont le sommet } j \text{ en est l'arrivée} \\ 0 & \text{sinon} \end{cases}$$

Exemple :



Ce graphe admet pour matrice d'adjacence :

$$M = \begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{matrix} A \\ B \\ C \end{matrix} \end{array}$$

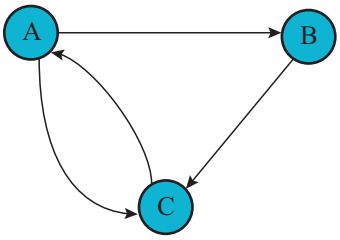
En effet,

- le sommet A n'est pas relié à lui-même (donc il y a un « 0 » à l'intersection de la 1re ligne et de la 1re colonne) ;
- les sommets A et B sont reliés (donc il y a un « 1 » à l'intersection de la 1re ligne et de la 2e colonne, ainsi qu'à l'intersection de la 1re colonne et de la 2e ligne ;
- etc.

Propriété 1

La matrice d'adjacence d'un graphe non orienté est nécessairement symétrique par rapport à sa diagonale principale (diagonale qui va du haut-gauche vers le bas-droit de la matrice).

Exemple (graphe orienté) :



Ce graphe admet pour matrice d'adjacence :

$$M = \begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{matrix} A \\ B \\ C \end{matrix} \end{array}$$

En effet,

- le sommet A n'est pas relié à lui-même (d'où le « 0 » à l'intersection de la 1re ligne et 1re colonne ;
- il y a un arc de A vers B, d'où le « 1 » à l'intersection de la 1re ligne et 2e colonne mais pas d'arc de B vers A, d'où le « 0 » à l'intersection de la 2e ligne et 1re colonne ;
- etc.

3.2 Matrice de valuation (graphe pondéré)

Définition 11

La *matrice de valuation* d'un graphe pondéré d'ordre n est la matrice $(m_{i,j})$ telle que :

- $m_{i,j}$ est le poids de l'arête (ou arc) du sommet i au sommet j ;
- $m_{i,j} = +\infty$ si aucune arête (ou arc) ne relie les sommets i et j.

Remarque : en Python, par convention, on remplace « $+\infty$ » par « 0 » ou « None ».

Exemple : reprenons le graphe de l'exemple du livreur (figure 5.4 page 133).
Sa matrice de valuation est :

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{pmatrix} 0 & 1 & 17 & 0 & 0 & 0 \\ 3 & 0 & 15 & 13 & 0 & 8 \\ 19 & 14 & 0 & 15 & 11 & 0 \\ 0 & 15 & 12 & 0 & 5 & 0 \\ 0 & 0 & 9 & 7 & 0 & 6 \\ 0 & 7 & 0 & 0 & 7 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix}$$

4 Implémentation d'un graphe

4.1 Densité d'un graphe

Définition 12

La densité d'un graphe est égale au rapport du nombre d'arêtes sur le nombre total d'arêtes possibles.
La densité est donc un nombre compris entre 0 et 1.

COURS

Remarques

- Une densité nulle correspond à un graphe sans arêtes : tous les sommets sont isolés.
- Une densité égale à 1 correspond à un graphe complet : chaque sommet est relié à tous les sommets, y compris lui-même.

Exemple : pour un graphe simple non orienté à n sommets, chaque sommet ne peut être relié qu'à $n-1$ sommets au maximum (car le graphe est simple, donc il n'y a pas de boucle). Il y a donc $n(n-1)$ arêtes au maximum.
La densité du graphe est alors :

$$d = \frac{2 \times \text{nombre d'arêtes}}{n(n-1)}.$$

En effet, si une arête est entre les sommets A et B, alors on compte deux fois cette arête dans le calcul de la densité (une pour A, et une pour B) car le graphe est non orienté.

Définition 13 : graphe creux, graphe denses

- Un graphe à n sommets est *creux* si son nombre d'arêtes « est proche » de n.
- Un graphe à n sommets est *dense* si son nombre d'arêtes « est proche » de n^2.

Propriété 2

Un graphe est dense si sa densité est « assez proche » de 1.
Un graphe est creux si sa densité est « assez éloignée » de 1.

Remarque : la notion de *graphe dense* n'est pas une notion précise. En effet, « assez proche » peut différer d'un individu à l'autre.
Aussi, dans la pratique, on pourra considérer qu'un graphe est dense si sa densité est au moins égale à 0,75 (tout en ayant à l'esprit que cette valeur est arbitraire).

Exemple : reprenons le graphe orienté 5.4 de la page 133.
Le nombre d'arêtes est égal à 18 et le nombre maximal d'arêtes est :

$$n(n-1) = 6 \times 5 = 30.$$

La densité de ce graphe est donc :

$$d = \frac{18}{30} = 0{,}6.$$

On peut alors estimer que le graphe est creux.

4.2 Graphes non pondérés

4.2.1 - Graphes denses : par matrice d'adjacence

Dans le cas des graphes denses, on attribue un booléen à chaque coefficient de la matrice.

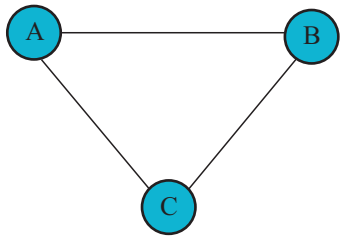
```
Pour i variant de 0 à n-1:
   Pour j variant de 0 à n-1:
     m[i][j] ← 1 si i et j sont reliés, 0 sinon
   Fin du Pour j
Fin du Pour i
```

L'un des avantages de cette représentation est sa simplicité : étant donnés deux sommets de rangs i et j, le booléen à la i-ième ligne et j-ième colonne représente l'existence ou l'absence d'une arrête les reliant.

Exemple : si nous reprenons le graphe de la figure 5.2 de la page 131, dont la matrice d'adjacence est $M = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ comme vu précédemment, nous pouvons implémenter ce graphe en Python comme ci-dessous.

En Python

```
G = [
    [0,1,1],
    [1,0,1],
    [1,1,0]
]
```



La complexité est ici en $O\left(|S|^2\right)$, où $|S|$ est le nombre de sommets du graphe.

En effet, il y a deux boucles de n itérations, donc $n \times n = n^2$ affectations pour les coefficients de la matrice, et n^2 affectations pour i et j, soit en tout $2n^2$ affectations élémentaires.

Or, $n = |S|$; la complexité est donc bien en $O(n^2)$.

4.2.2 - Graphes creux : par liste d'adjacence

Définition 14

Soit $G = (V, E)$ un graphe tel que $V = \{A_1; A_2; \ldots; A_n\}$.
L'*ensemble d'adjacence* d'un sommet A_k, $1 \leqslant k \leqslant n$, est l'ensemble des A_p tels que A_k et A_p sont reliés par une arête (graphe non orienté) ou un arc (graphe orienté).

COURS

Dans le cas des graphes creux, on utilise l'*ensemble d'adjacence* pour l'implémentation. L'idée consiste alors, pour chaque sommet du graphe, à construire cet ensemble.

```
G ← liste vide
Pour k variant de 0 à n-1:
   L[k] est une liste vide
   Pour p variant de 0 à n-1:
     Si A_k est relié à A_p:
       On ajoute A_p dans la liste L[k]
     Fin du Si
   Fin du Pour p
   G ← G + (A_k , L[k])
Fin du Pour k
```

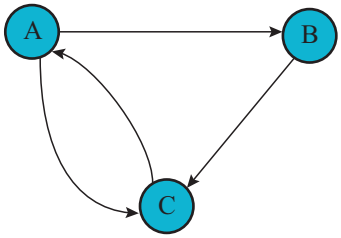
La taille de la représentation est ici en $O(|S| + |A|)$, où $|S|$ et $|A|$ sont respectivement le nombre de sommets et le nombre d'arêtes du graphe.

En Python, on pourra par exemple utiliser un dictionnaire.

Exemple : reprenons le graphe représenté sur la figure 5.3 page 132 en y ajoutant un sommet D, origine d'aucun arc. Nous pouvons l'implémenter comme suit :

En Python

```
G = {
    'A' : ['B','C'],
    'B' : ['C','D'],
    'C' : ['A','D'],
    'D' : [None]
}
```



Remarque : il existe plusieurs manières d'implémenter le sommet D n'est pas unique. Nous aurions aussi pu écrire par exemple :

```
'D' : [ ]
```

4.2.3 - À l'aide d'une classe

Un graphe peut être implémenté à l'aide d'une classe.

Voyons page suivante ce que cela peut donner en définissant par liste d'adjacence le graphe non orienté de la figure 5.5 page 142.

Implémentation d'un graphe par une classe

```
class Graphe:
    def __init__(self):
        self.liste_adjacence = {}

    def addArete(self, u, v):
        if v not in self.liste_adjacence:
            self.liste_adjacence[v] = []
        if u not in self.liste_adjacence:
            self.liste_adjacence[u] = []
        self.liste_adjacence[u].append(v)
        self.liste_adjacence[v].append(u)

G = Graphe()
G.addArete('A', 'B')
G.addArete('A', 'D')
...
G.addArete('G', 'H')
for key,value in G.liste_adjacence.items():
    print(key,":",value)
```

Ce programme affiche :

```
B : ['A', 'C']                C : ['B', 'D']
A : ['B', 'D', 'E']           F : ['E', 'G']
D : ['A', 'C', 'E']           G : ['E', 'F', 'H']
E : ['A', 'D', 'F', 'G']      H : ['G']
```

On peut ensuite utiliser cette classe afin d'y implémenter des parcours.

Téléchargez la classe complète.

4.3 Graphes pondérés

4.3.1 - Par liste d'adjacence

On peut implémenter un graphe pondéré à l'aide d'un dictionnaire dont les valeurs sont aussi des dictionnaires.

Ainsi, pour le graphe de la figure 5.4 page 133, une implémentation possible est celle présentée page suivante (on ne met ici que le début du dictionnaire).

COURS

Implémentation d'un graphe pondéré en Python

```
G = {
    'A' : {
        'B' : 1,
        'C' : 17
    },
    'B' : {
        'A' : 3,
        'C' : 15,
        'D' : 13,
        'F' : 8
    },
    ...
}
```

Les clés du dictionnaire principal sont les sommets du graphe et les valeurs sont des dictionnaires où chaque clé représente un sommet connecté à la clé du dictionnaire principal et où chaque valeur est la pondération de l'arc.

Remarque : un sommet qui n'est l'origine d'aucun arc peut être représenté par un dictionnaire vide par exemple.

Implémenter un graphe pondéré peut alors se faire en créant une classe comme ceci :

Implémentation d'un graphe pondéré par liste d'adjacence

```
class GraphePond:
    def __init__(self):
        self.liste_adjacence = {}

    def addArc(self,u,v,p):
        if u not in self.liste_adjacence:
            self.liste_adjacence[u] = { v : p }
        else:
            self.liste_adjacence[u][v] = p
```

Cela donne en mode console :

```
>>> G = GraphePond()
>>> G.addArc('K','M',4)
>>> G.addArc('K','E',1)
>>> G.addArc('A','B',5)
>>> G.Liste
{'K': {'M': 4, 'E': 1}, 'A': {'B': 5}}
```

4.3.2 - Par matrice d'adjacence

Sur le même modèle que précédemment, on peut créer une classe qui définit un graphe quand on connaît sa matrice d'adjacence :

Implémentation d'un graphe pondéré par matrice d'adjacence

```
class GraphePondM:
    def __init__(self):
        self.matrice = []

    def add(self,L):
        self.matrice.append(L)
```

En mode console, on aura par exemple :

```
>>> G = GraphePondM()
>>> G.add([0,1,2])
>>> G.add([2,0,5])
>>> G.add([5,3,0])
>>> G.Liste
[[0, 1, 2], [2, 0, 5], [5, 3, 0]]
```

5 Algorithmique

5.1 Parcours d'un graphe

Parcourir un graphe consiste à lister tous ses sommets une seule fois. Il peut y avoir deux raisons principales au parcours d'un graphe : lister simplement tous les sommets ou trouver un chemin pour relier un sommet à un autre.

5.2 Parcours en largeur

L'idée est d'utiliser une file de la manière suivante :

1. on part d'un sommet et on initialise une file vide (file) qui contiendra les sommets à visiter ;
2. on ajoute le premier sommet dans une liste `sortie` initialement vide ;
3. tant que `file` n'est pas vide, on défile `file` et pour chaque voisin du sommet défilé, s'il n'a pas été visité, on le marque comme visité et on l'ajoute à `sortie`.

COURS

Exemple : regardons par rapport au graphe suivant :

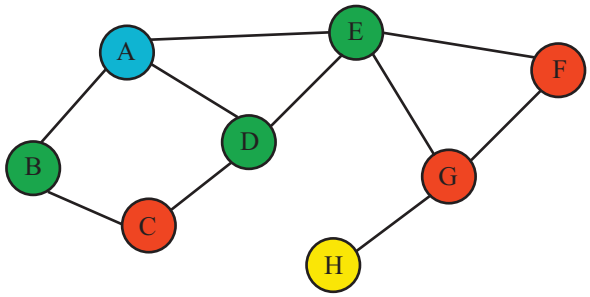


Figure 5.5 – Parcours en largeur d'un graphe

Nous souhaitons parcourir ce graphe en partant du sommet A.

- On défile alors A. Les sommets B, D et E sont les voisins de A ; ils sont donc enfilés en premier (peu importe l'ordre).
- On défile B : parmi les voisins de B, seul C n'a pas été visité donc on l'ajoute à la file.
- On défile D : parmi ses voisins, tous ont été visités donc on ne fait rien.
- On défile E : parmi ses voisins, G et F n'ont pas été visités donc on les enfile.
- On défile C : pas de voisins non visités.
- On défile G : le sommet H n'a pas été visité donc on l'enfile.
- On défile F : pas de voisins non visités.
- On défile H : pas de voisins non visités.

Parcours en largeur écrit en Python

```
G = {
    'A' : ['B','D','E'] ,
    'B' : ['A','C'] ,
    'C' : ['B','D'],
    'D' : ['A','C','E'],
    'E' : ['A','D','G','F'],
    'F' : ['E','G'],
    'G' : ['E','F','H'],
    'H' : ['G']
}

sortie = []
file = []
file.append('A')
```

```
while len(file) > 0:
    S = file.pop(0)  # Utilise pop(0) pour défiler
    if S not in sortie:
        sortie.append(S)
        unvisited = [n for n in G[S] if n not in sortie]
        file.extend(unvisited)

print(sortie)
```

Ce programme affiche :

```
['A', 'B', 'D', 'E', 'C', 'G', 'F', 'H']
```

L'algorithme du parcours en largeur est le suivant :

Parcours en largeur d'un graphe

```
F est une file vide
On enfile le premier sommet (S)
Marquer S comme visité
Tant que F n'est pas vide:
   S ← tête de F
   Défiler F
   Pour chaque voisin (v) de S:
     Si v n'est pas visité:
       Marquer v comme visité
       Enfiler v dans F
   Fin do Pour
Fin du Tant que
```

Pour le cas de graphes implémentés par matrice d'adjacence, vous trouverez un exercice à ce sujet (exercice 8 page 154).

L'algorithme de parcours en largeur permet de dresser la liste des sommets d'un graphe par distance croissante au sommet d'origine. On peut ainsi s'en servir pour trouver un chemin de distance minimale entre deux sommets.

5.3 Parcours en profondeur

L'idée est d'utiliser une pile de la manière suivante :

1. on part d'un sommet que l'on empile ;
2. si le sommet de la pile a des voisins qui ne sont pas dans la pile et qui ne sont pas déjà passés dans la pile, on choisit l'un des ces voisins et on l'empile. Dans le cas contraire, on dépile (on supprime l'élément du haut de la pile) ;
3. on revient au point 2 tant que la pile n'est pas vide.

COURS

Exemple : le programme suivant est basé sur le même graphe que précédemment.

Parcours en profondeur écrit en Python

```
G = {
    'A' : ['B','D','E'],
    'B' : ['A','C'],
    'C' : ['B','D'],
    'D' : ['A','C','E'],
    'E' : ['A','D','G','F'],
    'F' : ['E','G'],
    'G' : ['E','F','H'],
    'H' : ['G']
}

sortie = []
pile = []
pile.append('A')

while pile:
    S = pile.pop()  # Utilise pop pour dépiler
    if S not in sortie:
        sortie.append(S)
        non_visites = [n for n in G[S] if n not in sortie]
        pile.extend(non_visites)

print(sortie)
```

Il affiche : `['A', 'E', 'F', 'G', 'H', 'D', 'C', 'B']`

5.4 Recherche d'un chemin

On peut s'inspirer de l'un des parcours (en largeur ou en profondeur) pour élaborer un algorithme permettant de trouver un chemin d'un sommet à un autre dans un graphe.

Algorithme de recherche d'un chemin

```
G est un graphe
start ← sommet de départ
end ← sommet d'arrivée
P est une pile
```

Algorithme de recherche d'un chemin (suite)

```
Empiler le couple (start,[start])
Tant que P n'est pas vide:
   (S,path) ← tête de P
   list_nodes ← liste des sommets adjacents à S
                qui ne sont pas dans path
   Pour i dans list_nodes:
     Si i = end:
       Afficher path + [i]
     Sinon:
       Empiler (i,path+[i])
     Fin du Si
   Fin du Pour
Fin du Tant que
```

Le programme suivant donne par exemple tous les chemins qui mènent de A à H sans passer deux fois par le même sommet :

Chemins d'un sommet à un autre en Python

```
def chemins(start, end):
    pile = []
    pile.append((start, [start]))
    liste_chemins = []

    while pile:
        (S, path) = pile.pop()
        list_nodes = [n for n in G[S] if n not in path]
        for i in list_nodes:
            if i == end:
                liste_chemins.append(path + [i])
            else:
                pile.append((i, path + [i]))

    return liste_chemins

G = {
    'A' : ['B','D','E'],
    'B' : ['A','C'],
    'C' : ['B','D'],
    'D' : ['A','C','E'],
    'E' : ['A','D','G','F'],
    'F' : ['E','G'],
    'G' : ['E','F','H'],
    'H' : ['G']
}

print( chemins('A','H') )
```

COURS

Expliquons ce programme :

- on commence par définir le sommet d'origine et le sommet de destination ;
- on initialise ensuite une liste qui contiendra des couples de la forme :
  `(sommet de départ , [chemin à partir du sommet de départ])`
- tant que la pile n'est pas vide, on définit le couple `S,path` comme étant la tête de la pile, et on dépile ;
- on construit alors la liste `list_nodes` comme étant tous les sommets reliés à `S` qui ne sont pas déjà dans la liste `path` (représentant le chemin) ;
- on parcourt alors la liste `list_nodes` : si l'élément `i` sur lequel nous sommes est le sommet d'arrivée alors on affiche le chemin complet en y ajoutant `i`, qui est le sommet de destination ; sinon, on ajoute à la pile le couple composé de `i` accompagné du chemin auquel nous avons ajouté le sommet `i`.

Ce programme affiche alors :

```
['A', 'E', 'F', 'G', 'H']
['A', 'E', 'G', 'H']
['A', 'D', 'E', 'F', 'G', 'H']
['A', 'D', 'E', 'G', 'H']
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
['A', 'B', 'C', 'D', 'E', 'G', 'H']
```

5.5 Recherche d'un plus court chemin

Retrouvez en vidéo le principe de l'algorithme de Dijkstra.

Pour la recherche de plus courts chemins, l'algorithme le plus connu est l'algorithme de Dijkstra.

Il recherche autant de « plus courts chemins » qu'il existe de sommets accessibles depuis le sommet de départ. C'est pour cela qu'il ne reçoit pas, en entrée, le sommet de destination souhaité.

Dans cet algorithme,

- $G = (V, E)$ est un graphe pondéré ;
- `L` est une liste contenant les distances du sommet de départ vers les autres sommets ;
- `P` est une liste contenant le prédécesseur dans le chemin le plus court à partir du sommet de départ ;
- `M` est une liste de tous les sommets déjà traités ;
- `poids(A,B)` désigne le poids de l'arc ou arête qui va de A à B.

Algorithme de Dijkstra

```
*- initialisation -*
D ← sommet de départ
L[D] ← 0
P[D] ← D
Pour tout S (sommet) ≠ D:
   Si S est un successeur de D:
     L[S] ← poids(D,S)
     P[S] ← D
   Sinon:
     L[S] ← "0"
     P[S] ← ""
   Fin du Si
Fin du Pour
M ← S
*- Traitement -*
Tant que M ≠ V:
   Choisir S ∈ V \ M tel que ∀ J ∈ V \ M, L[S] ⩽ L[J]
   Pour J ∈ V \ M tel que J successeur de S:
     Si L[J] > L[S] + poids(S,J):
       L[J] ← L[S] + poids(S,J)
       P[J] ← S
     Fin du Si
   Fin du Pour
Ajouter à M le sommet S
Fin du Tant que
```

Téléchargez l'implémentation de Dijkstra en Python.

5.6 Recherche d'un cycle

Définition 15

Un *cycle* est un chemin dont l'origine et l'extrémité sont identiques.

Exemple : dans le graphe de la figure 5.5 page 142, le chemin :

A – B – C – D – A

est un cycle.

Pour obtenir tous les cycles d'un graphe, on peut s'inspirer de l'algorithme de recherche d'un chemin.

COURS

Voici ce que cela donne en Python :

Recherche d'un cycle en Python

```
G = {
    'A' : ['B','D','E'],
    'B' : ['A','C'],
    'C' : ['B','D'],
    'D' : ['A','C','E'],
    'E' : ['A','D','G','F'],
    'F' : ['E','G'],
    'G' : ['E','F','H'],
    'H' : ['G']
}

for key in G:
    start = key
    pile = []
    pile.append((start, []))

    while len(pile) > 0:
        (S, path) = pile.pop()  # Utilise pop pour dépiler
        list_nodes = [n for n in G[S] if n not in path]
        for i in list_nodes:
            if i == start:
                print([start] + path + [i])
            else:
                pile.append((i, path + [i]))
```

Solution de l'exercice type

Lycée Turgot, Paris

Nous pouvons avant tout construire la matrice d'adjacence de ce graphe :

$$M = \begin{array}{c} \begin{matrix} P & A & B & C & D & E & F & G & H \end{matrix} \\ \begin{pmatrix} 0 & 8 & 6 & 28 & 5 & 0 & 0 & 0 & 0 \\ 8 & 0 & 9 & 0 & 15 & 0 & 0 & 0 & 0 \\ 6 & 9 & 0 & 13 & 0 & 0 & 0 & 38 & 0 \\ 28 & 0 & 13 & 0 & 14 & 0 & 21 & 17 & 0 \\ 5 & 15 & 0 & 14 & 0 & 10 & 17 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 19 & 0 & 0 \\ 0 & 0 & 0 & 21 & 17 & 19 & 0 & 5 & 17 \\ 0 & 0 & 38 & 17 & 0 & 0 & 5 & 0 & 16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 17 & 16 & 0 \end{pmatrix} \begin{matrix} P \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix} \end{array}$$

Solution de l'exercice type (suite)

Lycée Turgot, Paris

À l'aide de la classe page 141, on peut alors implémenter le graphe. Mais on peut aussi utiliser le programme « `dijkstra` » fourni page 147.

Téléchargez le programme.

Les sommets sont représentés par des nombres (ici, de 0 à 8) et on cherche le plus court chemin de 0 à 8.
Ce programme affiche :

```
Plus courts chemins de...
Longueur 5 : 0 -> 4
Longueur 6 : 0 -> 2
Longueur 8 : 0 -> 1
Longueur 15 : 0 -> 4 -> 5
Longueur 19 : 0 -> 4 -> 3
Longueur 22 : 0 -> 4 -> 6
Longueur 27 : 0 -> 4 -> 6 -> 7
Longueur 39 : 0 -> 4 -> 6 -> 8
```

Ainsi, le chemin le plus court est : P – D – F – H.

Voir énoncé page 129

INTERROS

1 QCM Vérification de connaissances

5 min

Corrigé p. 162

On considère le graphe $\mathcal{G}$ dont la représentation sagittale est la suivante :

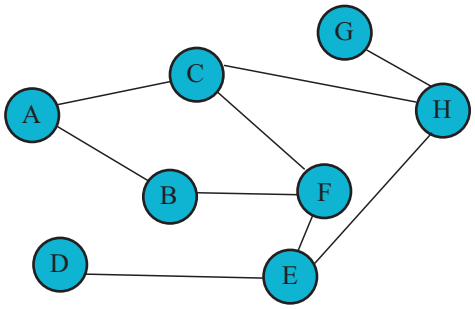


Figure 5.6 – Représentation sagittale du graphe $\mathcal{G}$

Pour chacune des questions suivantes, plusieurs propositions de réponses sont faites dont une seule est correcte. Laquelle ?

1 L'ordre de $\mathcal{G}$ est égal à :

a 9 **b** 8

2 Le graphe $\mathcal{G}$ est :

a orienté **b** non orienté

3 Le graphe $\mathcal{G}$ est :

a cyclique **b** non cyclique

4 Le graphe $\mathcal{G}$ est :

a connexe **b** non connexe

5 La matrice d'adjacence de $\mathcal{G}$ est :

a

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F & G & H \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix} \end{array}$$

b

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F & G & H \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix} \end{array}$$

Théorie

2 Le plan de ville

★ | 15 min | Corrigé p. 162

Lycée André Theuriet, Civray

Voici le plan simplifié du centre ville d'Aix-en-Provence :

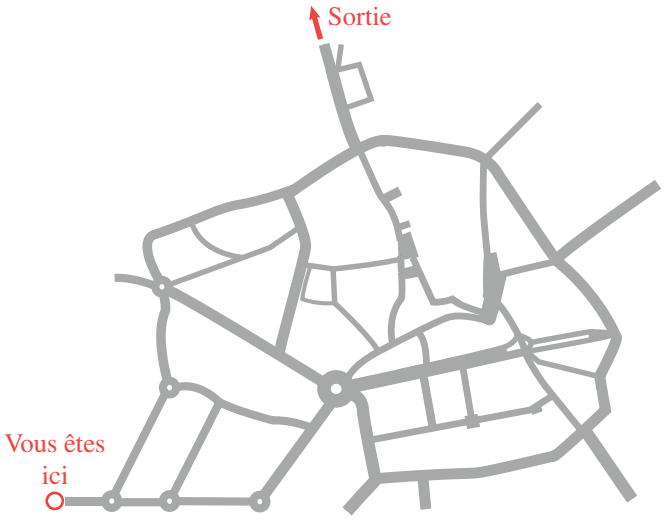


Figure 5.7 – Plan du centre ville d'Aix-en-Provence

Représenter ce centre ville sous forme de graphe en considérant que chaque sens giratoire et chaque intersection de rues est un sommet, et que chaque route est une arête.

On supposera qu'il n'y a pas de rue à sens unique et que le graphe peut être par conséquent non orienté.

Quelles caractéristiques a ce graphe ?

Donner un chemin pour aller du point marqué « Vous êtes ici » à la sortie indiquée par la flèche afin de passer par le moins d'intersections possible ?

3 La maison piégée

★★ | 20 min | Corrigé p. 163

Lycée Jean Macé, Niort

Une maison comporte quatre entrées et est composée de plusieurs pièces, certaines sans danger et d'autres qui sont piégées. Dans l'une des pièces sans danger se trouve une récompense (voir schéma page suivante).

1. Représenter cette maison à l'aide d'un graphe le plus simple possible, permettant de trouver l'entrée la plus optimisée et le chemin le plus court pour aller à la pièce « récompense » en évitant tous les pièges.

2 Trouver ce chemin optimal.

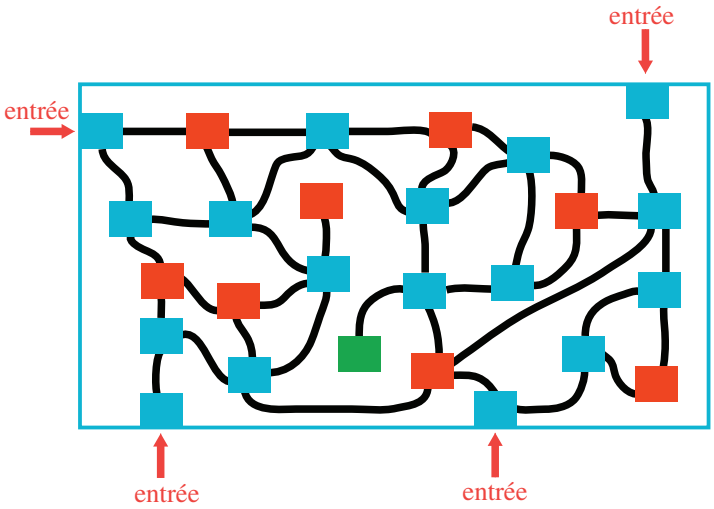


Figure 5.8 – Carte de la maison piégée

Plus court chemin

4 Le loup et compagnie...

★ 15 min Corrigé p. 165

Lycée François Villon, Les Mureaux

Sur la rive d'un fleuve se trouvent un loup, une chèvre, un chou et un passeur.

Le problème consiste à faire tous les faire passer sur l'autre rive à l'aide d'une barque, menée par le passeur, en respectant les règles suivantes :

- la chèvre et le chou ne peuvent pas rester sur la même rive sans le passeur,
- de même pour la chèvre et le loup,
- le passeur ne peut mettre qu'un seul « passager » avec lui.

On décide de représenter le passeur par la lettre P, la chèvre par C, le loup par L et le chou par X.

1 Représenter ce problème à l'aide d'un graphe où les sommets sont tous les états possibles sur la rive de départ (par exemple, « $PLCX$ » est un sommet représentant le fait que tous sont sur la rive de départ).

2 Trouver alors une solution au problème en indiquant chacun des déplacements.

5 Algorithme de Dijkstra

★★ 10 min Corrigé p. 166

Lycée Jean-Joseph Fourier, Auxerre

On considère le graphe suivant :

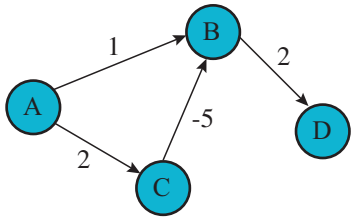


Figure 5.9 – Graphe orienté pondéré

1 Appliquer l'algorithme de Dijkstra à ce graphe en partant du sommet A.

2 Montrer que la distance entre A et D obtenue n'est pas minimale.

6 Le politicien

★★ 15 min Corrigé p. 166

Lycée Henri Alain-Fournier, Bourges

Un politicien se trouve à sa permanence (en P) et doit se rendre à l'hôpital (en H) le plus rapidement possible.

Le graphe ci-dessous représente le temps de trajet pour aller d'un point à un autre.

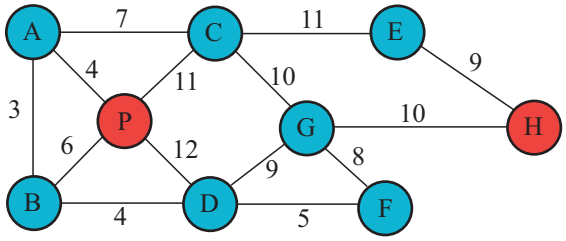


Figure 5.10 – Graphe pondéré par les temps de trajet

Quel chemin doit-il emprunter pour y arriver au plus vite ?

INTERROS

Implémentation

7 Matrice d'adjacence aléatoire

★ 5 min Corrigé p. 167

Lycée Sophie Germain, Paris

Écrire une fonction en Python admettant un paramètre entier n et permettant de créer une matrice d'adjacence aléatoire de dimensions $n \times n$.

8 Par matrice d'adjacence

★★ 15 min Corrigé p. 168

Lycée Camille Jullian, Bordeaux

On considère le graphe de sommets A, B, C, D, E et F dont la matrice d'adjacence est la suivante :

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix}$$

À l'aide de M, implémenter ce graphe par liste d'adjacence.

9 Flux migratoires

★★ 20 min Corrigé p. 169

Lycée polyvalent Émile Zola, Aix-en-Provence

Une région est composée de trois départements que l'on notera X, Y et Z.

Chaque année,

- 5 % des habitants du département X déménagent pour le département Y ;
- 1 % des habitants du département X déménagent pour le département Z ;
- 7 % des habitants du département Y déménagent pour le département X ;
- 3 % des habitants du département Y déménagent pour le département Z ;
- 2 % des habitants du département Z déménagent pour le département X ;
- 4 % des habitants du département Z déménagent pour le département Y ;
- 1 % des habitants du département X déménagent pour une autre région ;
- 3 % des habitants du département Y déménagent pour une autre région ;
- 2 % des habitants du département Z déménagent pour une autre région.
- On suppose que le pourcentage d'habitants des autres régions qui aménagent dans un des départements X, Y ou Z est assez faible pour l'assimiler à 0.

1 Représenter ces flux migratoires à l'aide d'un graphe (on n'oubliera pas les personnes restant dans leur département).

2 Donner la matrice de transition M de ce graphe.

3 À l'aide de `numpy`, implémenter M.

4 À l'aide du module `numpy.linalg` et de `matrix_power`, déterminer M^{10}.

5 En 2019, un recensement a établi qu'il y avait :

- 1,081 million d'habitants dans le département X,
- 1,076 million d'habitants dans le département Y,
- 0,327 million d'habitants dans le département Z,
- 67 millions d'habitants dans le pays.

On note $P_0 = \begin{pmatrix}1{,}081 & 1{,}076 & 0{,}327 & 64{,}516\end{pmatrix}$ la matrice ligne représentant l'état initial.

On suppose que le nombre d'habitants dans chaque département au bout de n années est donné par la matrice $P_n = P_0 \times M^n = \begin{pmatrix}x_n & y_n & z_n & a_n\end{pmatrix}$, où x_n, y_n et z_n représentent respectivement le nombre d'habitants des départements X, Y et Z (et où a_n représente le nombre d'habitants dans les autres régions).

À l'aide de la fonction `dot` de `numpy`, qui permet de calculer le produit de deux matrices, déterminer x_{10}, y_{10} et z_{10}.

6 Écrire un programme permettant de voir l'évolution du nombre d'habitants dans chaque département au cours des 200 années à venir. Que constate-t-on ? Quelles conclusions peut-on faire ?

10 Une simple histoire de temps

★★ 20 min Corrigé p. 172

Lycée Sainte-Clotilde, Strasbourg

Dans un pays imaginaire, si le temps d'un jour est pluvieux ou neigeux, alors il reste inchangé dans 50 % des cas le lendemain et ne devient beau que dans 25 % des cas. Lorsque le temps est beau, le lendemain il est pluvieux ou neigeux de manière équiprobable mais il ne restera pas ensoleillé deux jours à la suite.

Les habitants affirment qu'il ne fait beau qu'un jour sur cinq. Ont-ils raison ?

Pour répondre à cette question, on pourra :

- établir le graphe représentant les changements climatiques ;
- déterminer sa matrice de transition M ;
- s'aider de Python, et plus particulièrement de `numpy`, pour déterminer la probabilité qu'il fasse beau un jour donné en calculant une puissance de M assez grande.

INTERROS

11 Le labyrinthe

★★★ 30 min Corrigé p. 173

Lycée Camille Jullian, Bordeaux

Le « labyrinthe de Pan » est une attraction d'une fête foraine : c'est un labyrinthe dont les murs intérieurs sont des miroirs. Le plan de ce labyrinthe est donné ci-dessous sur une grille :

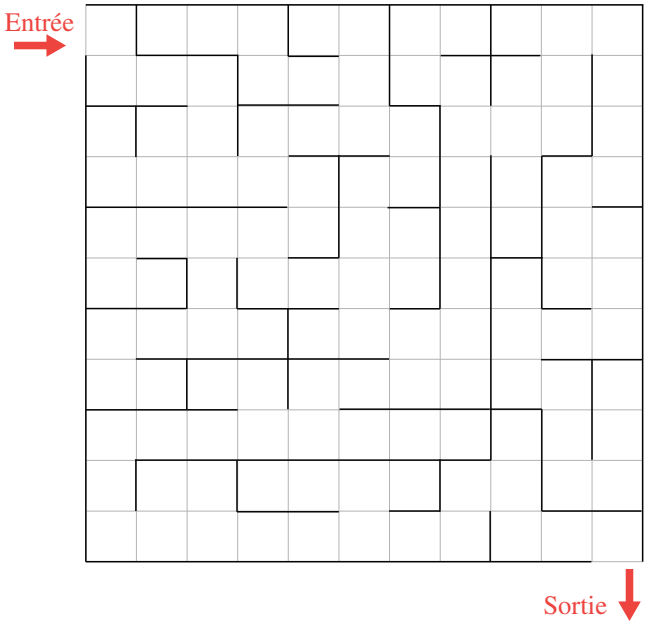


Figure 5.11 – Plan du « Labyrinthe de Pan »

En considérant chaque case comme un sommet de graphe, implémenter ce labyrinthe en Python, puis trouver le chemin le plus court de l'entrée à la sortie.

On pourra pour cela s'aider de la classe `Graphe` complète téléchargeable à la page 139.

Objectif bac

12 Recette

★★★ 45 min Corrigé p. 174

Lycée Richelieu, Versailles

On s'intéresse à la fabrication de pain. La recette est fournie sous la forme de tâches à réaliser page ci-contre. Cette recette est réalisée par une personne seule.

- **a** Préparer 500 g de farine.
- **b** Préparer 1/3 de litre d'eau (33cl).
- **c** Préparer 1 c. à café de sel.
- **d** Préparer 20 g de levure de boulanger.
- **e** Faire tiédir l'eau dans une casserole.
- **f** Délayer la levure dans l'eau tiède.
- **g** Laisser reposer la levure 5 minutes.
- **h** Préparer un grand saladier.
- **i** Verser la farine dans le saladier.
- **j** Verser le sel dans le saladier.
- **k** Mélanger la farine et le sel puis creuser un puits.
- **l** Verser l'eau mélangée à la levure dans le puits.
- **m** Pétrir jusqu'à obtenir une pâte homogène.
- **n** Couvrir à l'aide d'un linge humide et laisser fermenter au moins 1h30.
- **o** Disposer dans le fond du four un petit récipient contenant de l'eau.
- **p** Préchauffer un four à 200 degrés Celsius.
- **q** Fariner un plan de travail.
- **r** Verser la pâte à pain sur le plan de travail.
- **s** Pétrir rapidement la pâte à pain.
- **t** Disposer la pâte dans un moule à cake.
- **u** Mettre au four pour 15 à 20 minutes, arrêter le four et sortir le pain.

La figure 5.12 représente les différentes tâches et les dépendances entre ces tâches sous la forme d'un graphe. Chaque sommet du graphe représente une tâche à réaliser. Les dépendances entre les tâches sont représentées par les arcs entre les sommets.

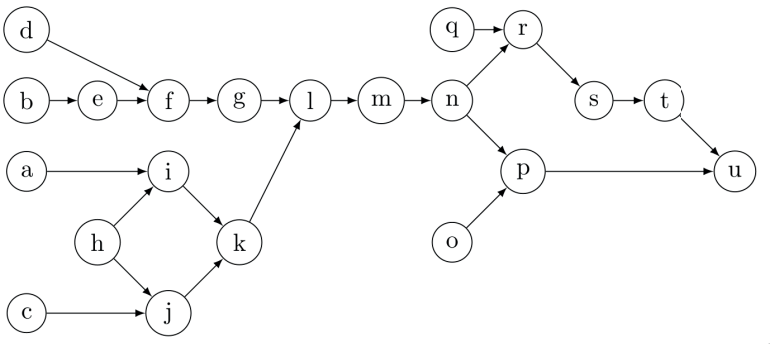


Figure 5.12 – Recette du pain : tâches à effectuer avec leurs dépendances

INTERROS

1 Dire, sans justifier, s'il s'agit d'un graphe orienté ou non orienté.

2 D'après le graphe, dire s'il est possible d'effectuer les réalisations dans chacun des ordres suivants :

(a) réaliser la tâche (f) puis la tâche (g)
(b) réaliser la tâche (g) puis la tâche (f)
(c) réaliser la tâche (i) puis la tâche (j)
(d) réaliser la tâche (j) puis la tâche (i)

3 Donner toutes les tâches qu'il faut nécessairement avoir réalisées depuis le début pour pouvoir réaliser la tâche (k). Ne donner que les tâches nécessaires.

4 Indiquer, sans justifier, si le graphe de la Figure 5.12 contient un cycle.

Graphe de tâches

On s'intéresse désormais de manière plus générale à un graphe de tâches avec des dépendances.

Les sommets sont nommés par des indices. Comme précédemment, un arc orienté d'un sommet d'indice i à un sommet d'indice j signifie que la tâche représentée par le sommet d'indice i doit être réalisée avant la tâche représentée par le sommet d'indice j.

Figure 5.13 – Exemple de graphe de dépendances entre 6 tâches

5 Déterminer un ordre permettant de réaliser toutes les tâches représentées dans le graphe de la Figure 5.13 en respectant les dépendances entre les tâches.

Voici une matrice d'adjacence d'un graphe écrite en langage Python et telle que si $m[i][j] = 1$ alors il existe un arc qui va du sommet d'indice i au sommet d'indice j. Par exemple, $m[0][1] = 1$ alors il existe un arc qui va du sommet d'indice 0 au sommet d'indice 1.

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

6 Représenter le graphe associé à cette matrice d'adjacence. Les noms des sommets seront leurs indices.

7 Déterminer s'il est possible de trouver un ordre permettant de réaliser les tâches représentées par le graphe de la question 6 en respectant leurs dépendances. Si oui, donner l'ordre. Si non, expliquer pourquoi.

On donne le code Python d'une fonction `mystere`.

```
1  def mystere(graphe, s, n, ouverts, fermes, resultat):
2      """ Paramètres :
3          graphe un graphe représenté par une matrice d'
       adjacence
4          s l'indice d'un sommet du graphe
5          n le nombre de sommets du graphe
6          ouverts une liste de booléens permettant de
       savoir
7                  si le traitement d'un sommet a été commencé
8          fermes une liste de booléens permettant de
       savoir
9                  si le traitement d'un sommet a été terminé
10         Retour : False s'il y a eu un "problème", True
       sinon.
11         Le paramètre resultat sera modifié ulté
       rieurement.
12     """
13     if ouverts[s]:
14         return False
15     if not fermes[s]:
16         ouverts[s] = True
17         for i in range(n):
18             if graphe[s][i] == 1:
19                 val = mystere(graphe, i, n, ouverts,
       fermes, resultat)
20             if not val:
21                 return False
22             ouverts[s] = False
23             fermes[s] = True
24             # ...
25     return True
```

8 En utilisant la matrice M donnée précédemment, déterminer si la variable `ok` vaut `True` ou `False` à l'issue des instructions suivantes :

```
n = len(M)
ouverts = [False for i in range(n)]
fermes = [False for i in range(n)]
ok = mystere(M, 1, n, ouverts, fermes, None)
```

INTERROS

Décrire précisément les appels effectués à la fonction `mystere` et les valeurs des tableaux `ouverts` et `fermes` lors de chaque appel. On pourra recopier et compléter le tableau ci-dessous.

Appel mystere	variable ouverts	variable fermes
Avant l'appel mystere	$[F,F,F,F,F]$	$[F,F,F,F,F]$
mystere(M, 1,5, $[F,F,F,F,F]$,	$[F,F,F,F,F]$	$[F,F,F,F,F]$
mystere(M, 2,5, $[F,T,F,F,F]$,	$[F,T,T,F,F]$	$[F,F,F,F,F]$
...		

9 De manière générale, expliquer dans quel cas cette fonction `mystere` renvoie `False`.

L'objectif est d'utiliser la fonction `mystere` pour écrire une fonction `ordre_realisation` qui, lorsque c'est possible, détermine l'ordre de réalisation des tâches d'un graphe donné par sa matrice d'adjacence en respectant les dépendances entre les tâches.

Une structure de données de pile est représentée par une classe `Pile` qui possède les méthodes suivantes :

- la méthode `estvide` qui renvoie `True` si la pile représentée par l'objet est vide, `False` sinon ;
- la méthode `empiler` qui prend en paramètre un élément et l'ajoute au sommet de la pile ;
- la méthode `depiler` qui renvoie la valeur du sommet de la pile et enlève cet élément.

10 Déterminer la valeur associée à la variable `elt` après l'exécution des instructions suivantes :

```
essai = Pile()
essai.empiler(3)
essai.empiler(2)
essai.empiler(10)
elt = essai.depiler()
elt = essai.depiler()
```

Lorsqu'il en existe un, un ordre de réalisation des tâches sera représenté par un objet de classe `Pile` contenant tous les sommets du graphe de manière à ce que les tâches qu'il faut réaliser en premier se retrouvent au sommet de la pile.

La fonction `ordre_realisation` est écrite de la manière suivante :

```
def ordre_realisation(graphe):
    n = len(graphe)
    ouverts = [False for i in range(n)]
    fermes = [False for i in range(n)]
    ordre = Pile()
    ok = True
    s = 0
    while (ok and s < n):
        ok = mystere(graphe, s, n, ouverts, fermes,
    ordre)
        s = s + 1
    if ok:
        return ordre
    return None
```

11 Sachant que dans la fonction `mystere`, la ligne 24 peut être remplacée par une ou plusieurs instructions, donner ce qu'il faut écrire pour que, lorsque c'est possible, `ordre_realisation` renvoie effectivement un ordre de réalisation des tâches du graphe.

CORRIGÉS

1 QCM Vérification de connaissances

Énoncé p. 150

1. Réponse b. L'ordre d'un graphe est le nombre de sommets qu'il comporte. Donc ici, c'est 8.
2. Réponse b. Le graphe est en effet non orienté car aucun sens n'est spécifié pour passer d'un sommet à un autre.
3. Réponse b. Le graphe n'est en effet pas cyclique car il n'existe aucun chemin reliant le sommet D, par exemple, à lui-même. Il en est de même pour le sommet G.
4. Réponse a. Le graphe est en effet connexe car il n'existe pas de sommets isolés (non reliés à au moins l'un des autres sommets).
5. Réponse b. La seule chose qui change entre les deux matrices proposées est la diagonale principale, qui ne comporte que des « 1 » pour la 1re et que des « 0 » pour la 2nde. Dans la mesure où le graphe ne comporte pas de boucle, il ne doit pas y avoir de « 1 » sur cette diagonale. La première matrice ne convient donc pas.

2 Le plan de ville

Énoncé p. 151

Lycée André Theuriet, Civray

Il n'y a rien de bien compliqué dans ce que l'on nous demande ici. Le travail est juste long...

D'abord, on doit repérer tous les sommets :

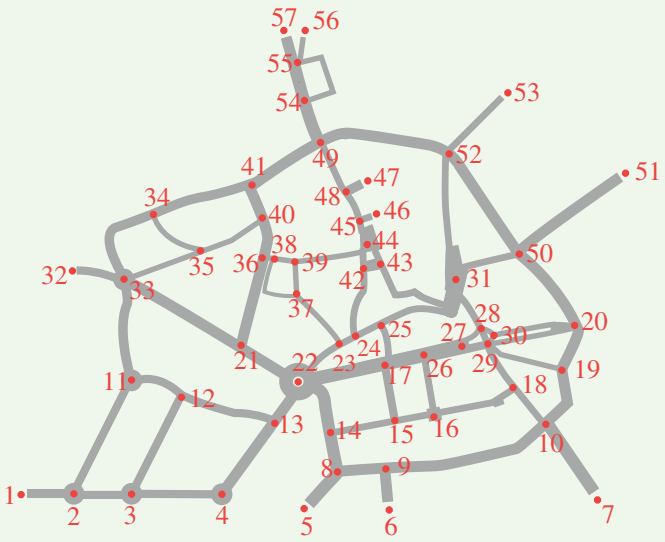


Figure 5.14 – Marquage des intersections

Nous avons ici nommé les sommets par des nombres car il y en a plus de 26, mais nous aurions pu les nommer S_1, S_2, etc.

Ensuite, nous simplifions le plan en mettant les arêtes :

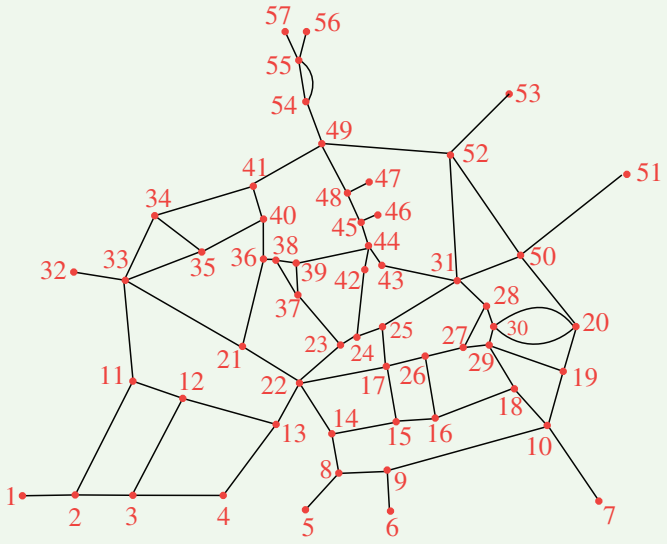


Figure 5.15 – Ajout des arêtes

On peut dire de ce graphe qu'il est :

- connexe (car aucun sommet n'est isolé) ;
- acyclique (il n'y a pas de cycle pour le sommet 1 par exemple) ;
- les sommets 30 et 20, ainsi que les sommets 54 et 55, sont reliés par deux arêtes et ce sont les seuls sommets dans ce cas.

Pour ce qui est d'un chemin pour rejoindre la sortie, nullement besoin d'un programme : on peut voir que le chemin :

```
1 - 2 - 11 - 33 - 34 - 41 - 49 - 54 - 55 - 57
```

convient, avec seulement 8 intersections entre les sommets 1 et 57.

Ce n'est pas le seul car on peut remplacer 34 par 35, mais ce sont les deux seuls chemins qui répondent à notre exigence.

3 La maison piégée

Énoncé p. 151

Lycée Jean Macé, Niort

1 Il semble assez évident de représenter ici les pièces par des sommets et les couloirs par des arêtes. La seule question éventuellement épineuse est de savoir comment représenter les pièces piégées. Dans la mesure où l'on souhaite un graphe le plus simple possible, et comme il ne faut pas aller dans ces pièces, il suffit de les ignorer : le graphe ne va donc pas contenir de sommets représentant les pièces piégées.

De plus, le graphe est non orienté car nullement question de sens dans les déplacements à l'intérieur de cette maison.

Avant de construire le graphe, enlevons donc les pièces piégées.

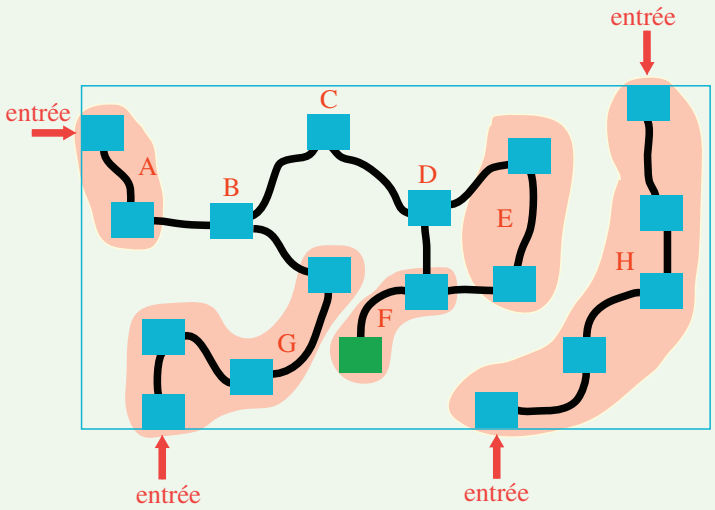


Figure 5.16 – Carte simplifiée de la maison

Nous voyons alors que l'on peut encore simplifier le problème car plusieurs pièces se trouvent sur un même chemin non ramifié, pièces que l'on a réunies sur la figure 5.18 en orange en « groupes » : A, E, G, F et H.

L'idée ici est de regrouper les pièces n'ayant qu'une entrée et une sortie : par exemple, la pièce B comporte trois connexions, donc elle ne fait pas partie d'un groupe. La pièce C comporte bien deux connexions mais aucune de ses deux pièces adjacentes (B et D) n'est, comme elle, limitée à deux connexions. C ne peut donc pas faire partie d'un groupe car elle ne peut pas en constituer un avec ses voisins.

Le problème peut donc se représenter par le graphe suivant :

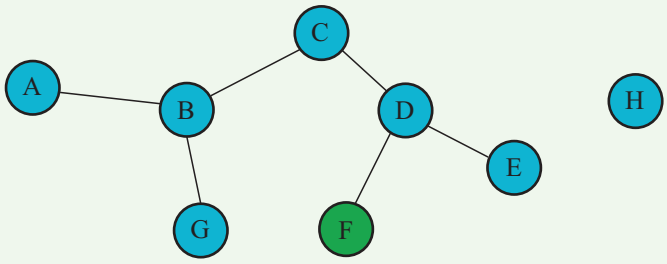


Figure 5.17 – Graphe représentant la maison

2 On déduit du graphe précédent que le chemin le plus court pourrait être :

A – B – C – D – F ou G – B – C – D – F

mais en regardant sur le schéma original de la maison, seul le chemin A – B – C – D – F convient, ce qui correspond au chemin page ci-contre dans la maison.

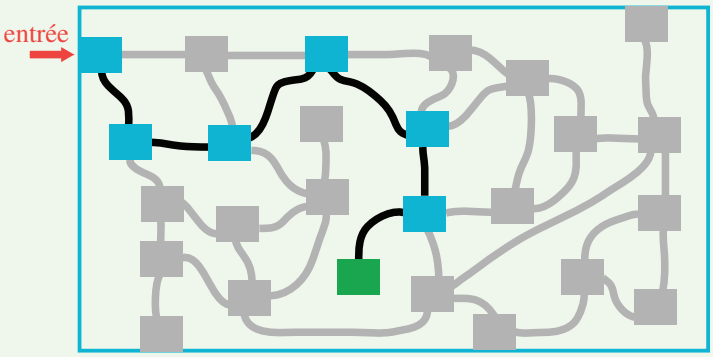


Figure 5.18 – Chemin le plus court

4 Le loup et compagnie...

Énoncé p. 152

Lycée François Villon, Les Mureaux

1 **MÉTHODE**

Dans ce genre de problème, il est bon d'énumérer tous les états qui font intervenir « P » :

« PLCX » , « PLC » , « PLX » , « PCX » et « PC ».

et sans « P » (en tenant compte des contraintes) :

« LX » , « L » , « C » , « X » et « vide ».

Ce dernier état signifie que la rive est vide, et donc que le problème est résolu.

Le graphe doit traduire les différents déplacements possibles entre chaque état. On peut alors avoir une représentation sagittale qui ressemble à la suivante :

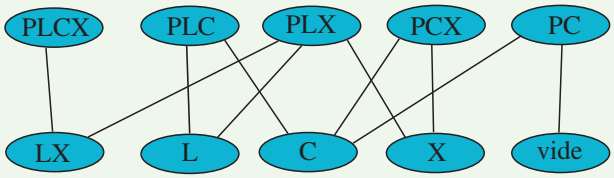


Figure 5.19 – Représentation du problème

2 Résoudre le problème revient alors à trouver un chemin du sommet « PLCX » au sommet « vide ». On obtient alors, par exemple :

PLCX → LX → PLX → X → PCX → C → PC → vide.

Cela signifie que :

- au départ, le passeur amène la chèvre sur la rive de destination, laissant le loup et le chou sur la rive de départ ;
- le passeur revient et se retrouve avec le loup et le chou ;
- il amène le loup sur la rive de destination, laissant le chou ;
- il ramène la chèvre sur la rive de départ car elle ne peut rester avec le loup sur la rive de destination ;
- il amène le chou sur la rive de destination, laissant la chèvre seule ;
- il revient seul car le loup et le chou peuvent rester ensemble sur la rive de destination ;
- il se retrouve donc avec la chèvre sur la rive de départ, et l'amène sur l'autre rive.

5 Algorithme de Dijkstra

Énoncé p. 153

Lycée Jean-Joseph Fourier, Auxerre

1 On a le tableau suivant :

M	L[A]	L[B]	L[C]	L[D]	P[A]	P[B]	P[C]	P[D]
A	0	1	2	∞	-	A	A	-
A,B	0	1	2	3	-	A	B	B
A,B,C,D	0	-3	2	3	-	C	A	B

2 La distance obtenue de A à D est égale à 3 avec l'algorithme de Dijkstra. Or, nous voyons bien que le chemin A – C – B – D a un poids total de $2 - 5 + 2 = -1$ et que ce poids est minimal.

La raison de ce « bug » est que le sommet C a été traité après le sommet B ; or, le poids de C à B étant négatif, L[B] a été mis à jour. L'algorithme de Dijkstra ne traitant qu'une seule fois un sommet, cela n'a pas pu entraîner la mise à jour de L[D].

C'est le problème de cet algorithme quand il y a des poids négatifs.

6 Le politicien

Énoncé p. 153

Lycée Henri Alain-Fournier, Bourges

On utilise l'algorithme de Dijkstra en indiquant chaque étape dans une ligne du tableau page suivante.

Dans la colonne « H », on voit qu'il y a deux chemins qui mènent à ce sommet ; on prend celui qui a la plus petite pondération.

La durée minimale est donc de 29 minutes avec le trajet :

$$P \to B \to D \to G \to H.$$

Remarque : notons que le « 11 (P) » a été reporté et non le « 11 (A) » car il a été considéré avant. C'est une règle générale : on reporte toujours ce qui a été considéré en premier.

P	A	B	C	D	E	F	G	H
×	4 (P)	6 (P)	11 (P)	12 (P)				
×	4 (P)	7 (A)	11 (A)					
×	×	6 (P)		10 (B)				
×	×	×		10 (B)		15 (D)	19 (D)	
×	×	×	11 (P)	×	22 (C)		21 (C)	
×	×	×	×	×		15 (D)	23 (F)	
×	×	×	×	×		×	19 (D)	
×	×	×	×	×		×	×	29 (G)
×	×	×	×	×	22 (C)	×	×	31 (E)

7 Matrice d'adjacence aléatoire

Énoncé p. 154

Lycée Sophie Germain, Paris

MÉTHODE

« aléatoire » implique souvent « appel au module random ». De plus, on exploite le fait qu'une matrice d'adjacence est carrée, ne comporte que des « 0 » et des « 1 », que sa diagonale principale est remplie de 0 et qu'elle est symétrique par rapport à la diagonale principale.

Une fonction possible est alors celle donnée page suivante.

L'idée est donc ici d'initialiser une matrice de dimensions $n \times n$ remplie par exemple de « 0 ».

Ensuite, pour chaque ligne (`for ligne in range(n)`), on parcourt chaque colonne : si on est au-dessus de la diagonale principale (`if colonne > ligne`) alors on choisit un coefficient au hasard entre 0 et 1. Si on est au contraire en dessous, le coefficient doit être égal à son symétrique par rapport à la diagonale principale (`M[ligne][colonne] = M[colonne][ligne]`).

Inutile de faire le cas où `ligne == colonne` car dans ce cas, le coefficient doit être nul, ce qui est déjà le cas par construction initiale de la matrice.

```
from random import randint

def matadj(n):
    M = [ [0 for j in range(n)] for i in range(n) ]
    for ligne in range(n):
        for colonne in range(n):
            if colonne > ligne:
                M[ligne][colonne] = randint(0,1)
            elif colonne < ligne:
                M[ligne][colonne] = M[colonne][ligne]

    return M
```

8 Par matrice d'adjacence

Énoncé p. 154

Lycée Camille Jullian, Bordeaux

On pourrait implémenter ce graphe « à la main » à l'aide de la classe `Graphe` donnée page 139, en énumérant toutes les arêtes grâce à la matrice d'adjacence donnée, mais si on pouvait implémenter une façon de passer de la matrice d'adjacence à une liste d'adjacence, ce serait mieux.

Le programme page suivante répond à cette requête.

Il affiche :

```
B : ['C', 'D', 'F']
A : ['B', 'C', 'E']
C : ['D', 'E', 'F']
E : ['F']
D : ['E']
F : []
```

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{pmatrix} \textcircled{0} & \textcircled{1} & \textcircled{1} & \textcircled{0} & \textcircled{1} & \textcircled{0} \\ 1 & \textcircled{0} & \textcircled{1} & \textcircled{1} & \textcircled{0} & \textcircled{1} \\ 1 & 0 & \textcircled{0} & \textcircled{1} & \textcircled{1} & \textcircled{1} \\ 0 & 1 & 1 & \textcircled{0} & \textcircled{1} & \textcircled{0} \\ 1 & 0 & 1 & 1 & \textcircled{0} & \textcircled{1} \\ 0 & 1 & 1 & 0 & 1 & \textcircled{0} \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix}$$

En effet, une arête est créée entre les sommets i et j si le coefficient m_{ij} est égal à 1. Il suffit donc de faire deux boucles, sur i et j (où i représente la ligne et j la colonne de la matrice) et d'effectuer un test de valeur pour savoir si on doit créer une arête ou non.

De plus, on sait que la matrice d'adjacence est toujours symétrique par rapport à sa diagonale principale. Il est donc inutile d'effectuer ces tests pour les coefficients m_{ij} avec $j < i$.

```
class Graphe:
    def __init__(self):
        self.Liste = {}

    def addArete(self, u, v):
        if v not in self.Liste:
            self.Liste[v] = []

        if u not in self.Liste:
            self.Liste[u] = []

        self.Liste[u].append(v)

    def affiche(self,M, sommets):
        for i in range(len(M)):
            for j in range(i, len(M)):
                if M[i][j] == 1:
                    self.addArete(sommets[i], sommets[j
    ])

        for key,value in self.Liste.items():
            print(key, ":", value)

M = [
        [ 0, 1, 1, 0, 1, 0 ],
        [ 1, 0, 1, 1, 0, 1 ],
        [ 1, 0, 0, 1, 1, 1 ],
        [ 0, 1, 1, 0, 1, 0 ],
        [ 1, 0, 1, 1, 0, 1 ],
        [ 0, 1, 1, 0, 1, 0 ]
    ]

G = Graphe()
sommets = [ 'A', 'B', 'C', 'D', 'E', 'F' ]

G.affiche(M, sommets)
```

9 Flux migratoires

Énoncé p. 154

Lycée polyvalent Émile Zola, Aix-en-Provence

1 Dans la mesure où certains habitants déménagent vers une autre région, nous allons représenter les flux migratoires par un graphe à quatre sommets : X, Y, Z et A (où A représentera tout ce qui n'est pas dans la région).

De plus, le graphe doit être orienté (car le mouvement X → Y n'est pas similaire au mouvement Y → X) et pondéré (par les pourcentages, mis en décimaux par exemple). Il ne faut pas oublier les pourcentages non indiqués dans l'énoncé, à savoir ceux correspondants aux habitants qui restent dans leur département.

On a alors la représentation sagittale suivante :

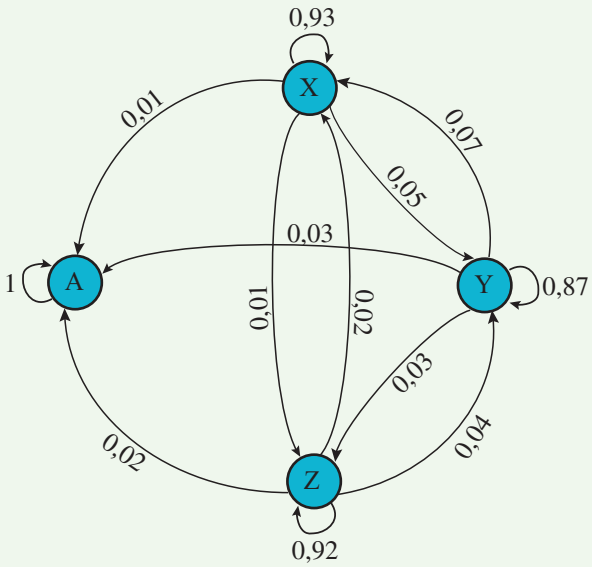


Figure 5.20 – Représentation des flux migratoires

2 La matrice de transition de ce graphe est :

$$M = \begin{array}{c} \begin{matrix} X & Y & Z & A \end{matrix} \\ \begin{pmatrix} 0{,}93 & 0{,}05 & 0{,}01 & 0{,}01 \\ 0{,}07 & 0{,}87 & 0{,}03 & 0{,}03 \\ 0{,}02 & 0{,}04 & 0{,}92 & 0{,}02 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} X \\ Y \\ Z \\ A \end{matrix}$$

3 On implémente la matrice de la manière suivante :

```
>>> from numpy import *
>>> M = array( [
  [0.93,0.05,0.01,0.01] ,
  [0.07,0.87,0.03,0.03] ,
  [0.02,0.04,0.92,0.02] ,
  [0,0,0,1]
  ] )
```

4 On obtient M^{10} en tapant :

```
>>> from numpy.linalg import matrix_power
>>> matrix_power(M,10)
array([[0.56932088, 0.21942888, 0.08467249,
0.12657775],
  [0.30805016, 0.33744598, 0.13505622, 0.21944764],
  [0.16509636, 0.18149117, 0.46765393, 0.18575854],
  [0. , 0. , 0. , 1. ]])
```

5 On peut écrire :

```
>>> P = array( [ 1.081 , 1.076 , 0.327 , 64.516] )
>>> dot( P , matrix_power(M,10) )
array([ 1.00088435, 0.65964211, 0.38977429,
64.94969926])
```

On obtient ainsi :

$$x_{10} \approx 1{,}000\,884 \quad , \quad y_{10} \approx 0{,}659\,642 \quad , \quad z_{10} \approx 0{,}389\,774.$$

6 Une simple boucle suffit :

```
from numpy import array, dot
from numpy.linalg import matrix_power

P = array([ 1.081, 1.076, 0.327, 64.516])

M = array([
    [0.93,0.05,0.01,0.01],
    [0.07,0.87,0.03,0.03],
    [0.02,0.04,0.92,0.02],
    [0,0,0,1]
    ])

for n in range(1,201):
    print(dot(P, matrix_power(M,n)))
```

On s'aperçoit alors que les nombres d'habitants des départements X, Y et Z ne font que décroitre, ce qui nous laisse à penser que ces départements se dépeuplent et qu'à long terme, ils seront désertés si aucun plan n'est adopté pour remédier à cette migration.

CORRIGÉS

10 Une simple histoire de temps

Énoncé p. 155

Lycée Sainte-Clotilde, Strasbourg

- Commençons par représenter les changements climatiques à l'aide d'un graphe :

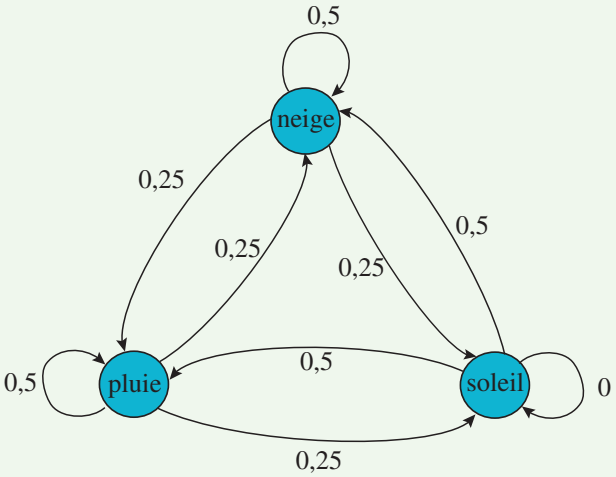


Figure 5.21 – Représentation des changements climatiques

- La matrice de transition de ce graphe est :

$$M = \begin{array}{c} \begin{matrix} N & P & S \end{matrix} \\ \begin{pmatrix} 0{,}5 & 0{,}25 & 0{,}25 \\ 0{,}25 & 0{,}5 & 0{,}25 \\ 0{,}5 & 0{,}5 & 0 \end{pmatrix} \end{array} \begin{matrix} N \\ P \\ S \end{matrix}$$

- On implémente alors la matrice de la manière suivante :

```
>>> from numpy import *
>>> M = array( [
  [0.5,0.25,0.25] ,
  [0.25,0.5,0.25] ,
  [0.5,0.5,0] ,   ] )
```

- On calcule par exemple M^{100} :

```
>>> from numpy.linalg import matrix_power
>>> matrix_power(M,100)
array([[0.4, 0.4, 0.2],
  [0.4, 0.4, 0.2],
  [0.4, 0.4, 0.2]])
```

Cette dernière matrice nous dit alors que la probabilité qu'il fasse beau à long terme est égale à 0,2 (par rapport à l'ordre que nous avons choisi, on regarde la dernière colonne), soit un jour sur cinq.

11 Le labyrinthe

Énoncé p. 156

Lycée Camille Jullian, Bordeaux

Il existe des méthodes plus économes que celle souhaitée par l'énoncé, comme par exemple la fusion de cases pour réduire le nombre de sommets du graphe représentant le labyrinthe, mais contentons-nous pour l'instant de la méthode la plus évidente : à chaque case son sommet. Notons $S_{i,j}$ le sommet de la case sur la i-ième ligne et j-ième colonne.

Si on fait un zoom sur l'entrée, cela donne :

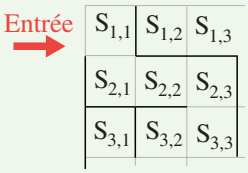


Figure 5.22 – Zoom sur l'entrée

On marque par une arête la présence d'un passage entre 2 cases adjacentes. Par exemple : il n'y a pas d'arête entre $S_{1,1}$ et $S_{1,2}$ mais il y en a une entre $S_{1,1}$ et $S_{2,1}$.

En tout, il aura $11 \times 11 = 121$ sommets. Le graphe est creux car il y aura très peu d'arêtes par rapport au nombre de sommets. Ceci nous pousse à implémenter le graphe par liste d'adjacence.

Téléchargez l'implémentation complète.

À l'aide de ce programme, on voit que le plus court chemin est le suivant, de longueur 36 (nombre d'arêtes) :

```
'S1-1', 'S2-1', 'S2-2', 'S2-3', 'S3-3', 'S4-3', 'S4-4',
'S4-5', 'S5-5', 'S5-4', 'S6-4', 'S6-5', 'S6-6', 'S7-6',
'S7-7', 'S8-7', 'S8-6', 'S8-5', 'S9-5', 'S9-4', 'S9-3',
  'S9-2', 'S9-1', 'S10-1', 'S11-1', 'S11-2', 'S11-3',
 'S11-4', 'S11-5', 'S11-6', 'S11-7', 'S11-8', 'S10-8',
        'S10-9', 'S11-9', 'S11-10', 'S11-11'
```

Cela correspond au chemin suivant :

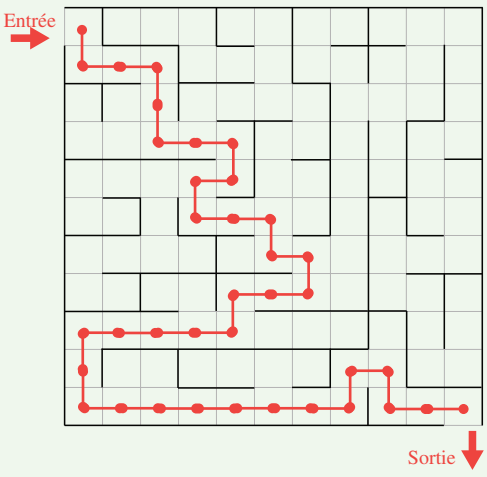


Figure 5.23 – Chemin le plus court

CORRIGÉS

12 Recette

Énoncé p. 156

Lycée Richelieu, Versailles

1 Ce graphe est orienté car le passage d'un nœud à l'autre a un sens.

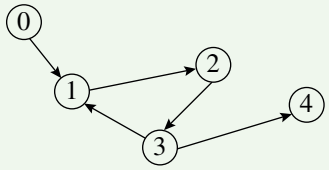
2 **(a)** Réaliser la tâche (f) puis la tâche (g) : oui.
(b) Réaliser la tâche (g) puis la tâche (f) : non.
(c) Réaliser la tâche (i) puis la tâche (j) : oui car il n'y a pas de dépendance entre les deux nœuds.
(d) Réaliser la tâche (j) puis la tâche (i) : oui, pour les mêmes raisons qu'à la question précédente.

3 Il faut nécessairement avoir réalisé les tâches (a), (c), (h), (i) et (j).

4 Le graphe ne contient pas de cycle.

5 • Graphe de gauche : 0, 2 puis 1. • Graphe de droite : 3, 5 puis 4.

6 Le graphe associé à la matrice d'adjacence donnée est le suivant :



7 Il est impossible de trouver un ordre permettant de réaliser les tâches représentées par le graphe de la question 6 en respectant leurs dépendances car un cycle y est présent.

8 Faisons un tableau d'exécution de la fonction `mystere` :

`Appel mystere`	variable `ouverts`	variable `fermes`
`Avant l'appel mystere`	[F, F, F, F, F]	[F, F, F, F, F]
`mystere(M, 1, 5, [F, F, F, F, F], [F, F, F, F, F], None)`	[F, T, F, F, F]	[F, F, F, F, F]
`mystere(M, 2, 5, [F, T, F, F, F], [F, F, F, F, F], None)`	[F, T, T, F, F]	[F, F, F, F, F]
`mystere(M, 3, 5, [F, T, T, F, F], [F, F, F, F, F], None)`	[F, T, T, T, F]	[F, F, F, F, F]
`mystere(M, 1, 5, [F, T, T, T, F], [F, F, F, F, F], None)`	[F, T, T, T, T]	[F, F, F, F, F]

Ainsi, `ok = False` car au niveau du dernier appel récursif, `ouverts[1]` est déjà égal à `T` ; la fonction renvoie donc `False`.

9 La fonction renvoie `False` quand le graphe comporte un cycle.

10 Après l'exécution des instructions, `elt = 2`.

11 À la ligne 24 de la fonction `mystère`, on peut insérer :

```
resultat.empiler(s)
```

Ainsi, lorsque cela est possible, `ordre_realisation` renvoie effectivement un ordre de réalisation des tâches du graphe.

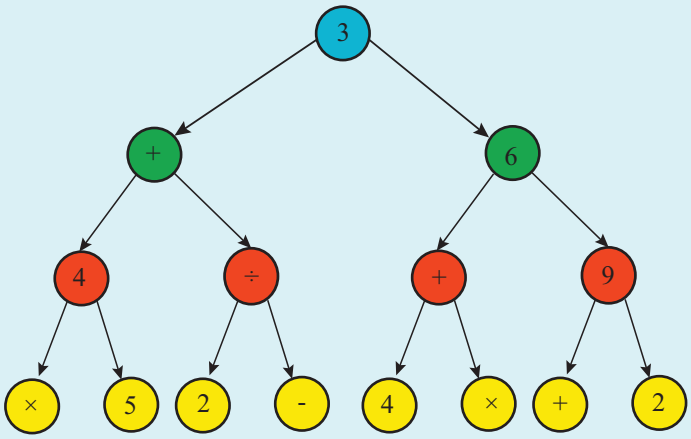
Chapitre 6

Les arbres : structures hiérarchiques

Plan du chapitre

Exercice type — **Lycée Camille Claudel, Blois**

On considère l'arbre suivant :



On parcourt cet arbre en profondeur avec un ordre préfixe.

1 Quel est le résultat de l'opération obtenue si l'on tient compte des priorités opératoires, c'est-à-dire du fait que la multiplication et la division sont prioritaires sur l'addition et la soustraction ?

2 Implémenter cet arbre et retrouver le résultat de la question précédente à l'aide d'un programme écrit en Python.

Voir corrigé page 189

COURS

Les listes et tableaux sont des structures de données peu pratiques à manipuler dès lors que l'on souhaite effectuer des recherches sur leurs valeurs, ou même ajouter et supprimer des valeurs.

PROBLÉMATIQUE

Comment représenter un ensemble ordonné de clés de sorte à effectuer simplement des opérations élémentaires ?

1 Notion d'arbres

Définition 1

En informatique, un *arbre* est est un graphe acyclique orienté connexe dont tout sommet est sommet d'arrivée d'au plus une arête.

Définitions 2 : vocabulaire

- Le seul sommet n'étant l'arrivée d'aucune arête d'un arbre est la *racine*.
- Les arêtes d'un arbre sont aussi appelées *branches*.
- Les sommets sont appelés les *nœuds*.
- Les sommets étant sommets d'arrivée d'une arête dont l'origine est le nœud N sont appelés *sommets issus* de N.
- Les nœuds issus d'un nœud N sont appelés les *fils* de N. N est alors appelé le *père*.
- Les nœuds d'un même niveau sont appelés les *nœuds frères*.
- Les nœuds n'ayant pas de fils sont aussi appelés des *feuilles*.
- La *distance* d'un nœud à un autre est le nombre de branches reliant l'un à l'autre.
- La *profondeur* d'un nœud est la distance de la racine à ce nœud.
- La *hauteur* de l'arbre est la plus grande distance de la racine à une feuille.
- La *taille* de l'arbre est le nombre de nœuds.
- Une *clé* est la valeur d'un nœud.

Remarque : dans cet ouvrage, la convention utilisée est celle-ci : la hauteur d'un arbre vide est égale à -1. Il existe cependant d'autres conventions (par exemple, hauteur de l'arbre vide égale à 0).

Exemple : on a représenté ci-dessous un arbre de hauteur 2 et de taille 9.

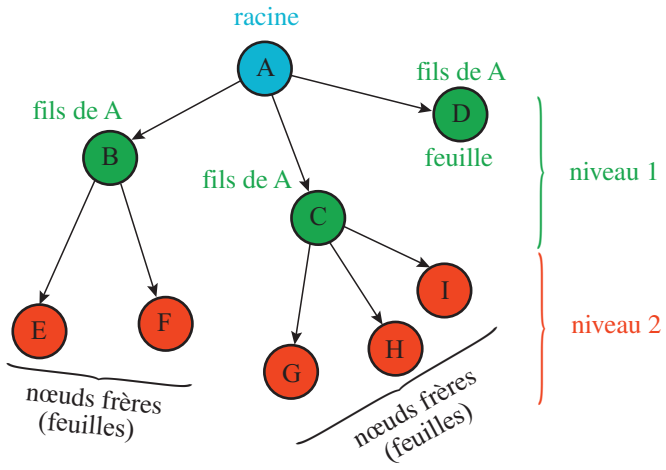


Figure 6.1 – Représentation sagittale d'un arbre

On peut représenter un arbre par une liste commençant par la racine et suivie d'une suite de listes représentant les différents chemins de l'arbre.

Exemple : reprenons l'arbre de l'exemple précédent en ajoutant un autre niveau :

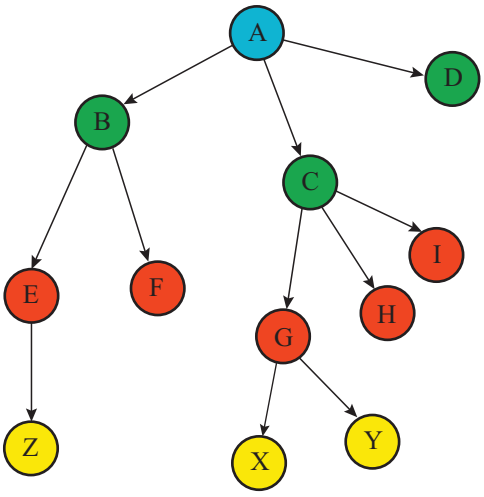


Figure 6.2 – Arbre à trois niveaux

Cet arbre pourra alors être implémenté par la liste :

```
[ 'A' ,
  ['B',
    ['E' ,
      ['Z']
    ] ,
    ['F']
  ],
  ['C' ,
    ['G' ,
      ['X'] ,
      ['Y']
    ] ,
    ['H'] ,
    ['I']
  ],
['D']
]
```

COURS

2 Arbre binaire

2.1 Définitions

Définitions 3

Un arbre est dit *binaire* si chaque nœud admet au plus deux fils, que l'on appelle *fils gauche* et *fils droit*.
Si, pour tout nœud de l'arbre, les hauteurs des sous-arbres gauche et droit diffèrent d'au plus 1, l'arbre est un *arbre AVL*, du nom de Georgy Adelson-Velsky et Evgenii Landis, qui l'ont inventé en 1962.

Définition 4 : sous-arbres gauche et droit

Soit T un arbre binaire. On appelle *sous-arbre gauche* (resp. *droit*) d'un nœud l'arbre issu du fils gauche (resp. droit) de ce nœud.

Exemple : on a représenté ci-dessous un arbre binaire de hauteur 3 et de taille 9.

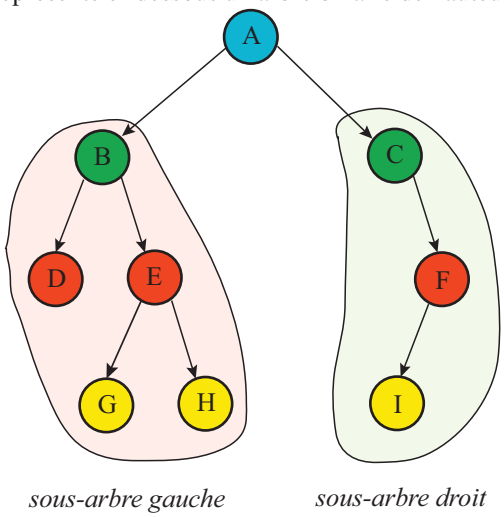


Figure 6.3 – Représentation des sous-arbres gauche et droit de la racine

2.2 Propriétés sur les arbres binaires

Propriété 1

Un arbre binaire de hauteur h admet au plus $1 + 2 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$ nœuds.

Propriétés 2

- Un arbre binaire ayant n nœuds a une hauteur h telle que :
$$\log_2(n+1) - 1 \leqslant h \leqslant n$$
c'est-à-dire :
$$\frac{\ln(n+1)}{\ln 2} - 1 \leqslant h \leqslant n.$$
- Tout arbre binaire ayant f feuilles a une hauteur h telle que :
$$h \geqslant \log_2(f).$$
- Soit un arbre binaire ayant f feuilles et n nœuds. Alors,
$$\log_2(f) \leqslant \log_2\left(\frac{n+1}{2}\right).$$

2.3 Arbre Binaire de Recherche (ABR)

Retrouvez en vidéo la notion d'arbre binaire de recherche.

Définition 5

Un *arbre binaire de recherche*, en abrégé : ABR, est un arbre vide ou un arbre binaire dont :

- tous les nœuds sont étiquetés par des nombres ;
- la racine est supérieure ou égale à tout nœud du sous-arbre gauche ;
- la racine est inférieure à tout nœud du sous-arbre droit ;
- les deux sous-arbres de la racine sont eux-mêmes des ABR.

Exemple : l'abre binaire ci-dessous est un ABR.

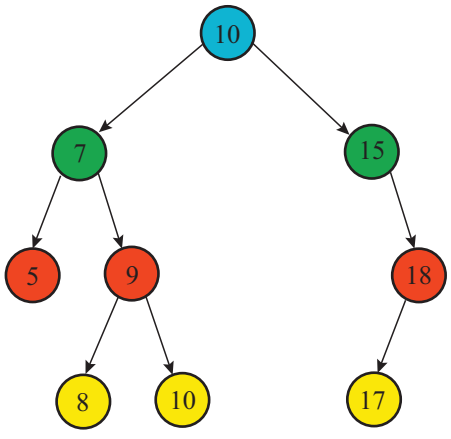


Figure 6.4 – Représentation d'un ABR

COURS

3 Représentations des arbres binaires

3.1 Représentation contigüe

On peut représenter un arbre binaire à l'aide d'un tableau (une liste) à une dimension.

Si L est cette liste :

- `L[0]` est la racine ;
- `L[1]` est le fils gauche de la racine (notons-le R.g) ;
- `L[2]` est le fils droit de la racine (notons-le R.d) ;
- `L[3]` est le fils gauche de R.g ;
- `L[4]` est le fils droit de R.g ;
- `L[5]` est le fils gauche de R.d ;
- `L[6]` est le fils droit de R.d ;
- etc.

Exemple : la représentation de l'arbre de la figure 2.1 page 178 est donnée ci-dessous.

Rang	0	1	2	3	4	5	6	7	8	9	10	11	12
Valeur	A	B	C	D	E		F			G	H	I	

```
L = [A , B , C , D , E , , F , , , G , H , I , ]
```

3.2 Représentation avec une classe

3.2.1 - Première possibilité

Nous pouvons représenter un arbre binaire à l'aide d'une classe `node` (par exemple) contenant trois champs :

- la valeur du nœud ;
- le parent du nœud ;
- la position du nœud par rapport à son parent (gauche ou droit)

puis d'une classe `tree` contenant une liste des nœuds, dont le premier élément est la racine.

Si nous implémentons en Python l'arbre de la figure 2.1, cela donne :

```
class Node:
    def __init__(self , data , parent , position):
        self.data = data
        self.parent = parent
        self.position = position
```

```
class Tree:
    def __init__(self , liste):
        self.racine = liste[0].data
        self.noeud = liste[1:]

A = Node("A" , None , None)
B = Node("B" , "A" , "Gauche")
C = Node("C" , "A" , "Droit")
D = Node("D" , "B" , "Gauche")
E = Node("E" , "B" , "Droit")
F = Node("F" , "C" , "Droit")
G = Node("G" , "E" , "Gauche")
H = Node("H" , "E" , "Droit")
I = Node("I" , "F" , "Gauche")

arbre = Tree([A,B,C,D,E,F,G,H,I])

print( arbre.noeud[2].data ,
        "est le fils" ,
        arbre.noeud[2].position ,
        "de" , arbre.noeud[2].parent)
```

qui affiche : `D est le fils Gauche de B`

3.2.2 - Autre possibilité

Nous pouvons utiliser la récursivité implicitement liée aux arbres binaires. Ainsi, pour l'arbre de la figure 2.1), on peut définir de manière simple une classe `Arbre` comme dans le code donné page suivante.

```
class ArbreBinaire:
    def __init__(self, racine, filsGauche, filsDroit):
        self.racine = racine
        self.filsGauche = filsGauche
        self.filsDroit = filsDroit

    def affiche(self, space = 0):
        spaces = "  " * space
        print(spaces,self.racine)
        if self.filsGauche is not None:
            self.filsGauche.Affiche(space+1)
        if self.filsDroit is not None:
            self.filsDroit.Affiche(space+1)
```

COURS

```
# ArbreBinaire(racine, fils gauche, fils droit)
t1 = ArbreBinaire('B',
                  ArbreBinaire('D',None,None),
                  ArbreBinaire('E',
                               ArbreBinaire('G',None,None),
                               ArbreBinaire('H',None,None)
                               )
                  )

t2 = ArbreBinaire('C',
                  ArbreBinaire('-',None,None),
                  ArbreBinaire('F',
                               ArbreBinaire('I',None,None),
                               ArbreBinaire('-',None,None)
                               )
                  )

tree = ArbreBinaire('A', t1, t2)

tree.affiche()
```

Ce programme affiche :

```
A
   B
      D
      E
         G
         H
   C
      -
      F
         I
         -
```

4 Algorithmique

4.1 Calcul de la taille d'un arbre binaire

Ce calcul se fait à l'aide d'un algorithme récursif.

```
fonction taille(arbre):
   Si arbre est vide:
     retourner 0
   Sinon:
     G ← taille(gauche(arbre))
     D ← taille(droit(arbre))
     return 1 + G + D
```

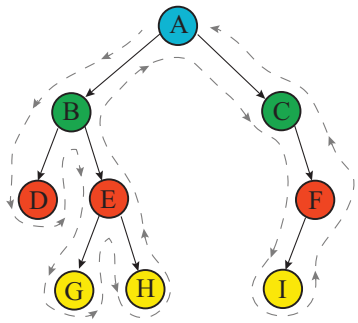
Ici, `gauche(arbre)` représente le sous-arbre gauche du nœud sur lequel nous sommes et `droit(arbre)` son sous-arbre droit.

4.2 Calcul de la hauteur d'un arbre binaire

On utilise ici le même type d'algorithme que pour le calcul de la taille, en convenant de prendre -1 pour hauteur d'un arbre vide (où seule la racine existe).

```
fonction hauteur(arbre):
   Si arbre est vide:
     retourner -1
   Sinon:
     G ← hauteur(gauche(arbre))
     D ← hauteur(droit(arbre))
     return 1 + max(G,D)
```

4.3 Parcours en profondeur d'un arbre binaire



Parcourir un arbre binaire signifie accéder à l'information contenue dans les nœuds.
Nous allons donc imaginer que l'on parcourt l'arbre de la façon indiquée ci-contre.

Figure 6.5 – Parcours d'un arbre binaire

Ce parcours est appelé *parcours en profondeur* de l'arbre.

Il existe trois façons d'énumérer les nœuds.

COURS

4.3.1 - Ordre préfixe

On liste les nœuds dans l'ordre dans lequel on les rencontre pour la première fois.

Exemple : l'arbre de la figure 6.5 donne :

A – B – D – E – G – H – C – F – I

Algorithme (récursif) de parcours préfixe

```
ParcoursPrefixe(arbre):
    Afficher la clé de l'arbre
    ParcoursPrefixe(gauche(arbre))
    ParcoursPrefixe(droit(arbre))
```

4.3.2 - Ordre postfixe

On liste les nœuds dans l'ordre dans lequel on les rencontre pour la dernière fois.

Exemple : l'arbre de la figure 6.5 donne :

D – G – H – E – B – I – F – C – A

Algorithme (récursif) de parcours postfixe

```
ParcoursPostfixe(arbre):
    ParcoursPostfixe(gauche(arbre))
    ParcoursPostfixe(droit(arbre))
    Afficher la clé de l'arbre
```

4.3.3 - Ordre infixe

On liste les nœuds selon la règle suivante :

- chaque nœud sans fils gauche est répertorié la première fois qu'on le rencontre ;
- chaque nœud comportant un fils gauche est répertorié la seconde fois qu'on le rencontre ;

Exemple : pour l'arbre de la figure 6.5, nous allons avant tout faire un tableau.

Sommets	Fils gauche	Répertorié la...	Parcours
A	Oui	2nde fois	
B	Oui	2nde fois	
D	Non	1re fois	D
B	Oui	2nde fois	D – B
E	Oui	2nde fois	D – B
G	Non	1re fois	D – B – G
E	Oui	2nde fois	D – B – G – E
H	Non	1re fois	D – B – G – E – H
E	Oui	2nde fois	D – B – G – E – H
B	Oui	2nde fois	D – B – G – E – H
A	Oui	2nde fois	D – B – G – E – H – A
C	Non	1re fois	D – B – G – E – H – A – C
F	Oui	2nde fois	D – B – G – E – H – A – C
I	Non	1re fois	D – B – G – E – H – A – C – I
F	Oui	2nde fois	D – B – G – E – H – A – C – I – F
C	Non	1re fois	D – B – G – E – H – A – C – I – F
A	Oui	2nde fois	D – B – G – E – H – A – C – I – F

Algorithme (récursif) de parcours infixe

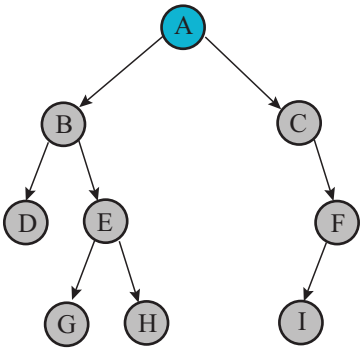
```
ParcoursInfixe(arbre):
    ParcoursInfixe(gauche(arbre))
    Afficher la clé de l'arbre
    ParcoursInfixe(droit(arbre))
```

4.4 Parcours en largeur

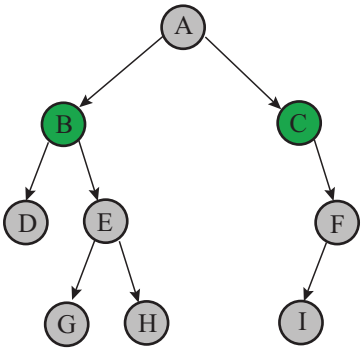
Reprenons l'arbre de la figure 2.1 page 178.

L'idée consiste à lister chaque nœud-fils niveau par niveau.
Nous avons présenté page suivante les différentes étapes.

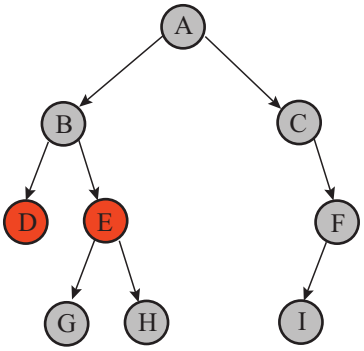
COURS



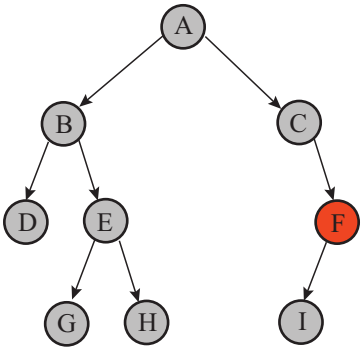
Étape 1



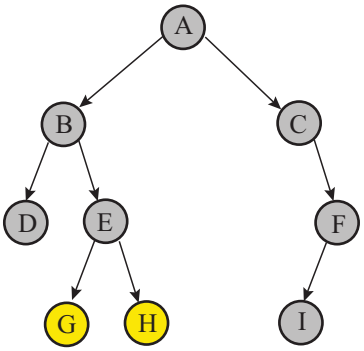
Étape 2



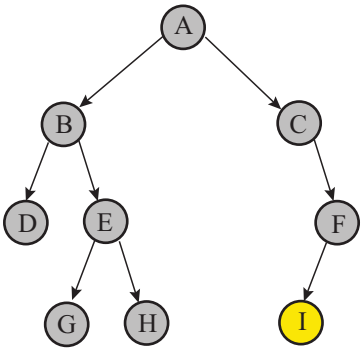
Étape 3



Étape 4



Étape 5



Étape 6

L'algorithme de parcours en largeur est basé sur une file.

Algorithme de parcours en largeur

```
F est une file vide
P est une liste vide
Enfiler racine à F
Ajouter racine à P
Tant que F non vide:
   U ← tête de F
   Pour V fils de U:
     Enfiler V à F
     Ajouter V à P
   Fin du Pour
   Défiler U de F
Fin du Tant que
```

4.5 Recherche dans un ABR

La structure d'un ABR facilite grandement les recherches, et c'est la raison pour laquelle cette structure de données est intéressante. En effet, on sait par définition que chaque clé de nœuds-fils-gauches est inférieure à celle du nœud courant et que chaque clé de nœuds-fils-droits est supérieure. Cela donne l'algorithme suivant :

Algorithme (récursif) de recherche dans un ABR

```
Recherche(ABR,val):
   Si ABR est vide:
     Retourner Faux
   Sinon:
     r ← racine de ABR
     Si val = r:
       Retourner ABR
     Sinon:
       Si val ⩽ r:
         Retourner Recherche(gauche(ABR),val)
       Sinon:
         Retourner Recherche(droite(ABR),val)
       Fin du Si val ⩽ r
     Fin du Si val = r
   Fin du Si ABR est vide
```

Propriété 3

Le coût d'une recherche dans un ABR est en $O(h)$, où h est la hauteur de l'arbre.

COURS

4.6 Insertion dans un ABR

Pour insérer une clé dans un ABR, il suffit de faire une recherche et de faire l'insertion à l'endroit où la recherche est infructueuse.

Algorithme d'insertion dans un ABR

```
Insere(ABR,clé):
   Si ABR est vide:
     return CreerABR(clé)
   Sinon:
     Si clé ⩽ racine(ABR):
       G ← Insere(gauche(ABR),clé)
       gauche(ABR) ← G
       Retourner ABR
     Sinon:
       D ← Insere(droit(ABR),clé)
       droit(ABR) ← D
       Retourner ABR
     Fin du Si clé ⩽ racine(ABR)
   Fin du Si ABR est vide
```

Remarque : la fonction `CreerABR` est une fonction qui crée un arbre à partir d'une valeur ; quant à `gauche(ABR)` et `droit(ABR)`, ce sont deux fonctions qui retournent respectivement les fils gauche et droit de ABR.

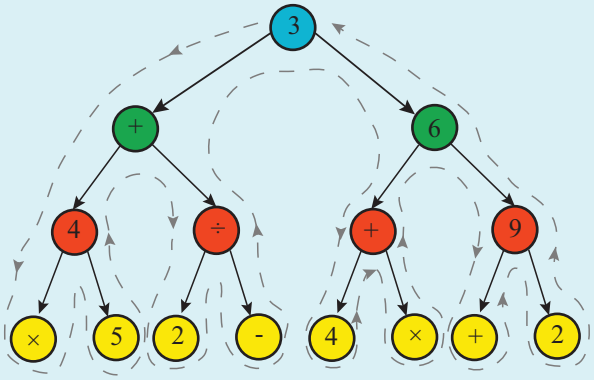
À RETENIR

Arbre binaire	Chaque nœud admet au plus deux fils : un fils gauche et un fils droit.
Arbre AVL	Pour tout nœud de l'arbre, les hauteurs des sous-arbre gauche et droit diffèrent d'au plus 1.
Arbre Binaire de Recherche (ABR)	Arbre vide ou arbre étiqueté par des nombres où : • la racine est supérieure ou égale à tout nœud du sous-arbre gauche ; • la racine est inférieure à tout nœud du sous-arbre droit ; • les deux sous-arbres de la racine sont eux-mêmes des ABR.

Solution de l'exercice type

Lycée Camille Claudel, Blois

1 Le parcours en profondeur est le parcours suivant :



Il donne alors l'opération : $3 + 4 \times 5 \div 2 - 6 + 4 \times 9 + 2 = 45$.

2 Nous pouvons nous inspirer de la classe `Arbre` de la page 181 et de sa méthode `Affiche` (qui permet de parcourir en profondeur l'arbre avec un ordre préfixe). On obtient ainsi le programme suivant :

```
class Arbre:
    def __init__(self,racine,filsGauche,filsDroit):
        self.root = racine
        self.G = filsGauche
        self.D = filsDroit

    def parcours(self , L = []):
        L.append(self.root)
        if self.G:
            self.G.parcours(L)
        if self.D:
            self.D.parcours(L)
        return L
```

Solution de l'exercice type (suite)

Lycée Camille Claudel, Blois

```
tree = Arbre(3,
        Arbre(
            '+',
            Arbre( 4 , Arbre('*',None,None) , Arbre
    (5,None,None) ),
            Arbre( '/' , Arbre(2,None,None) , Arbre
    ('-',None,None) )
        ),
        Arbre(
            6,
            Arbre( '+' , Arbre(4,None,None) , Arbre
    ('*',None,None) ),
            Arbre( 9 , Arbre('+',None,None) , Arbre
    (2,None,None) )
        ) )

L = tree.parcours()
calcul = ''.join(str(i) for i in L)
print ( eval(calcul) )
```

Une fois l'arbre défini, on construit une liste `L` à l'aide de la méthode `parcours`, dans laquelle on ajoute tous les nœuds de l'arbre dans l'ordre voulu. La liste `L` est alors transformée en chaîne de caractères (avec la méthode `join`) afin de pouvoir l'interpréter comme un calcul (avec la fonction `eval`).

Voir énoncé page 175

1 V/F Vérification de connaissances

5 min

Corrigé p. 202

Les affirmations suivantes sont-elles vraies ou fausses ? Justifier la réponse.

1. Le nombre maximal de nœuds d'un arbre binaire de hauteur h est $2^h - 1$.
2. L'arbre binaire suivant est un ABR.

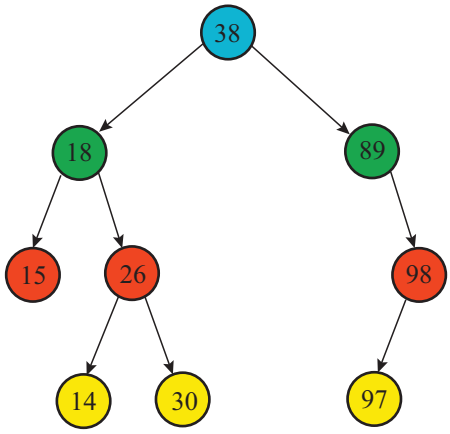


Figure 6.6 – Arbre binaire

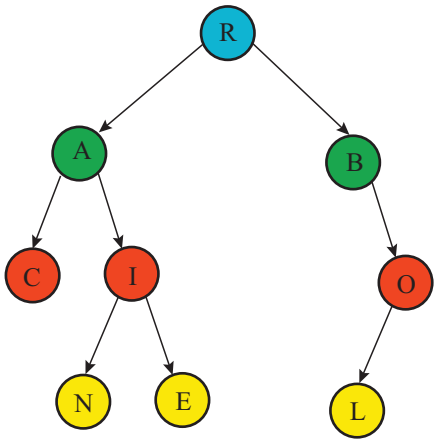


Figure 6.7 – Arbre binaire

3. Le parcours en profondeur par ordre préfixe de l'arbre binaire de la figure 6.7 donne :

R – A – C – I – N – E – B – O – L

4. Le parcours en profondeur par ordre postfixe de l'arbre binaire de la figure 6.7 donne :

C – N – E – I – A – L – O – B – R

5. Le parcours en profondeur par ordre infixe de l'arbre binaire de la figure 6.7 donne :

C – A – N – E – I – B – O – L – R

INTERROS

6 Le parcours en largeur de l'arbre binaire de la figure 6.7 donne :

R – A – B – C – I – O – N – E – L

2 Arbres de Fibonacci

★ 5 min Corrigé p. 203

Lycée Henri Alain-Fournier, Bourges

On définit un *arbre de Fibonacci* comme étant un arbre A_n tel que les sous-arbres gauche et droit de A_n sont respectivement A_{n-1} et A_{n-2}, en admettant que A_0 est un arbre avec une unique racine et A_1 est composé d'une racine ayant un fils gauche et un fils droit.

1 Donner une représentation sagittale de l'arbre A_3.

2 Les arbres de Fibonacci sont-ils des AVL ? Justifier.

3 Écrire une fonction d'affichage

★ 10 min Corrigé p. 204

Lycée Raoul Follereau, Nevers

On considère un arbre implémenté comme ci-dessous :

```
arbre = [
'A' ,
['B', ['E'] , ['F'] ],
['C' , ['G'] , ['H'] , ['I'] ],
['D']
]
```

Écrire une fonction `afficher(arbre,marge)`, où `marge` est un nombre entier, qui affiche l'arbre sous une forme plus facile à lire que cette expression : chaque nœud doit être sur une ligne, avec une marge à gauche qui reflète la position hiérarchique (la *profondeur*) du nœud.

4 Arbre binaire complet

★ 10 min Corrigé p. 205

Lycée Sainte-Clotilde, Strasbourg

Un arbre binaire complet est un arbre binaire dans lequel tous les nœuds qui ne sont pas des feuilles ont exactement deux fils.

Dans ce cas, il peut être représenté par une liste de longueur $2^{h+1} - 1$, où h est sa hauteur, où les fils gauche et droit de l'élément d'indice i sont stockés en indices $2i + 1$ et $2i + 2$. Les feuilles manquantes peuvent être représentées par des éléments vides ou des éléments `None` par exemple.

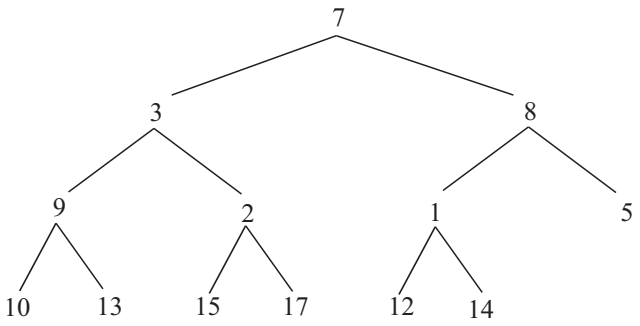


Figure 6.8 – Un exemple d'arbre binaire complet

1. Quelle est, en Python, la liste représentant l'arbre de la figure 6.8 ?
2. Quelle est la formule permettant de calculer la hauteur d'un arbre binaire complet en fonction de la longueur de la liste des nœuds ?
3. Écrire alors une fonction `affiche(arbre)` qui affiche l'arbre en fonction de la liste `arbre` en utilisant un affichage similaire à la figure 6.8 (sans les traits).

5 Labyrinthe et arbre binaire

★★ 10 min Corrigé p. 206

Lycée Sainte-Clotilde, Strasbourg Lycée Jean Macé, Niort

On considère le labyrinthe représenté ci-dessous :

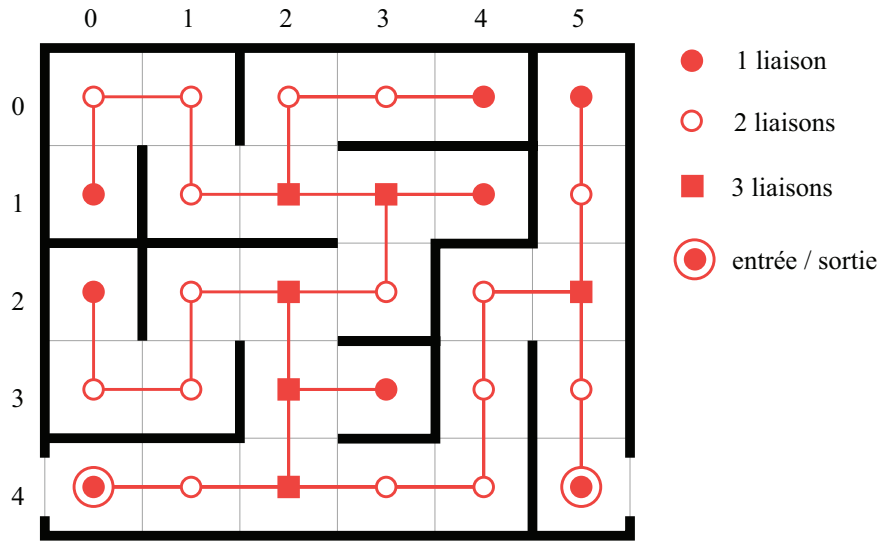


Figure 6.9 – Source : wikipédia

Construire un arbre binaire représentant ce labyrinthe dans lequel chaque case est représentée par un nœud. On partira du nœud noté (4,0) et chaque nœud sera noté (i, j) où i et j représentent respectivement la ligne et la colonne de la case correspondante.

Construction de classes

6 Arborescence d'un répertoire

★★ 20 min Corrigé p. 207

Lycée Jean-Joseph Fourier, Auxerre

Dans cet exercice, on souhaite créer une classe nous permettant d'afficher l'arborescence d'un répertoire.

On s'aide alors de la classe suivante :

```
class Node:
  def __init__(self,nodeValue):
    self.subNodes = []
    self.value = nodeValue

  def addSubNode(self,node):
    self.subNodes.append(node)

  def view(self,depth=0):
    if self.subNodes:
      print("{}[{}]".format("  " * depth, self.value))
    else:
      print("  " * depth, self.value)
    depth += 1
    for node in self.subNodes:
            node.view(depth)
```

1. Expliquez la façon dont est défini le constructeur de la classe `Node`.
2. Expliquez les deux méthodes de cette classe.
3. En faisant appel au module `os`, écrire une fonction admettant comme argument le chemin d'un répertoire, et qui retourne un arbre représentant l'architecture du répertoire informé en argument, arbre que l'on pourra afficher à l'aide de la méthode `view` de la classe `Node`. On pourra nommer cette fonction : `getDirectoryNode(path)` .

7 Arbres binaires

★★★ 45 min Corrigé p. 208

Lycée Henri Bergson, Angers

Dans cet exercice, on souhaite construire une classe `ArbreBinaire`, initialisée à partir d'une liste.

On se basera sur l'arbre suivant :

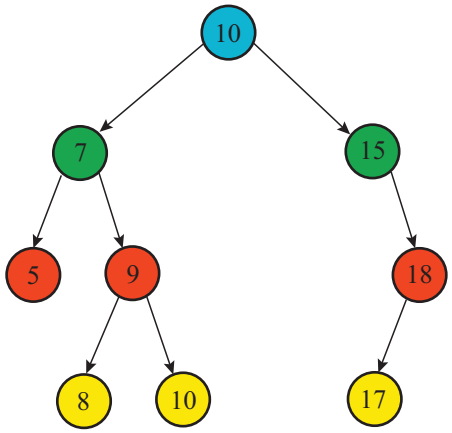


Figure 6.10 – Arbre binaire servant d'exemple

On souhaite que cet arbre soit implémenté par la liste suivante :

```
L =
[ 10 ,
   [ 7 ,
     [5 , [] , []] ,
     [9 , [8 , [] , []] , [10 , [] , []] ]
   ] ,
  [ 15 ,
     [ ],
     [ 18 , [17 , [] , []] , [ ] ]
   ]
]
```

1. Écrire un constructeur de cette classe ainsi qu'une méthode `arbor` de sorte à avoir l'affichage illustré sur la figure 6.11 (voir page suivante).
2. Écrire une méthode `infixe` permettant de parcourir l'arbre avec un ordre infixe et de retourner la liste des nœuds selon cet ordre.
3. Déduire de la question précédente une méthode permettant de vérifier si l'arbre binaire est un ABR.
4. Écrire une méthode `min_max` retournant un couple composé du minimum et du maximum des étiquettes de l'arbre.

5 Écrire une méthode `hauteur` qui retourne la hauteur de l'arbre, puis en déduire une méthode `isAVL` qui permet de vérifier si l'arbre est un arbre AVL.

```
>>> A = ArbreBinaire(L)
>>> A.arbor()
10
   7
     5
       -
       -
     9
       8
         -
         -
       10
         -
         -
   15
     -
     18
       17
         -
         -
       -
```

Figure 6.11 – Illustration de l'affichage souhaité

8 Une classe « Arbre » complète

★★★ 45 min Corrigé p. 211

Lycée Camille Jullian, Bordeaux

Dans cet exercice, on se propose de construire une classe `Arbre` relativement complète en se basant sur ce qui a été vu en cours, notamment les algorithmes. Cette classe admet le constructeur suivant :

```
class Arbre:
    def __init__(self,racine,*enfants):
        self.root = racine
        self.enfants = enfants
```

Nous souhaitons implémenter un arbre comme celui de la page suivante (en fin d'énoncé).

1 Implémenter une méthode `Prefixe` qui retourne une liste donnant tous les nœuds de l'arbre dans un ordre préfixe, à l'aide d'un parcours en profondeur.

2. Implémenter une méthode `Postfixe` qui retourne une liste donnant tous les nœuds de l'arbre dans un ordre postfixe, à l'aide d'un parcours en profondeur.

3. Implémenter une méthode `Largeur` qui retourne une liste donnant tous les nœuds de l'arbre en ayant parcouru ce dernier en largeur.

4. Implémenter alors une méthode `AfficheListe(chaine)` de sorte à avoir :

```
>>> tree.AfficheListe('prefixe')
['A', 'B', 'C', 'D', 'Z', 'E', 'X', 'Y', 'P', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
>>> tree.AfficheListe('postfixe')
['Z', 'D', 'X', 'Y', 'P', 'E', 'C', 'G', 'H', 'F', 'B',
'K','L', 'J', 'N', 'O', 'M', 'I', 'A']
>>> tree.AfficheListe('largeur')
['A', 'B', 'I', 'C', 'F', 'J', 'M', 'D', 'E', 'G', 'H',
'K','L', 'N', 'O', 'Z', 'X', 'Y', 'P']
```

5. Implémenter une méthode `Affiche` qui permet d'afficher l'arbre avec une marge, comme dans le paragraphe 3.2.2 page 181.

```
tree =
Arbre('A',
    Arbre('B',
        Arbre('C',
            Arbre('D', Arbre('Z')),
            Arbre('E',
                Arbre('X'), Arbre('Y'), Arbre('P'))),
        Arbre('F', Arbre('G'), Arbre('H'))),
    Arbre('I',
        Arbre('J', Arbre('K'), Arbre('L')),
        Arbre('M', Arbre('N'), Arbre('O'))))
```

9 Une classe pour ABR

★★★ 60 min Corrigé p. 212

Lycée polyvalent Émile Zola, Aix-en-Provence

Dans cet exercice, on se propose de construire une classe pour représenter des arbres binaires de recherche.

On part alors du constructeur suivant :

```
class ABR:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.parent = None
```

où :

- `data` représente la valeur d'un nœud ;
- `left` et `right` désignent respectivement les fils gauche et droit ;
- `parent` est le parent du nœud.

Ainsi, par défaut, un nœud n'a ni fils ni parent.

On initialiser un ABR en écrivant :

```
A = ABR(150)
```

où « 150 » est la valeur de la racine de l'arbre.

Pour tester les méthodes que nous allons implémenter, nous représenterons l'arbre suivant :

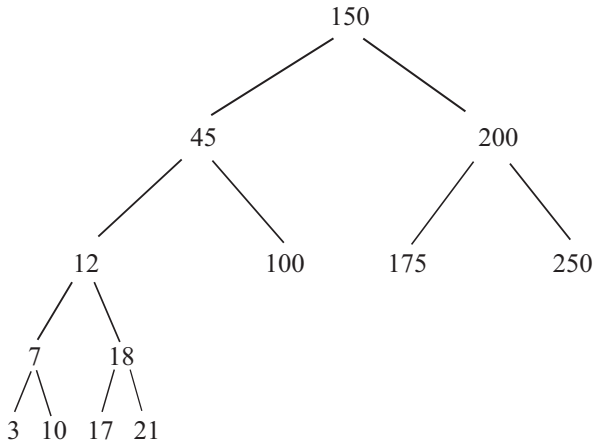


Figure 6.12 – ABR à implémenter

1 Écrire une méthode `insert` qui permet d'insérer une valeur dans l'ABR.

Suite des questions page suivante.

2 Écrire une méthode `affiche` permettant d'afficher l'arbre horizontalement, c'est-à-dire que l'arbre de la figure 6.12 s'affichera sous la forme :

```
                250
        200
                175
150
                100
        45
                        21
                18
                        17
                12
                        10
                7
                        3
```

3 Écrire une méthode `hauteur` qui renvoie la hauteur de l'arbre.

4 Écrire une méthode `trouve(data)` qui retourne le nœud qui contient la valeur `data`, et qui retourne « `None` » si la valeur n'est pas trouvée dans l'ABR.

5 Écrire une méthode `liste` qui renvoie une liste ordonnée des valeurs contenues dans l'arbre.

6 Écrire deux méthodes `is_left_child` et `is_right_child` qui vérifient si un nœud est le fils gauche ou droit de son parent, ou pas. Ces méthodes doivent renvoyer un booléen.

7 Écrire une méthode `minimum` qui retourne le nœud ayant la plus petite valeur.

8 Le successeur d'un nœud N est le nœud ayant la plus petite valeur supérieure à N.

Écrire une méthode `successeur(self)` qui retourne le successeur d'un nœud.

9 Écrire enfin une méthode `delete` qui permet de supprimer le nœud dont la valeur est un nombre passé en argument.

On pourra s'aider des méthodes `trouve` et `successeur`.

Objectif bac

10 Stockage d'identifiants

★★★ 45 min Corrigé p. 217

Lycée Richelieu, Versailles

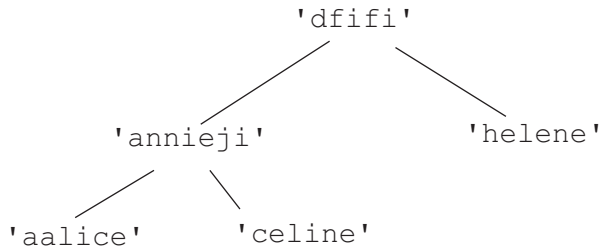
Une entreprise stocke les identifiants de ses clients dans un arbre binaire de recherche. On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit. La taille d'un arbre est le nombre de nœuds qu'il contient.

INTERROS

Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœuds racine à l'une des feuilles. On convient que la hauteur d'un arbre ne contenant qu'un nœuds vaut 1 et celle de l'arbre vide vaut 0. Dans cet arbre binaire de recherche, chaque nœuds contient une valeur, ici une chaine de caractères, qui est, avec l'ordre lexicographique (celui du dictionnaire) :

- strictement supérieure à toutes les valeurs des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les valeurs des nœuds du sous-arbre droit.

Ainsi les valeurs de cet arbre sont toutes distinctes. On considère l'arbre binaire de recherche suivant :



1 Donner la taille et la hauteur de l'arbre binaire de recherche.

2 Recopier cet arbre après l'ajout des identifiants suivants : '`davidbg`' et '`papicoeur`' dans cet ordre.

3 On décide de parcourir cet arbre pour obtenir la liste des identifiants dans l'ordre lexicographique.

Recopier la lettre correspondant au parcours à utiliser parmi les propositions suivantes :

a Parcours en largeur d'abord
b Parcours en profondeur dans l'ordre préfixe
c Parcours en profondeur dans l'ordre infixe
d Parcours en profondeur dans l'ordre suffixe (ou postfixe)

4 Pour traiter informatiquement les arbres binaires, nous allons utiliser une classe `ABR`.

Un arbre binaire de recherche, nommé `abr` dispose des méthodes suivantes :

- `abr.est_vide()` : renvoie `True` si `abr` est vide et `False` sinon.
- `abr.racine()` : renvoie l'élément situé à la racine de `abr` si `abr` n'est pas vide et `None` sinon.
- `abr.sg()` : renvoie le sous-arbre gauche de `abr` s'il existe et `None` sinon.
- `abr.sd()` : renvoie le sous-arbre droit de `abr` s'il existe et `None` sinon.

On a commencé à écrire une méthode récursive `present` de la classe ABR, où le paramètre identifiant est une chaine de caractères et qui retourne `True` si `identifiant` est dans l'arbre et `False` sinon.

```
def present(self, identifiant):
    # Si l'arbre est vide, retourner False
    if self.est_vide() is not None:
        return False

    # Si l'identifiant correspond à la racine,
    retourner True
    if self.racine() == identifiant:
        return ...

    # Si l'identifiant est supérieur à la racine,
    chercher dans le sous-arbre droit
    if self.racine() < identifiant:
        return ...

    # Si l'identifiant est inférieur à la racine,
    chercher dans le sous-arbre gauche
    if self.racine() > identifiant:
        return ...
```

Recopier et compléter cette méthode.

CORRIGÉS

1 V/F Vérification de connaissances

Énoncé p. 191

1 *Faux*. Il y a en effet au maximum :

- 1 nœud à la racine, niveau 0 ;
- 2 nœuds au niveau suivant, niveau 1 ;
- 2^2 nœuds au niveau 2 ;
- etc. jusqu'à 2^h nœuds au niveau h.

D'après la formule $1 + q + q^2 + \cdots + q^n = \dfrac{q^{n+1} - 1}{q - 1}$, avec $q = 2$ et $n = h$, on obtient au maximum $2^{h+1} - 1$ nœuds.

2 *Faux*. En effet, pour qu'un arbre soit binaire de recherche, il est nécessaire que la clé de tout nœud du sous-arbre droit issu d'un nœud N_k soit supérieure à la clé de N_k. Or, « 14 » est la clé d'un nœud du sous-arbre droit issu du nœud de clé « 18 » et $14 < 18$.

3 *Vrai*. Le parcours en profondeur par ordre préfixe donne les nœuds dans l'ordre dans lesquels on les rencontre pour la première fois :

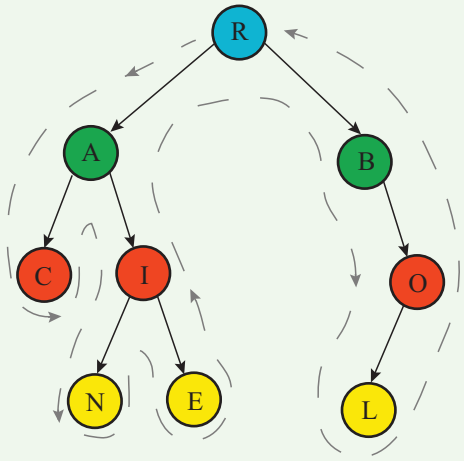


Figure 6.13 – Parcours en profondeur de l'arbre

Ce qui donne :

R – A – C – I – N – E – B – O – L.

À RETENIR

PRÉfixe = PREmière fois

4 *Vrai*. Quand on liste les nœuds dans l'ordre postfixe dans un parcours en profondeur, on les énumère dans l'ordre dans lequel on les rencontre pour la dernière fois. Cela donne bien :

C – N – E – I – A – L – O – B – R.

5 *Faux*. Quand on liste les nœuds dans l'ordre infixe dans un parcours en profondeur, si le nœud a un fils gauche, on le liste lors du deuxième passage sinon, lors du premier. Cela donne alors :

C – A – N – I – E – R – B – L – O.

6 *Vrai*. Dans un parcours en largeur, les nœuds sont listés par hauteur (par niveau) : on commence par la racine, on continue avec les nœuds-fils de hauteur 1, puis on continue avec ceux de hauteur 2, etc.

Cela donne bien :

R – A – B – C – I – O – N – E – L.

2 Arbres de Fibonacci

Énoncé p. 192

Lycée Henri Alain-Fournier, Bourges

1 Pour connaître la représentation sagittale de A_3, il nous faut avant tout obtenir A_2.

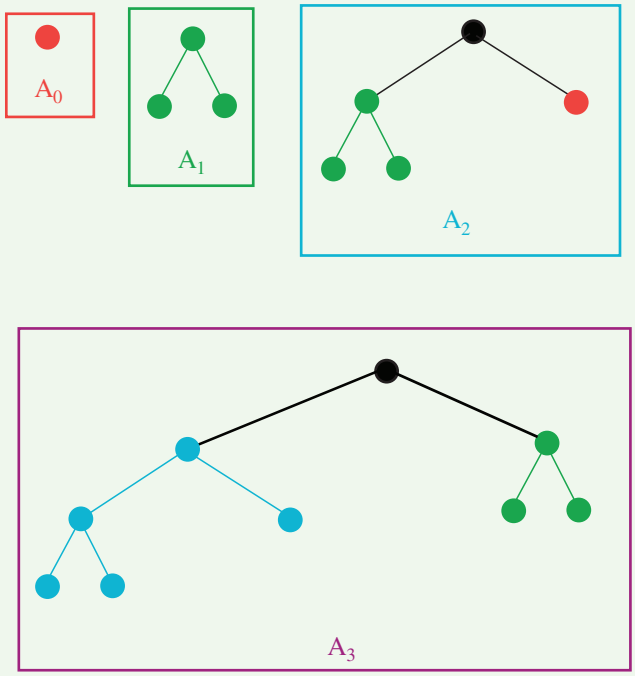


Figure 6.14 – Construction des premiers arbres de Fibonacci

2 Rappelons qu'un arbre AVL est un arbre dans lequel, pour tout nœud, les hauteurs des sous-arbres gauche et droit diffèrent d'au plus 1. Les arbres de Fibonacci sont donc des AVL. En effet, pour tout entier naturel k, A_k a une hauteur égale à k. Donc la différence de hauteur entre A_k et A_{k+1} est égale à 1. Or, A_n est composé de A_{n-1} (fils gauche) et A_{n-2} (fils droit), de hauteur $n-1$ et $n-2$, hauteurs différant donc de 1.

CORRIGÉS

3 Écrire une fonction d'affichage

Énoncé p. 192

Lycée Raoul Follereau, Nevers

Il s'agit ici de parcourir la liste `arbre` et, pour chaque élément de cette liste, d'afficher son premier élément avec une marge égale à la valeur de la variable `marge`, puis de parcourir les listes qui suivent et de procéder de même en multipliant la marge par n, où n est la position de la liste.

On voit alors une récursivité dans le traitement des informations.

Un programme possible est donc le suivant :

```
def afficher(arbre,marge=0):
    print(" " * marge + arbre[0])
    for subtree in arbre[1:]:
        afficher(subtree,marge+2)

arbre = [
  'A',
  ['B', ['E'], ['F'] ],
  ['C', ['G'], ['H'], ['I'] ],
  ['D']
]

afficher(arbre)
```

Ce programme affiche :

```
A
  B
    E
    F
  C
    G
    H
    I
  D
```

4 Arbre binaire complet

Énoncé p. 192

Lycée Sainte-Clotilde, Strasbourg

1 L'arbre de la figure 6.8 se représente par la liste :

```
[7,3,8,9,2,1,5,10,13,15,17,12,14,None,None]
```

Il suffit de lire les nœuds de gauche à droite, en descendant à partir de la racine.

2 Notons $n = 2^{h+1} - 1$ la longueur de la liste des nœuds.

Dans notre exemple, $15 = 2^4 - 1$ (il y a 15 éléments dans la liste, et la hauteur de l'arbre est égale à 3).

Alors,

$$\begin{aligned} n = 2^{h+1} - 1 &\iff 2^{h+1} = n + 1 \\ &\iff h = \log_2(n+1) - 1. \end{aligned}$$

3 Une implémentation possible est la suivante :

```
from math import log

def affiche(arbre):
    hauteur = int(log(len(arbre) + 1, 2))

    for i in range(hauteur):
        ligne = ''
        # ecartement entre les noeuds sur cette
    ligne
        ecart_ligne = (2**(hauteur-i+1) - 3) * ' '
        # écart au début
        ecart_debut= (2**(hauteur-i) - 2) * ' '
        for j in range(2**i - 1, min(2**(i+1) - 1,
    len(arbre))):
            ligne += "{0:^3}".format(str(arbre[j]))
            if j < min(2**(i+1) - 1, len(arbre)) -
    1:
                ligne += ecart_ligne
        print('\n'+ecart_debut+ligne)

arbre = [7,3,8,9,2,1,5,10,13,15,17,12,14,None,None]
affiche(arbre)
```

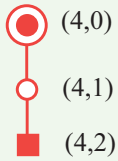
CORRIGÉS

5 Labyrinthe et arbre binaire

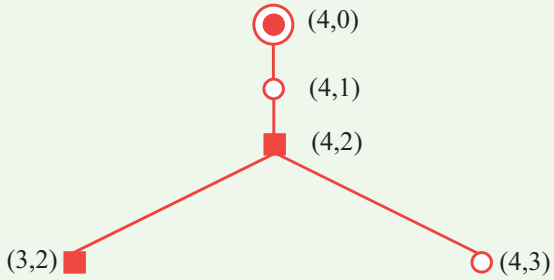
Énoncé p. 193

Lycée Sainte-Clotilde, Strasbourg Lycée Jean Macé, Niort

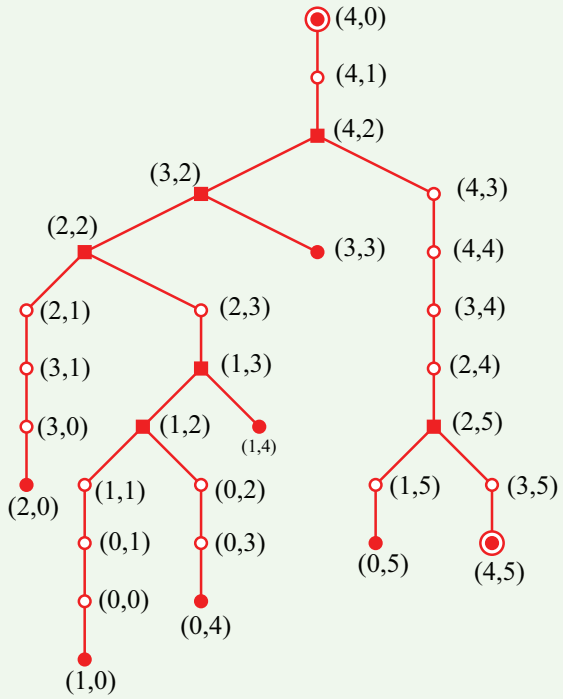
On part de la case (4,0) et nous avons un unique choix : aller à la case (4,1), et ensuite, il n'y a pas d'autre choix que d'aller en (4,2) :



Maintenant, nous avons le choix entre aller en (3,2) ou en (4,3) :



On continue ainsi jusqu'à chaque feuille pour obtenir l'arbre suivant :



6 Arborescence d'un répertoire

Énoncé p. 194

Lycée Jean-Joseph Fourier, Auxerre

1 Le constructeur de la classe `Node` permet d'initialiser une liste nommée `self.subNodes`, ainsi qu'une variable `self.value` prenant pour valeur celle de l'argument renseigné lors de l'appel à cette classe.

Les noms des variables étant explicites, on peut supposer que la liste contiendra les nœuds fils et que la variable `self.value` représente l'étiquette du nœud.

2 La méthode `addSubNode` admet un argument : `node`. A priori, elle ajoute à la liste `self.subNodes` la valeur de ce dernier argument.

Cette méthode a donc pour but d'ajouter à la liste des nœuds déjà existante un nœud supplémentaire.

Dans la définition de la méthode `view`, on commence par tester si la liste `self.subNodes` n'est pas vide, auquel cas on affiche la valeur du nœud entre crochets, avec une marge dépendant de sa profondeur dans l'arbre. Dans le cas contraire, si la liste est vide, on affiche simplement la valeur du nœud, sans crochet (pour signifier que c'est une feuille).

On augmente ensuite la profondeur et on passe aux éventuels enfants par récursivité : on fait appel à cette méthode en prenant cette fois-ci chaque nœud fils.

3 Voici une proposition de fonction répondant au cahier des charges :

```
def get_directory_node(path):
    node = Node(os.path.split(path)[1])
    if os.path.isdir(path):
        for itemName in os.listdir(path):
            fullPath = os.path.join(path,itemName)
            node.addSubNode(get_directory_node(
    fullPath))
    return node
```

Dans cette fonction, on commence par définir la racine de notre arbre : c'est le répertoire dont on veut trouver l'architecture. On prend soin ici de ne garder que le nom du répertoire et non le chemin complet, à l'aide de l'instruction `os.path.split(path)[1]`.

Le test qui suit nous sert à voir si le chemin désigne un répertoire (`if os.path.isdir(path)`), auquel cas on ajoute à l'arbre créé un sous-nœud pour chaque répertoire et fichier qu'il contient.

Ainsi, pour afficher l'arborescence d'un répertoire, on écrira par exemple :

```
>>> tree = getDirectoryNode("C:\\test_directory")
>>> tree.view()
```

```
[test_directory]
    [dir_1]
       [dir_11]
          file_11a.txt
          file_11b.txt
       [dir_12]
          [dir_123]
             fil_123a.txt
    [dir_2]
    ⋮
```

Téléchargez le programme complet.

7 Arbres binaires

Énoncé p. 195

Lycée Henri Bergson, Angers

1 Voici une proposition de script répondant au cahier des charges de cette première question :

```
class ArbreBinaire:
    def __init__(self, L):
        if L:
            self.root = L[0]
            self.filsGauche = ArbreBinaire(L[1])
            self.filsDroit = ArbreBinaire(L[2])
        else:
            self.root = None
            self.filsGauche = None
            self.filsDroit = None

    def arbor(self, space = 0):
        if self.root != None:
            print(" "*space,self.root)
            if self.filsGauche.root == None:
                spaces = " " * (space + 2)
                print(spaces,"-")
            else:
                self.filsGauche.arbor(space + 2)
            if self.filsDroit.root == None:
                spaces = " " * (space + 2)
                print(spaces,"-")
            else:
                self.filsDroit.arbor(space + 2)
```

L'idée consiste ici à construire un arbre binaire basé sur une liste de la forme `[racine , filsGauche , filsDroit]`, où `filsGauche` et `filsDroit` peuvent aussi être des arbres binaires.

2 Le parcours infixe de l'arbre peut-être obtenu à l'aide de la méthode suivante (inspirée de l'algorithme du cours page 185).

```
def infixe(self, L = []):
    if self.filsGauche:
        self.filsGauche.infixe(L)
    if self.root != None:
        L.append(self.root)
    if self.filsDroit:
        self.filsDroit.infixe(L)
    return L
```

3 Cette question pourrait être perçue comme très compliquée si nous ne remarquons pas que le parcours infixe d'un ABR donne toujours une liste croissante de nombres. On s'aide alors de la méthode `infixe` implémentée à la question précédente, ce qui donne :

```
def isABR(self):
    L = self.infixe()
    prev = None
    for node in L:
        if prev == None:
            prev = node
        elif node < prev:
            return False
    return True
```

4 **MÉTHODE**

Pour déterminer le minimum et le maximum d'un arbre (dans le cas où les étiquettes sont des nombres), il suffit de se baser sur la liste donnée par son parcours infixe : le minimum est le premier élément et le maximum, son dernier.

Cela donne la méthode suivante :

```
def min_max(self):
    L = self.infixe()
    return L[0], L[-1]
```

5 Pour calculer la hauteur d'un arbre, on ajoute 1 à la hauteur maximale des sous-arbres, ce qui donne :

```
def hauteur(self):
    # -2 si les sous-arbres n'existent pas
    hauteur_gauche = -2
    hauteur_droite = -2

    # Vérifier si le sous-arbre gauche existe
    if self.filsGauche is not None:
        hauteur_gauche = self.filsGauche.hauteur()

    # Vérifier si le sous-arbre droit existe
    if self.filsDroit is not None:
        hauteur_droite = self.filsDroit.hauteur()

    # Retourner la hauteur maximale+1
    return 1 + max(hauteur_gauche, hauteur_droite)
```

Pour vérifier si l'arbre est bien un AVL, il faut s'assurer que pour tout nœud, les hauteurs des sous-arbres diffèrent d'au plus 1. On va donc s'aider de la méthode `hauteur` :

```
def isAVL(self):
    if self.filsDroit is not None:
        hD =  self.filsDroit.hauteur()
    else:
        hD = 0
    if self.filsGauche is not None:
        hG = self.filsGauche.hauteur()
    else:
        hG = 0
    if abs(hD-hG) > 1:
        return False
    if self.filsDroit is not None:
        d = self.filsDroit.isAVL()
    if not d: return d
  if self.filsGauche is not None:
    g = self.filsGauche.isAVL()
    if not g: return g
  return True
```

On s'assure ici que l'arbre donné dans l'énoncé n'est pas un AVL.

Téléchargez la classe complète.

8 Une classe « Arbre » complète

Énoncé p. 196

Lycée Camille Jullian, Bordeaux

1 Voici une méthode `Prefixe` basée sur l'algorithme de la page 184 :

```
def prefixe(self, L = []):
    L.append(self.root)
    for child in self.enfants:
        child.prefixe(L)
    return L
```

2 Voici une méthode `Postfixe` basée sur l'algorithme de la page 184 :

```
def postfixe(self, L = []):
    for child in self.enfants:
        child.postfixe(L)
    L.append(self.root)
    return L
```

3 Voici une méthode `Largeur` basée sur l'algorithme de la page 187 :

```
def largeur(self):
    F = []
    L = []

    # Ajouter la racine et l'objet courant à L et F
     respectivement
    L.append(self.root)
    F.append(self)

    # Tant qu'il y a des éléments dans F
    while F != []:
        # Prendre le premier élément de F
        current = F.pop(0)

        # Ajouter tous les enfants à L et F
        for child in current.enfants:
            L.append(child.root)
            F.append(child)

    return L
```

4 Une méthode d'affichage sous forme de liste est alors la suivante :

```
def afficher_liste(self, ordre):
    if ordre == 'prefixe':
        print(self.prefixe())
    elif ordre == 'postfixe':
         print(self.postfixe())
    elif ordre == 'largeur':
        print(self.largeur())
```

5 En s'inspirant du script permettant l'affichage vu en page 181 du cours, on obtient la méthode suivante :

```
def affiche(self, space = 0):
    spaces = " " * space
    print(spaces, self.root)
    for child in self.enfants:
        child.affiche(space + 2)
```

Téléchargez la classe complète.

9 Une classe pour ABR

Énoncé p. 197

Lycée polyvalent Émile Zola, Aix-en-Provence

1 Quand on insère une valeur dans un ABR, il faut voir si celle-ci est inférieure ou égale à sa racine, auquel cas la valeur sera placée dans le sous-arbre gauche, ou au contraire supérieure, auquel cas elle sera placée dans le sous-arbre droit.

Si la valeur est inférieure ou égale à la valeur de la racine, on regarde s'il existe un fils gauche : si tel n'est pas le cas, on définit le fils gauche par la valeur à insérer, et on définit le parent comme étant la racine. Si, au contraire, le fils gauche existe, alors on répète l'opération en prenant pour racine le fils gauche : on fait donc appel à la récursivité.

Pour une valeur strictement supérieure à la valeur de la racine, on utilise un raisonnement analogue. On a alors la méthode écrite page ci-contre.

```
def insert(self, data):
    if data <= self.data:
        if self.left == None:
            self.left = ABR(data)
            self.left.parent = self
        else:
            self.left.insert(data)
    else:
        if self.right == None:
            self.right = ABR(data)
            self.right.parent = self
        else:
            self.right.insert(data)
```

2 Le nœud le plus à droite de l'arbre sera affiché sur la première ligne avec une marge dépendant de sa hauteur. À la ligne suivante devra s'afficher le parent du nœud précédemment affiché, et ensuite, le fils droit du parent.

Cette façon de raisonner nous pousse à utiliser la récursivité ; en effet, tant qu'il y a un fils droit, on fait appel à la méthode `affiche` en augmentant la hauteur de 1 ; quand il n'y en a plus, on affiche le nœud et on passe au fils gauche. D'où la méthode suivante :

```
def affiche(self, h = 0):
    if self.right:
        self.right.affiche(h + 1)
    spaces = ' ' * 7 * h
    print(spaces,self.data)
    if self.left:
        self.left.affiche(h + 1)
```

3 Pour calculer la hauteur d'un arbre, on ajoute 1 à la plus grande des hauteurs entre celles des sous-arbres gauche et droit, d'où la méthode récursive suivante :

```
def hauteur(self):
    hauteur_gauche = -1
    hauteur_droite = -1

    if self.left is not None:
        hauteur_gauche = self.left.hauteur()

    if self.right is not None:
        hauteur_droite = self.right.hauteur()

    return 1 + max(hauteur_gauche, hauteur_droite)
```

4

```
def trouve(self, data):
    # Si la donnée correspond à la donnée courante
    if data == self.data:
        return self

    # Si la donnée est inférieure à la donnée
    courante et qu'il y a un sous-arbre gauche
    if data < self.data and self.left is not None:
        return self.left.trouve(data)

    # Si la donnée est supérieure à la donnée
    courante et qu'il y a un sous-arbre droit
    if data > self.data and self.right is not None:
        return self.right.trouve(data)

    # Si la donnée n'est pas trouvée
    return None
```

Il n'y a rien de compliqué pour cette méthode : si `data` coïncide avec la valeur du nœud courant, on retourne le nœud ; si elle est inférieure, on va chercher du côté gauche et dans le cas contraire, du côté droit. Si rien n'est trouvé au final, on retourne `None`.

5 Là encore, rien de bien compliqué : on initialise une liste vide en argument de la méthode, on y ajoute la valeur du nœud courant et si ce nœud a des fils, on fait appel à cette méthode en passant la liste en argument pour chacun des fils, ce qui donne :

```
def liste(self, L = []):
    L.append(self.data)
    if self.left != None:
        self.left.liste(L)
    if self.right != None:
        self.right.liste(L)
    return sorted(L)
```

6 Une possibilité est celle donnée page suivante.

L'idée consiste ici à comparer le parent du nœud courant et le fils gauche (ou droit) du parent.

« `self is self.parent.left` » est déjà un booléen : il représente si le nœud courant est bien le fils gauche de son parent.

Ainsi, « `self.parent and self is self.parent.left` » s'assure que le parent existe ET que le nœud courant est son fils gauche.

Il en est bien sûr de même pour le fils droit.

```
# l'enfant est-il le fils gauche de son parent ?
def is_left_child(self):
    return self.parent and self is self.parent.left

# l'enfant est-il le fils droit de son parent ?
def is_right_child(self):
    return self.parent and self is self.parent.
    right
```

7 Comme nous sommes dans un ABR, le nœud ayant la plus petite valeur est celui qui est situé le plus à gauche, d'où la solution suivante :

```
def minimum(self):
    node = self
    while node.left is note None:
        node = node.left
    return node
```

On parcourt l'arbre tant qu'un fils gauche existe, et si tel n'est pas le cas, c'est que l'on est sur le nœud qui a la plus petite valeur.

8 Tous les nœuds du sous-arbre gauche de N ont une valeur plus petite, donc son successeur se trouve dans son sous-arbre droit. D'après les propriétés de l'ABR, ça sera la plus petite valeur de ce sous-arbre droit : son dernier fils gauche s'il y a des descendants à gauche, sinon son premier nœud.

Si le nœud n'a pas de sous-arbre droit, il faut remonter dans l'arbre pour trouver une valeur plus grande car c'est le seul endroit où on pourra en trouver une. Le successeur sera le premier des parents tel que N apparaît dans son sous-arbre gauche. Cela donne alors la méthode suivante :

```
def successeur(self):
    if self.right is not None:
        return self.right.minimum()
    node = self
    while node.is_right_child():
        node = node.parent
    return node.parent
```

9 C'est sans doute la méthode la plus difficile à écrire ; il faut donc s'armer de patience et de persévérance pour arriver à nos fins.

On commence par rechercher la valeur dans l'arbre avec la méthode `trouve` car si la valeur n'existe pas, inutile de faire quoi que ce soit.

Ensuite, en définissant la méthode par : `def delete(self,data)`, on teste la valeur du nœud : `if self.data == data`.

Le cas le plus simple est quand le nœud n'a pas de fils : dans ce cas, on le supprime tout simplement en spécifiant que le parent n'a plus de fils gauche ou droit selon sa position.

Si le nœud a un fils unique, alors il est remplacé par son fils.

Si le nœud a deux fils, on remplace le nœud par son successeur :

- son successeur sera forcément dans son sous-arbre droit car le nœud a deux enfants, donc un sous-arbre droit ;
- si son successeur a des enfants, ils ne pourront être qu'à sa droite car son successeur est dans notre cas la plus petite valeur du sous-arbre droit ;
- on remplace donc la valeur du nœud courant par celle de son successeur ;
- on corrige les branches de l'arbre pour aller de notre nœud vers le fils de son successeur (et vice-versa) pour rendre le successeur orphelin et le sortir de l'arbre ;
- on supprime son successeur.

Si la valeur `data` est inférieure à celle du nœud courant, on s'oriente vers le sous-arbre gauche et si elle lui est supérieure, vers le sous-arbre droit. Cela donne la méthode suivante `delete` dans la classe à télécharger :

```
def delete(self, data):
    if not self.trouve(data):
        return
    if data == self.data:
        # si le noeud n'a pas de fils
        if self.left is None and self.right is None:

            if self.parent.left.data == data:
                self.parent.left = None
            else:
                self.parent.right = None
            del self
       # si le noeud a un fils...
        elif self.left is None:
            if self.parent.left is not None and
    self.parent.left.data == data:
                self.parent.left = self.right
            else:
                self.parent.right = self.right
            del self
    # (suite dans le programme complet...)
```

10 Stockage d'identifiants

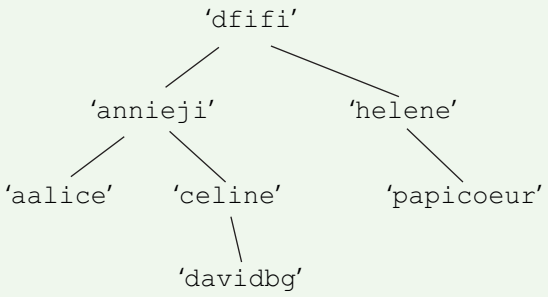
Énoncé p. 199

Lycée Richelieu, Versailles

1 La taille de l'arbre est 5 ; en effet, cet arbre comporte 5 nœuds.

Quant à sa hauteur, elle est égale à 3 car il y a 3 niveaux dans cet arbre.

2 L'arbre complété est la suivant :



3 Pour obtenir la liste des identifiants dans l'ordre lexicographique, il faut utiliser un parcours en profondeur dans l'ordre infixe de l'arbre binaire : réponse **C**.

4 La méthode dûement complétée est la suivante :

```
def present(self, identifiant):
    # Si l'arbre est vide, retourner False
    if self.est_vide() is not None:
        return False

    # Si l'identifiant correspond à la racine,
    retourner True
    if self.racine() == identifiant:
        return True

    # Si l'identifiant est supérieur à la racine,
    chercher dans le sous-arbre droit
    if self.racine() < identifiant:
        return self.sd().present(identifiant)

    # Si l'identifiant est inférieur à la racine,
    chercher dans le sous-arbre gauche
    if self.racine() > identifiant:
        return self.sg().present(identifiant)
```

Chapitre 7

Bases de données

Plan du chapitre

Exercice type — **Lycée Jean Macé, Niort**

Une base de données a la structure relationnelle suivante :

```
personnes (id_prenom,prenom)
genres (id_genre,designation)
livres (id_livre,titre,auteur,id_genre,id_prenom)
```

- La table « personnes » désigne l'ensemble des membres de la famille Numos ;
- La table « genres » représente l'ensemble des genres des livres que possède la famille (policier, science-fiction, etc.) ;
- La table « livres » désigne l'ensemble des livres de la famille Numos. L'attribut « id_prenom » sera vide si le livre n'est pas en la possession d'un membre de la famille.

Les attributs des tables sont explicites.

1 Pour chaque table, quel attribut peut être une clé primaire ?

2 Pour la table « livres », existe-t-il un attribut qui peut être défini comme clé étrangère ?

3 La famille Numos est composée de Mathilde, François, Nathan, Ludovic et Jeanne.
Écrire une requête SQL permettant d'insérer une de ces informations dans la table « personnes ».

4 On suppose qu'un membre de la famille peut être en possession de plusieurs livres en même temps.
Écrire une requête SQL permettant de trouver tous les livres en possession de Mathilde.

5 Écrire une requête SQL permettant de calculer le nombre total de livres indisponibles.

Voir corrigé page 236

COURS

PROBLÉMATIQUE

Comment stocker, manipuler et partager un volume d'informations toujours plus important ?

1 Systèmes de Gestion de Bases de Données (SGBD)

1.1 Introduction

Définition 1

Un Système de Gestion de Bases de Données (SGBD) est un outil informatique permettant la sauvegarde, l'interrogation, la recherche et la mise en forme de données.

Un SGBD est donc un ensemble de logiciels systèmes qui permettent aux programmateurs d'insérer, de modifier et de rechercher des données spécifiques dans une grande masse d'informations partagées par plusieurs utilisateurs.

Ces informations sont généralement enregistrées sur des disques magnétiques (par exemple, des serveurs).

Un SGBD donne l'impression à chaque utilisateur qu'il est le seul à travailler avec les données.

Oracle Database, 4D, Microsoft SQL Server, SQLite ou MySQL sont des exemples de SGBD très répandus. SQLite et MySQL sont des logiciels libres (open source), et sont par conséquent très utilisés, aussi bien par le grand public que par les professionnels.

1.2 Fonctions d'un SGBD

En plus des fonctions primaires citées dans la définition précédente, un SGBD :

- assure le partage des données ;
- vérifie qu'une opération s'effectue entièrement ou pas du tout : c'est ce que l'on nomme l'*atomicité* ;
- protège les données contre tout incident (détérioration, suppression accidentelle, etc.) ;
- optimise les performances (temps de recherche minimisé par exemple).

Les SGBD permettent la description des données (définition des types par des noms, formats, ...) de manière séparée de leur utilisation (mise à jour et recherche). Ils permettent aussi de retrouver les caractéristiques d'un type de données à partir de son nom (par exemple, comment est décrit un produit).

1.3 Les différents modèles de SGBD

Il existe cinq modèles de SGBD, mais seul le modèle relationnel est au programme de NSI.

- *le modèle hiérarchique* : les données sont classées hiérarchiquement, selon une arborescence descendante (arbre). Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD.

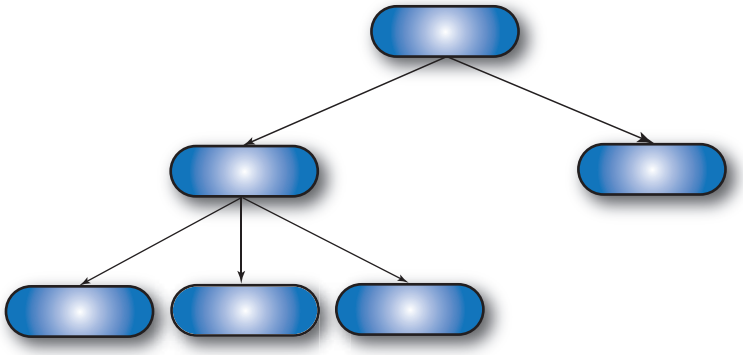


Figure 7.1 – Modèle hiérarchique d'un SGBD

- *le modèle réseau* : ce modèle ressemble au modèle hiérarchique à ceci près qu'il n'est plus nécessairement descendant.

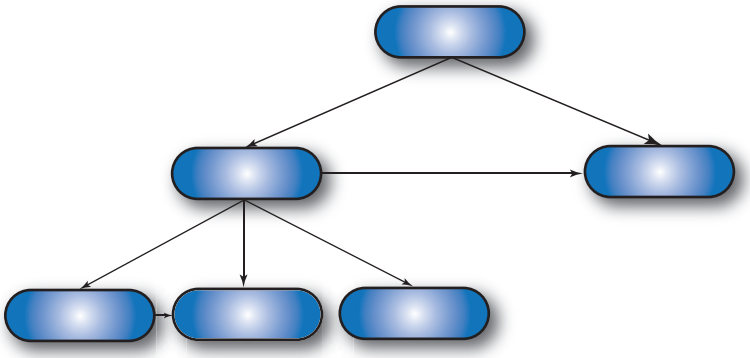


Figure 7.2 – Modèle réseau d'un SGBD

- *le modèle relationnel (SGBDR)* : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes) ;
- *le modèle déductif* : similaire au modèle relationnel, mais la manipulation des tables se fait différemment (nous ne donnerons pas de détails sur les calculs utilisés car hors programme) ;
- *le modèle objet (SGBDO)* : les données sont stockées sous forme de classes.

COURS

Depuis la fin des années 1990, le modèle relationnel est le plus répandu (environ trois quarts des bases de données utilisent ce modèle). Dans la suite de cet ouvrage, nous ne nous intéresserons qu'à ce modèle.

2 Le modèle relationnel

2.1 Historique

Le modèle relationnel a été imaginé dans les années 1970 par l'informaticien britannique Edgar Frank Codd. Avant cela, les modèles de bases de données ne permettaient pas de décrire de façon satisfaisante les relations entre deux données à l'aide de pointeurs logiques. Dix années de recherches ont été nécessaires avant de publier l'article « A Relational Model of Data for Large Shared Data Banks » (*un modèle de données relationnel pour de grandes banques de données partagées*), CACM 13, No. 6, June 1970.

2.2 Représentations

Comme nous l'avons dit dans la section précédente, le modèle relationnel consiste à représenter les données dans des tableaux, que l'on appelle *tables*. Chaque table est une sorte de classe, qui admet donc des attributs.

Prenons l'exemple d'un site internet, qui demande à ses utilisateurs trois données afin de pouvoir leur créer un compte : nom, prénom et email. On peut alors imaginer une classe *user*, représentant un utilisateur ou une utilisatrice, classe comportant les trois attributs : surname, name et email. Cette classe est alors représentée par la table :

user		
surname	name	email

Tableau 7.1 – Structure de la table représentant les utilisateurs

Une fois la structure de la table connue, nous pouvons y insérer les données, par exemple :

user		
surname	name	email
Dupont	Jean	jean.dupont@free.fr
Durand	Anne	anne.dur@orange.fr
⋮	⋮	⋮

Tableau 7.2 – Insertion de données dans la table représentant les utilisateurs

2.3 Définitions

Définitions 2

- Le *schéma relationnel* d'une table SQL décrit sa structure ; il spécifie le nom de la table ainsi que les différentes catégories d'informations qu'elle va contenir (les colonnes).
- Dans une table, chaque ligne constitue un *enregistrement*.
- Les *attributs* d'une table sont les noms de ses colonnes.
- Le *degré* d'une table est le nombre de ses attributs.
- Le *domaine* d'un attribut est son type : entier, flottant, chaîne de caractères, booléen, ...
- La *clé primaire* d'une table est l'attribut qui permet d'identifier de manière unique (sans aucun risque de doublon) un enregistrement de cette table.
- Une *clé étrangère* est un attribut d'une table qui fait référence à la clé primaire d'une autre table. Une clé étrangère permet de mettre en relation un enregistrement d'une table (appelée *table fille*) qui la contient avec un enregistrement de la table référencée (appelée *table parent*).

Exemple : reprenons la table 7.2 précédente et ajoutons-y une clé primaire *id* :

user			
id	surname	name	email
1	Dupont	Jean	jean.dupont@free.fr
2	Durand	Anne	anne.dur@orange.fr
⋮	⋮	⋮	⋮

Tableau 7.3 – Table avec clé primaire

Les attributs de cette table sont donc : *id*, *surname*, *name* et *email*.
Le degré de la table est donc égal à 4 car il y a quatre attributs.
Le domaine de l'attribut *id* est : entier, et celui des autres attributs est : chaîne de caractères.

Imaginons maintenant que ce site internet propose plusieurs abonnements : « Free », « Basic » et « Premium ».
On peut alors créer une seconde table nommée « abonnement » représentant les abonnements (ci-contre).

abonnement	
id_abo	type
1	Free
2	Basic
3	Premium

Tableau 7.4 – Table des abonnements

COURS

On peut alors mettre en relation les tables « user » et « abonnement » en insérant une clé étrangère dans la première table :

user				
id	surname	name	email	id_abo
1	Dupont	Jean	jean.dupont@free.fr	3
2	Durand	Anne	anne.dur@orange.fr	1
⋮	⋮	⋮	⋮	

Tableau 7.5 – Table des utilisateurs avec clé étrangère

On met ici en relation les deux tables en considérant que l'attribut *id_abo* de la table « user » correspond à l'attribut *id_abo* de la table « abonnement ».
Ainsi, si l'on regarde le premier enregistrement de la table « user », on peut voir que Jean Dupont a souscrit à un abonnement Premium, c'est-à-dire à l'enregistrement de la table « abonnement » dont l'attribut *id_abo* est 3.

2.4 Schématisation

Pour schématiser la relation qu'il peut y avoir entre deux tables, on peut utiliser la représentation suivante (en reprenant les deux tables de l'exemple précédent) :

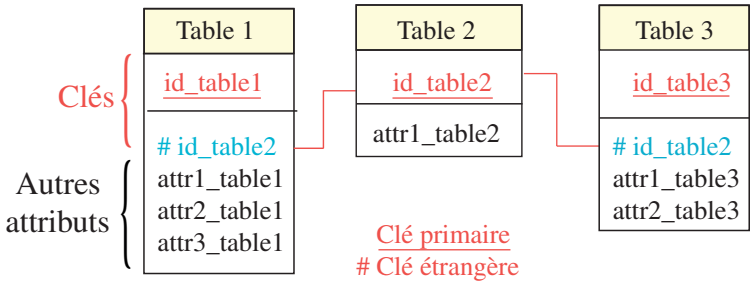


Figure 7.3 – Schématisation de la relation entre deux tables

La relation entre les deux tables est indiquée par un trait qui les lie.

Dans chacune des tables, on peut indiquer la présence d'une clé primaire en la soulignant, et la présence d'une clé étrangère en précédant l'attribut d'un dièse (#).

On peut aussi, pour plus de clarté, séparer les clés des autres attributs.

Définition 3

Une *base de données* est un ensemble de tables, dont certaines peuvent être mises en relation.

3 Jointures

3.1 Produit cartésien

Définition 4

On considère deux tables T_1 et T_2 d'une base de données.
Le *produit cartésien* $T_1 \times T_2$ est l'ensemble :

$$T_1 \times T_2 = \{(e_1, e_2) \mid e_1 \in T_1 \text{ et } e_2 \in T_2\}.$$

Exemple : si l'on considère les deux tables suivantes,

T_1	
id	surname
1	Dupont
2	Durand

Tableau 7.6 – Table 1

T_2	
stockage	price
201	15
41	7

Tableau 7.7 – Table 2

le produit cartésien de ces deux tables est :

$T_1 \times T_2$			
id	T1.surname	T2.stockage	T2.price
1	Dupont	201	15
1	Dupont	41	7
2	Durand	201	15
2	Durand	41	7

Tableau 7.8 – Produit cartésien

3.2 Jointure

Une jointure permet de créer une liaison entre deux ou plusieurs tables pour pouvoir récupérer un résultat bien précis. Le résultat est en quelque sorte une fusion virtuelle de plusieurs tables.

Définition 5 : jointure

Soient T_1 et T_2 deux tables. La *jointure interne* de ces deux tables sur un attribut donné A est une table formée en ne prenant que les enregistrements de $T_1 \times T_2$ tels que $T_1.A = T_2.A$.

On peut représenter cette jointure par le schéma suivant :

COURS

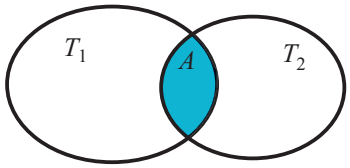


Figure 7.4 – Schématisation d'une jointure interne

Exemple : en considérant les deux tables suivantes,

user			
id	id_town	surname	name
1	1	Dupont	Jean
2	2	Durand	Agathe
3	1	Dufon	Jérôme

Tableau 7.9 – Table T_1

town	
id_town	name_town
1	Bordeaux
3	Paris

Tableau 7.10 – Table T_2

leur jointure sur l'attribut « id_town » est :

id	id_town	surname	name	id_town	name_town
1	1	Dupont	Jean	1	Bordeaux
3	1	Dufon	Jérôme	1	Bordeaux

Tableau 7.11 – Jointure interne de T_1 et T_2

4 Le langage SQL

4.1 Introduction

Développé par IBM dans les années 1970, le langage SQL (pour « Structured Query Language », ou « langage de requête structurée » en français) est rapidement devenu le standard des langages de bases de données relationnelles. Il permet de manipuler la structure d'une base de données et d'y rechercher, insérer, modifier ou supprimer les données à l'aide d'une vingtaine d'instructions dont la syntaxe ressemble à celle de phrases ordinaires en anglais. SQL peut s'utiliser de manière interactive, ou à l'intérieur d'un langage hôte tel que C, Fortran, ou Cobol, par exemple.

4.2 Environnements nécessaire à l'utilisation de SQL

4.2.1 - Généralités

Pour travailler sur des bases de données avec SQL, plusieurs outils sont nécessaires :

- un serveur pour héberger la base,
- un SGBD pour la manipuler.

Si l'on souhaite utiliser SQL de manière interactive, il nous faudra un logiciel d'édition ou une interface graphique pour entrer manuellement les requêtes.

Au Lycée, vous aurez principalement affaire à SQLite et MyPHP, dont nous allons rapidement étudier les environnements.

4.2.2 - Environnement SQLite

SQLite est une solution légère et gratuite, ce qui en fait la plus utilisée au monde. Quelque soit votre système d'exploitation, vous pouvez télécharger les outils nécessaires sur :

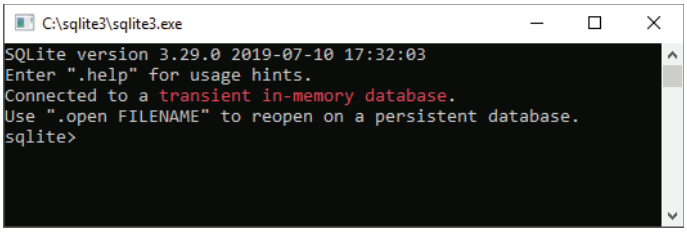
https ://www.sqlite.org/download.html

Par exemple, sous Windows 10, vous pouvez télécharger le fichier :

`sqlite-tools-win-x64-3490100.zip`

qui contient trois exécutables : `sqlite3`, `sqldiff` et `sqlite3_analyser`.

Lorsque le programme principal sqlite3 est exécuté, une console permet de lancer les requêtes SQL nécessaires à la manipulation des bases de données.



Cependant, cette console n'étant pas toujours très pratique, on peut aussi utiliser une interface graphique telle que DB Browser ou SQLiteSpy.

4.2.3 - Environnement MySQL

MySQL étant un logiciel libre, il s'intègre parfaitement dans un environnement composé du système d'exploitation Linux, du serveur Apache et du langage PHP. Cet environnement est appelé *LAMP*, acronyme de « Linux - Apache - MySQL - PHP ».

COURS

Sous Windows, l'environnement correspondant s'appelle *WAMP* (W pour Windows), et sous Mac *OS MAMP* (M pour Mac).

Notons que le langage PHP est le complément idéal à MySQL pour la création de pages Web dynamiques via un serveur HTTPS.

PhpMyAdmin est l'interface la plus connue pour utiliser MySQL.

Une fois l'environnement installé avec MySQL ou SQLite, nous sommes prêts à manipuler des bases de données avec SQL.

4.3 Les requêtes SQL

Retrouvez une explication des requêtes SQL en vidéo.

4.3.1 - Fonctions de base

La syntaxe des fonctions de base en SQL est très simple.

- Projection d'un ou plusieurs attributs dans une table :

```
SELECT
    attribut1,
    attribut2
FROM
    matable
```

- Classification des données selon un ou plusieurs attributs :

```
SELECT
    *
FROM
    matable
ORDER BY
    attribut
```

À noter que « * » désigne la totalité des attributs.

- Projection d'un ou plusieurs attributs dans une table avec contrainte :

```
SELECT
    attribut1,
    attribut2
FROM
    matable
WHERE
  condition
```

- Insertion d'un enregistrement dans une table :

```
INSERT INTO
    matable
VALUES (
    'valeur 1',
    'valeur 2',
    ...
)
```

- Modification d'une ou plusieurs valeurs d'attributs :

```
UPDATE
    matable
SET
    attribut1 = 'valeur',
    attribut2 = 'valeur'
WHERE
    condition
```

- Suppression d'un enregistrement dans une table :

```
DELETE FROM
    matable
WHERE
    condition
```

- Suppression des doublons sur un attribut :

```
SELECT DISTINCT
    attribut
FROM
    matable
```

Les deux requêtes suivantes sont hors-programme,mais nous choisissons de vous les présenter en complément :

- Création d'une table nommée « matable » dans la base de données :

```
CREATE TABLE matable (
  attribut1 type,
  attribut2 type,
  ...
)
```

COURS

- Suppression de la table :

```
DROP TABLE matable
```

Exemple : à l'aide d'une interface graphique (DB Browser pour SQLite ou PhpMyAdmin pour MySQL), il est facile de créer une base de données nommée « gestion » ainsi que deux tables :

→ « restaurants » d'attributs : id, nom, rue, cp, ville, email, tel et id_gerant ;

→ « gerants » d'attributs : id, nom, prenom, email et tel.

- Pour changer l'adresse email de l'enregistrement dont l'id est 1, on écrira :

```
UPDATE
    restaurants
SET
    email = "chezmomo@free.fr"
WHERE
    id = 1;
```

- Pour supprimer tous les enregistrements concernant la ville de Nantes dans la table « restaurants », on écrira :

```
DELETE FROM
    restaurants
WHERE
    ville = "Nantes"
```

4.3.2 - Jointure

La syntaxe pour une jointure est la suivante :

```
SELECT
    champ1,
    ...
FROM
    table1
JOIN
    table2
ON
    table1.champA = table2.champB
```

Cette requête joint les deux tables `table1` et `table2` et on choisit un critère de sélection avec « ON ». La condition qui vient après peut être une égalité ou autre chose.

4.3.3 - Opérations d'agrégation

4.3.3.1 - Ajouter

Dans certains cas, il est nécessaire d'ajouter toutes les entrées d'une colonne (d'un attribut). On pourra le faire à l'aide de la fonction d'agrégation SUM.

Exemple : considérons la table « commandes » suivante :

commandes		
id_commande	id_client	total
1	12	125
2	7	35
3	8	50

Tableau 7.12 – Table « commandes »

La requête :

```
SELECT SUM(montant) AS somme
FROM commandes
```

retourne :

somme
210

où « 210 » représente le total des valeurs de l'attribut `montant` des commandes. Il n'est pas indispensable de spécifier d'alias (`AS`), mais c'est assez pratique dans les requêtes plus complexes. Dans notre exemple, nous avons choisi l'alias « somme » pour désigner la somme obtenue.

4.3.3.2 - Compter

Dans certains cas, savoir combien une colonne comporte d'entrées s'avère utile. On le fera avec la fonction d'agrégation COUNT.

Exemple : prenons à nouveau la table « commandes » du tableau 7.12 de l'exemple précédent.

Si on considère la requête suivante :

```
SELECT COUNT(*) AS nb
FROM commandes
WHERE montant < 100
```

alors, elle affiche :

nb
2

car il y a deux enregistrements où la valeur de l'attribut « montant » est strictement inférieure à 100.

4.3.3.3 - Effectuer une moyenne

COURS

Cela se fait à l'aide de la fonction d'agrégation AVG (abréviation de *average*, qui signifie « moyenne » en anglais).

Exemple : en reprenant la table représentée dans le tableau 7.12, la requête :

```
SELECT AVG(montant) AS
    moy
FROM commandes
```

affiche :

moy
66.6667

où « 66,6667 » représente donc la moyenne des commandes.

4.3.3.4 - Maximum et minimum

Connaître le maximum et le minimum des valeurs d'un attribut se fait avec les fonctions d'agrégation MAX et MIN.

Exemple : toujours avec la table représentée par le tableau 7.12, la requête :

```
SELECT
    MIN(C.montant) AS minimum ,
    MAX(C.montant) AS maximum
FROM commandes C
```

affiche :

minimum	maximum
25	125

4.3.4 - Complément : créer une vue (hors-programme)

Il sera de temps en temps utile de nommer la table obtenue à l'issue d'une requête ; c'est ce que l'on appelle *créer une vue* de la requête. Prenons par exemple la requête précédente qui permet de calculer le maximum et le minimum, et créons une vue que l'on nomme « mavue »

```
CREATE VIEW mavue AS
    SELECT
        MIN(C.montant) AS minimum ,
        MAX(C.montant) AS maximum
    FROM commandes C
```

On peut ainsi utiliser la table (la vue) dans une autre requête, par exemple pour calculer l'écart entre le maximum et le minimum :

```
SELECT maximum - minimum FROM mavue
```

5 Manipuler les bases de données avec Python

Pour manipuler des bases de données avec Python, on fera appel au module `sqlite3` ou au module `mysql.connector` selon le serveur SQL que l'on aura installé.

5.1 Avec SQLite 3

Retrouvez une mise en pratique de SQLite 3 en Python en vidéo.

5.1.1 - Préliminaires

Il faut d'abord faire appel au module `sqlite3` :

```
import sqlite3
```

Ensuite, il faut établir une connexion à une base de données :

```
conn = sqlite3.connect('gestion.db')
```

Ici, on donne le nom « conn » à notre connexion, et on donne le nom « gestion.db » à notre base de données. À ce stade, cette dernière n'existe pas encore. Pour la créer, on écrit :

```
cursor = conn.cursor()
```

On donne en quelque sorte le curseur à notre base.

Cette opération provoque la création d'un fichier `gestion.db` dans le répertoire qui contient le script Python. C'est dans ce fichier que l'on va stocker les tables créées ainsi que leurs enregistrements.

ATTENTION

Le format du fichier est propre à Sqlite (mélange de binaire et de texte) et ne doit pas être manipulé directement via un éditeur de texte. C'est le SGBD qui maintient ce fichier à jour en y exécutant les requêtes qui lui sont transmises (ici par Python).

5.1.2 - Passer une requête SQL

Pour exécuter une requête SQL, on utilisera la commande `cursor.execute` :

```
cursor.execute("""<requête>""")
```

COURS

Exemples

- Pour créer une table :

```
cursor.execute("""
CREATE TABLE restaurants (id INTEGER PRIMARY KEY,
     nom TEXT,
     ville TEXT) """)
```

- Pour insérer un enregistrement :

```
cursor.execute("""
INSERT INTO restaurants(nom, ville) VALUES(?, ?)""",
("Chez Momo", "Marseille"))
```

5.1.3 - Insérer un enregistrement à partir d'un dictionnaire

On utilise toujours la méthode `execute` mais la syntaxe est légèrement différente, comme le montre l'exemple de la page suivante.
Exemple :

```
dico = {"nom" : "Chez Fanny", "ville" : "Marseille"}
cursor.execute("""
INSERT INTO restaurants(nom, ville)
VALUES(:nom, :ville)""", dico)
```

5.1.4 - Insérer plusieurs enregistrements à partir d'une liste

On utilise ici la méthode `executemany` comme dans l'exemple suivant.

Exemple :

```
liste = []
liste.append(("Le cochon vert","Paris"))
liste.append(("Le gazon maudit","Paris"))
cursor.executemany("""
INSERT INTO restaurants(nom, ville) VALUES(?, ?)""", liste)
```

5.1.5 - Afficher les enregistrements

On pourra utiliser une syntaxe comme celle de l'exemple suivant.

Exemple :

```
print('Tous les enregistrements :')
cursor.execute("""
SELECT id, nom, ville FROM restaurants""")
for row in cursor:
    print('{0} : {1}, {2}'.format(row[0], row[1], row[2]))
```

5.1.6 - Enregistrements avec la clause WHERE

Quand on souhaite afficher les enregistrements en fonction de la valeur d'un attribut avec la clause WHERE, la syntaxe est un peu spéciale, comme le montre l'exemple suivant.

Exemple :

```
print('Les restaurants de la ville de Marseille :')
cursor.execute("""
SELECT id, nom FROM restaurants
WHERE ville = %s""", ("Marseille", ))
for row in cursor:
    print('{0} : {1}'.format(row[0], row[1]))
```

5.1.7 - Valider les modifications

Avant de clore la connexion, il faut s'assurer que les éventuelles modifications ou insertions soient prises en compte. On utilise alors la méthode `commit()` :

```
conn.commit()
```

5.1.8 - Clôture de la connexion

Quand on a fini de se servir de la base de données, il ne faut pas oublier de clore la connexion.

```
conn.close()
```

Télécharger le programme Python complet sur sqlite3.

COURS

Pour plus d'informations sur le module `sqlite3`, n'hésitez pas à vous servir de la commande *help* :

```
>>> import sqlite3
>>> help(sqlite3)
```

5.2 Avec mysql.connector

Par défaut, ce module (auquel on doit faire appel si l'on utilise MySQL) n'est pas installé. Il faudra donc se renseigner pour l'installer sur votre distribution Python (avec la commande `pip` ou, sous Anaconda, la commande `conda`).

Si l'on part de la base de données « gestion » définie précédemment, voici un exemple.

```
import mysql.connector
# connexion à la base de données "gestion"
conn = mysql.connector.connect(host="localhost", user="root
    ",
    password="", database="gestion")
cursor = conn.cursor()
# Afficher selon un critère sur la valeur de "ville"
cursor.execute("""SELECT id, nom, email
    FROM restaurants
    WHERE ville = %s""", ("Marseille",))
rows = cursor.fetchall()
for row in rows:
    print('{0} : {1} - {2}'.format(row[0], row[1], row[2]))
conn.close()
```

Solution de l'exercice type — Lycée Jean Macé, Niort

1. Pour la table « personnes », l'attribut « id_prenom » peut être une clé primaire. En effet, une clé primaire est un attribut dont les valeurs sont toutes distinctes. Dans notre situation, pour la table « personnes », tous les enregistrements ont un id_prenom différent.
   Il en est de même pour la table « genres » : « id_genre » peut être une clé primaire.
   Pour la table « livres », l'attribut « id_livres » peut être une clé primaire.
2. Pour la table « livres », l'attribut « id_prenom » est lié à la table « personnes » donc il peut être défini comme clé étrangère. Mais « id_genre » est lié à la table « genres » donc peut aussi être une clé étrangère.

Solution de l'exercice type — Lycée Jean Macé, Niort

3 Une requête SQL permettant d'insérer une des informations dans la table « personnes » est :

```
INSERT INTO
    personne (prenom)
VALUES
    ('Mathilde')
```

Pour insérer tous les enregistrements, il suffit de répéter quatre autres fois cette requête en changeant bien entendu le prénom.

On peut aussi, plus simplement, faire la requête suivante :

```
INSERT INTO
    personne (prenom)
VALUES
    ('Mathilde'), ('François'),
    ('Nathan'),   ('Ludovic'),
    ('Jeanne')
```

4 Dans cette question, la difficulté réside dans le fait que dans la table « films », les prénoms n'apparaissent pas ; seuls les identifiants sont enregistrés. On va donc utiliser une jointure :

```
SELECT *
FROM livres
   JOIN personne
   ON personne.id_prenom = livres.id_prenom
WHERE
   personne.prenom = 'Mathilde'
```

5 Une requête SQL permettant de calculer le nombre total de livres indisponibles est la suivante :

```
SELECT
    COUNT(*)
FROM
    livres
WHERE
    id_prenom IS NOT NULL
```

COURS
INTERROS
CORRIGÉS

Solution de l'exercice type (suite)

Lycée Jean Macé, Niort

Il s'agit ici de compter tous les enregistrements de la table « livres » dont la valeur de l'attribut « id_prenom » n'est pas vide (ce qui signifie qu'un membre de la famille possède le livre correspondant, et donc que ce dernier est indisponible).

L'attribut « id_prenom » étant par définition un entier, sa valeur sera nécessairement « NULL » si aucune valeur ne lui a été attribuée.

Voir énoncé page 219

1 QCM Vérification de connaissances

5 min

Corrigé p. 255

On dispose d'une base de données comportant les tables suivantes :

```
clients
    (idClient,designation,adresse,cp,ville,email,tel)
produits
    (idProduit,designation,prix,idEntrepot)
commandes
    (idCommande,date,idProduit,idClient,quantite)
entrepots
    (idEntrepot,designation,adresse,cp,ville,superficie)
```

Le nom des champs (entre les parenthèses) est choisi de façon implicite pour qu'il n'y ait pas d'ambiguïté.

1 La requête :

```
SELECT * FROM clients ORDER BY ville DESC
```

a affiche le nom de tous les clients par ordre alphabétique de leur ville ;

b affiche le nom de tous les clients selon un ordre alphabétique inverse de leur ville ;

c affiche tous les champs de la table « clients » par ordre alphabétique inverse de leur nom ;

d affiche tous les champs de la table « clients » par ordre alphabétique inverse du nom de leur ville.

2 La requête :

```
SELECT *
FROM (
    clients JOIN entrepots
    ON clients.ville = entrepots.ville)
```

a affiche tous les champs des tables « clients » et « entrepots » qui ont la même ville ;

b affiche tous les champs de la table « clients » où le contenu du champ « ville » est identique à celui d'au moins un champ de la table « entrepots » ;

c affiche tous les champs de la table « entrepots » où le contenu du champ « ville » est identique à celui d'au moins un champ de la table « clients » ;

d affiche les contenus du champ « ville » communs aux tables « entrepots » et « clients ».

3 La requête :

```
SELECT *
FROM (
    clients JOIN entrepots
    ON clients.ville = entrepots.ville)
```

est équivalente à :

a
```
SELECT * FROM clients
WHERE clients.ville = entrepots.ville
```
b
```
SELECT * FROM entrepots
WHERE clients.ville = entrepots.ville
```
c
```
SELECT * FROM clients,entrepots
WHERE clients.ville = entrepots.ville
```
d
```
SELECT ville FROM clients,entrepots
WHERE clients.ville = entrepots.ville
```

4 La requête :

```
SELECT
    SUM(commandes.total) AS somme
FROM
    commandes
WHERE
    commandes.date > '2019-09-02'
```

a affiche le total des commandes effectuées après le 2019-09-02 ;
b affiche le total de chaque commande effectuée après le 2019-09-02 ;
c affiche toutes les entrées de la table « commandes » où la date est supérieure à 2019-09-02 en effectuant la somme des totaux petit à petit ;
d n'affiche rien.

2 V/F Vérification de connaissances

10 min Corrigé p. 255

Les affirmations suivantes sont-elles vraies ou fausses ? Justifier la réponse.

On se place dans une base de données nommée « bddPerso ».

1 Une ligne d'une table de `bddPerso` est appelée une entrée.

2 Une table de `bddPerso` est définie par la structure relationnelle suivante :

```
clients (idClient , nom , prenom)
```

Alors « nom » est un attribut de la table « clients ».

3 Dans la table « clients » de la question précédente, on peut prendre « nom » comme clé primaire.

4 On définit une autre table de la manière suivante :

```
connexions (idCommande , idClient , date)
```

où « idClient » fait référence à la clé du même nom de la table « clients ».

« idClient » est alors une clé étrangère de la table « connexions ».

3 Festival de musique

★ 15 min Corrigé p. 256

Lycée André Theuriet, Civray

On dispose d'une base de données d'un festival de musique, où il y a plusieurs représentations. Un ou plusieurs musiciens peuvent participer à une représentation, mais un musicien ne peut participer qu'à une seule représentation.

La structure de la base de données est la suivante :

```
Representation (idRep , titreRep , lieu)
Musicien (idMus , nom , idRep)
Programmer (Date , idRep, tarif)
```

Écrire la requête SQL permettant d'afficher :

1. La liste des titres des représentations.
2. La liste des titres des représentations ayant lieu sur le lieu nommé « Dionysos ».
3. La liste des noms des musiciens et les titres des représentations auxquelles ils participent.
4. La liste des titres des représentations, les lieux et les tarifs d'une date précisée.
5. Le nombre des musiciens qui participent à la représentation numéro 7.
6. Les représentations et leurs dates dont le tarif ne dépasse pas 30 €.

4 Les employés du département

★ 15 min Corrigé p. 257

Lycée Henri Alain-Fournier, Bourges

On dispose d'une base de données dont la structure est la suivante :

```
Departements (DNO, DNOM, DIR, VILLE)
Employes (ENO, ENOM, PROF, DATEEMB, SAL, COMM, DNO)
```

Donner les requêtes SQL qui permettent d'obtenir :

1. la liste des employés ayant une commission (attribut « COMM » déclaré comme booléen) ;

2. les noms, emplois et salaires des employés obtenus en classant les emplois dans l'ordre alphabétique, et pour chaque emploi, en classant les salaires du plus grands au plus petit ;
3. le salaire moyen des employés.
4. le salaire moyen dans le département nommé « Production » (`DNOM = 'Production'`).

5 Notes annuelles des étudiants ★★ 20 min Corrigé p. 258

Lycée Sainte-Clotilde, Strasbourg

On considère le modèle relationnel suivant concernant la gestion des notes annuelles d'une promotion d'étudiants :

```
ETUDIANT
    (NumEtudiant, Nom, Prenom)
MATIERE
    (CodeMat, LibelleMat, CoeffMat)
EVALUER
    (NumEval, #NumEtudiant, #CodeMat, Date, Note)
```

Les clés étrangères sont précédées d'un « # » et les clés primaires sont soulignées.

Exprimez en SQL les requêtes suivantes.

1. Quel est le nombre total d'étudiants ?
2. Quelles sont, parmi l'ensemble des notes, la note la plus haute et la note la plus basse ?
3. Quelles sont les moyennes de chaque étudiant dans chacune des matières ?
4. Quelles sont les moyennes de la classe par matière ?
5. Quelle est la moyenne générale de chaque étudiant ?
6. Quelle est la moyenne générale de la promotion ?

Aide : on pourra utiliser « GROUP BY » pour plusieurs questions, qui regroupe les entrées. Par exemple, si l'on considère la table suivante :

clients		
id	nom	montant
1	Marisol Pleureur	125.65
2	Jacques Umul	45.7
3	Marisol Pleureur	100.5
4	Jacques Umul	78.6

alors, la requête :

```
SELECT
    nom ,
    SUM(montant) AS total
FROM
    clients
GROUP BY
    nom
```

calculera le total des valeurs des attributs « montant » en les regroupant par nom, et donnera la table :

nom	total
Jacques Umul	124.3
Marisol Pleureur	226.15

6 Un cas pratique : site web marchand ★★★ 30 min Corrigé p. 261

Lycée Jean-Joseph Fourier, Auxerre

Un webmaster souhaite créer une base de données nommée « venteAdonf » pour un site internet marchand `vente-a-donf.com` dont la clientèle est uniquement française (pour simplifier les formats des enregistrements).

Dans cette base de données, il devra y avoir les quatre tables suivantes :

- `users (idUser,nom,prenom,email,rue,cp,ville,tel)`
  où « cp » désigne le code postal de sa ville de résidence, les autres attributs étant explicites ;
- `modeles (idModele,nomModele)`
- `articles (idArticle,nom,descriptif,idModele,prix)`
  où « idModele » est une clé étrangère relative à la colonne `idModele` de la table `modeles`.
- `panier (idPanier,idUser,idArticle,idModele,quantite, total)`
  Ici, « total » représente le résultat de « quantite * prix de l'article ».

On décide de mettre en clés primaires les premières colonnes de chaque table.

1 Écrire un fichier SQL nommé « venteAdonf-create.sql » permettant de créer ces quatre tables.

On rappelle qu'un tel fichier est composé de quatre requêtes séparées par un point-virgule.

INTERROS

2 Voici les premiers articles mis en vente :

Nom	Descriptif	Modèle	Tarif
Mug	Magnifique mug personnalisable avec votre photo.	Unique	7,99 €
T-shirt « I Kiffe Beef »	T-shirt unique en son genre.	S / M / L / XL / XXL	24,99 €

Écrire un fichier SQL nommé « venteAdonf-insertArt.sql » permettant d'insérer ces informations dans la base de données.

3 Le jour de la mise en ligne du site, deux personnes ont passé commande :

Nom	Dupont	Aimaun
Prénom	Jean	Anne
Email	jdupont@free.fr	amz@free.fr
Rue	3 rue des tulipes	1 rue du glas
Cp	75000	33000
Ville	Paris	Bordeaux
Tel	+33601020304	+33799989796

Écrire un fichier SQL nommé « venteAdonf-insertUsers.sql » permettant d'insérer ces informations dans la base de données.

4 Jean Dupont a commandé 2 mugs et 3 tee-shirts (tailles S, M et L). Anne Aimaun a, quant à elle, commandé 1 mug et 1 tee-shirt (taille M).

(a) Écrire un fichier SQL nommé « venteAdonf-insertPanier.sql » permettant d'insérer ces informations dans la base de données.

(b) Écrire une requête permettant d'afficher le montant total du panier de Jean Dupont.

(c) Si la table `panier` n'avait pas de colonne « total », quelle requête permettrait de renvoyer le résultat de la question précédente ?

Objectif bac

7 Voyages dans l'espace

★★★ 45 min Corrigé p. 264

Lycée Richelieu, Versailles

On considère une gestion simplifiée des voyages dans l'espace. La base de données utilisée est constituée de quatre relations nommées `Astronaute`, `Fusee`, `Equipe` et `Vol`. Voici le contenu des tables `Astronaute`, `Fusee`, `Equipe` et `Vol`.

Les clés primaires sont soulignées et les clefs étrangères sont précédées d'un # :

Astronaute				
id_astronaute	nom	prenom	nationalite	nb_vols
1	'PESQUET'	'Thomas'	'français'	2
2	'AMSTRONG'	'Neil'	'américain'	8
3	'MAURER'	'Mathias'	'allemand'	1
4	'MCARTHUR'	'Megan'	'américain'	5

Fusee			
id_fusee	modele	constructeur	nb_places
1	'Falcon 9'	'SpaceX'	6
2	'Starship'	'SpaceX'	100
3	'Soyouz'	'TsSKB Progress'	2
4	'SLS'	'Boeing'	6

Equipe	
id_vol	#id_astronaute
1	1
1	2
1	3
2	1
2	3
3	1
3	2
3	4
4	4
4	4

Vol		
id_vol	#id_fusee	Date
1	1	'12/09/2022'
2	4	'25/10/2022'
3	3	'18/11/2022'
4	2	'23/12/2022'

On pourra utiliser les mots clés suivants : COUNT, FROM, INSERT, INTO, JOIN, ON, ORDER BY, SELECT, VALUES, WHERE.

- Le mot clé COUNT permet de récupérer le nombre d'enregistrements issu de la requête.
  Par exemple, la requête suivante renvoie la valeur 4.

```
SELECT COUNT (*) FROM Astronaute;
```

- Le mot clé ORDER BY permet de trier les éléments par ordre alphabétique.
  Par exemple, la requête suivante :

```
SELECT modele FROM Fusee ORDER BY modele;
```

  renvoie la table :

'Falcon 9'
'SLS'
'Soyouz'
'Starship'

INTERROS

1 On s'intéresse ici à la notion de clés primaire et étrangère.

(a) Donner la définition d'une clé primaire.

(b) Dans la table `Astronaute`, la clé primaire est `id_astronaute`. Expliquer pourquoi cette requête SQL renvoie une erreur :

```
INSERT INTO Astronaute
VALUES (3, 'HAIGNERE', 'Claudie', 'français', 3);
```

(c) Le schéma relationnel de la table `Astronaute` est :

```
Astronaute (
      id_astronaute : INT,
      nom : TEXT,
      prenom : TEXT,
      nationalite : TEXT,
      nb_vols : INT)
```

Écrire le schéma relationnel de la table `Fusee` en précisant le domaine de chaque attribut.

2 On s'intéresse ici à la récupération d'informations issues de la base de données.

(a) Écrire le résultat que la requête suivante renvoie :

```
SELECT COUNT (*)
FROM Fusee
WHERE constructeur = 'SpaceX';
```

(b) Écrire une requête SQL qui renvoie le modèle et le constructeur des fusées ayant au moins quatre places.

(c) Écrire une requête SQL qui renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom.

3 **(a)** Recopier et compléter les requêtes SQL suivantes permettant d'ajouter un cinquième vol avec la fusée 'Soyouz' le 12/04/2023 avec l'équipage composé de PESQUET Thomas et MCARTHUR Megan. On ne s'intéresse pas ici à la mise à jour qui suivra.

```
INSERT INTO Vol VALUES(...);
INSERT INTO Equipe VALUES(...);
INSERT INTO ... VALUES (...);
```

(b) Écrire une requête SQL permettant d'obtenir le nom et le prénom des astronautes ayant décollé le '25/10/2022'.

8 Collection de CD

★★★ 45 min Corrigé p. 265

D'après Bac 2024, Asie 2, Jour 2

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL et la programmation en Python.

L'énoncé de cet exercice utilise des mots-clés du langage SQL suivants :

`SELECT, FROM, WHERE, JOIN... ON, UPDATE... SET, INSERT INTO... VALUES..., COUNT, ORDER BY.`

La clause ORDER BY suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut précisé. `SELECT COUNT (*)` renvoie le nombre de lignes d'une requête.

Amélie souhaite organiser sa collection de CD. Elle a commencé par enregistrer toutes les informations sur un fichier CSV mais elle trouve que la recherche d'informations est longue et fastidieuse. Elle repense à son cours sur les bases de données et elle se dit qu'elle doit pouvoir utiliser une base de données relationnelle pour organiser sa collection.

Partie A

Dans cette partie on utilise une seule table. Voici un extrait de la table `chanson`.

Chanson			
id	titre	album	groupe
1	Sunburn	Showbiz	Muse
2	Muscle Museum	Showbiz	Muse
3	Showbiz	Showbiz	Muse
4	New Born	Origin of Symmetry	Muse
5	Sing for Absolution	Absolution	Muse
6	Hysteria	Absolution	Muse
7	Welcome too the Jungle	Appetite for Destruction	Guns N' Roses
8	Muscle Museum	Hullabaloo	Muse
9	Showbiz	Hullabaloo	Muse

1. L'attribut titre peut-il être une clé primaire pour la table chanson ? Justifier.
2. Donner le résultat de la requête suivante :

```
SELECT titre, album
FROM Chanson
WHERE groupe = 'Guns N'Roses';
```

3. Écrire une requête SQL permettant d'obtenir tous les titres des chansons de l'album `Showbiz` dans l'ordre croissant.
4. Écrire une requête SQL permettant d'ajouter la chanson dont le titre est Megalomania de l'album Hullabaloo du groupe Muse.

INTERROS

5 Amélie a remarqué une faute de frappe dans la chanson Welcome too the Jungle qui s'écrit normalement Welcome to the Jungle. Écrire une requête SQL permettant de corriger cette erreur.

Partie B

Dans cette partie on utilise trois tables. Voici des extraits des trois tables `Chanson`, `Album` et `Groupe`.

Chanson		
id	titre	id_album
1	Sunburn	1
2	Muscle Museum	1
3	Showbiz	1
4	New Born	2
5	Sing for Absolution	4
6	Hysteria	4
7	Welcome to the Jungle	5
8	Muscle Museum	3
9	Showbiz	3

Album			
id	titre	année	id_groupe
1	Showbiz	1999	1
2	Origin of Symmetry	2001	1
3	Hullabaloo	2002	1
4	Absolution	2003	1
5	Appetite for Destruction	1987	2

Groupe	
id	nom
1	Muse
2	Guns N' Roses

1 Expliquer l'intérêt d'utiliser trois tables `Chanson`, `Album` et `Groupe` au lieu de regrouper toutes les informations dans une seule table.

2 Expliquer le rôle de l'attribut `id_album` de la table chanson.

3 Proposer alors un schéma relationnel pour cette version de la base de données. On pensera à bien spécifier les clés primaires en les soulignant et les clés étrangères en les faisant précéder par le symbole #.

4 Écrire une requête SQL permettant d'obtenir tous les noms des albums contenant la chanson Showbiz.

5 Écrire une requête SQL permettant d'obtenir tous les titres avec le nom de l'album des chansons du groupe Muse.

6 Décrire par une phrase ce qu'effectue la requête SQL suivante :

```
SELECT COUNT(*) AS tot
FROM Album AS a
JOIN Groupe AS g ON a.id_groupe = g.id
WHERE g.nom = 'Muse';
```

Partie C

Dans cette partie, on utilise Python. Amélie a remarqué que son professeur ne parle jamais d'ordre alphabétique mais d'ordre lexicographique lorsqu'il fait une requête avec ORDER BY.

Elle a compris qu'il s'agissait de l'ordre du dictionnaire mais elle se demande comment elle pourrait elle-même écrire une fonction ordre_lex (mot1, mot2) de comparaison entre deux chaînes de caractères en utilisant l'ordre lexicographique. La fonction

ordre_lex (mot1, mot2) prend en arguments deux chaines de caractères et renvoie un booléen. Une rapide recherche lui permet de trouver le résultat suivant :

Lorsque l'on compare deux chaînes de caractères suivant l'ordre lexicographique, on commence par comparer les deux premiers caractères de chacune des deux chaînes, puis en cas d'égalité on s'intéresse au second, et ainsi de suite. Le classement est donc le même que celui d'un dictionnaire. Si lors de ce procédé on dépasse la longueur d'une seule des deux chaînes, elle est considérée plus petite que l'autre. Lorsqu'on dépasse la longueur des deux chaînes au même moment, elles sont nécessairement égales

Amélie commence par écrire quelques assertions que sa fonction devra vérifier.

1 Compléter les assertions suivantes :

```
1 assert ordre_lex("", "a") == True
2 assert ordre_lex("b", "a") == ...
3 assert ordre_lex("aaa", "aaba") == ...
```

On suppose que les chaînes de caractères mot 1 et mot 2 ne sont composées que des lettres de l'alphabet, en minuscule, et la comparaison entre deux lettres peut se faire avec les opérateurs classiques == et <.

Par exemple :

```
1 >>> "" < "a"
2 True
3 >>> "b" == "a"
4 False
```

Enfin, le slice mot1 [1 :] renvoie la chaîne de caractère de mot1 privée de son premier caractère.

Par exemple :

```
1 >>> mot1 = "abcde"
2 >>> mot2 = mot1[1:]
3 >>> mot2
4 "bcde"
```

2 Recopier et compléter la fonction récursive ordre_lex ci-dessous qui prend pour paramètre deux chaînes de caractères mot1 et mot2 et qui renvoie True si mot 1 précède mot 2 dans l'ordre lexicographique.

```
def ordre_lex(mot1, mot2):
  if mot1 == "":
    return True
  elif mot2 == "":
    return False
  else:
    c1 = mot1[0]
    c2 = mot2[0]
    if c1 < c2:
      return ...
    elif c1 > c2:
      return ...
    else:
      return ...
```

3 Proposer une version itérative de la fonction `ordre_lex`.

9 Entreprise de location

★★★ 45 min Corrigé p. 268

Extrait du BAC 2024, Jour 1, Métropole

Cet exercice porte sur la programmation Python, les bases de données relationnelles et les requêtes SQL.

En particulier, les mots-clés suivants peuvent être utilisés : `SELECT`, `CREATE TABLE`, `FROM`, `WHERE`, `JOIN ON`, `INSERT INTO`, `VALUES`, `UPDATE`, `SET`, `COUNT`, `DELETE`, `DISTINCT`, `AND`, `OR`, `AS`.

On utilise une base de données relationnelle. Une entreprise de location de voitures propose à ses clients la possibilité de rapporter le véhicule dans une autre agence. Les informations correspondantes sont rangées dans une base de données. Voici les extraits de deux tables utilisées.

Agences					
Agence	Ville	CP	Departemen	Adresse	Telephone
Licorne	Paris	75001	Paris	123 Rue de la Ré-volution	0123456789
Le Carosse	Paris	75001	Paris	456 Virtual Street	0156789012
Vroum	Lyon	69001	Rhône	789 Virtual Lane	0456789034
Rapide	Toulouse	31000	Haute Garonne	321 Virtual Avenue	0567890123
Deep Place	Bordeaux	33000	Gironde	987 Virtual Road	0567890145

Voitures						
id_voiture	marque	modele	kilometrage	nombre_places	type	carburant
1	Renault	Clio	64022	5	Berline	Essence
2	Renault	Clio	50350	5	Berline	Essence
3	Dacia	Sandero	62031	5	Berline	Essence
4	Dacia	Sandero	58955	5	Berline	Essence
5	Dacia	Sandero	65779	5	Berline	Essence
6	Dacia	Sandero	56253	5	Berline	Essence
7	Renault	Clio	49660	5	Berline	Essence
8	Fiat	500	2545	4	Citadine	Electrique
9	Fiat	500	1953	4	Citadine	Electrique
10	Fiat	500	549	4	Citadine	Electrique

1 Donner pour la table Agences un type possible pour l'attribut CP qui indique le code postal.

La fonction `COUNT` permet de compter le nombre d'enregistrements et le mot-clé `DISTINCT` permet de ne pas prendre en compte les doublons.

2 Donner le résultat de la requête suivante pour les extraits des tables données :

```
SELECT COUNT(DISTINCT Telephone)
FROM Agences;
```

3 Expliquer à quelle condition l'attribut Telephone pourrait servir de clé primaire pour la table Agences.

Certaines agences ont décidé de partager un même standard téléphonique. On ajoute à la table Agences l'attribut id_agence qui est utilisé comme clé primaire.

La nouvelle table obtenue est donnée page suivante.

Agences						
id_agence	Agence	Ville	CP	Departemen	Adresse	Telephone
1	Licorne	Paris	75001	Paris	123 Rue de la Révolution	0123456789
2	Le Carosse	Paris	75001	Paris	456 Virtual Street	0123456789
3	Vroum	Lyon	69001	Rhône	789 Virtual Lane	0456789034
4	Rapide	Toulouse	31000	Haute Garonne	321 Virtual Avenue	0567890123
5	Deep Place	Bordeaux	33000	Gironde	987 Virtual Road	0567890145

Nous souhaitons créer une table `couple_voitures_agences` qui permettra d'associer la table Agences à la table voitures. On précise que la table voitures a pour clé primaire id_voiture.

4. Décrire les attributs de cette nouvelle table ; une clé primaire sera soulignée et une clé étrangère commencera par un dièse (#).
5. Écrire une requête qui permet d'enregistrer dans la table `couple_voitures_agences` le fait que le véhicule 2 est associé à l'agence Deep Place.
6. Écrire une requête qui permet d'actualiser la table pour indiquer que le véhicule 2 se trouve maintenant à l'agence Le Carosse.
7. Écrire une requête qui permet d'afficher le type, la marque et le nom de l'agence pour l'ensemble des véhicules.

10 Gestion d'un camping municipal

★★★ 45 min Corrigé p. 268

D'après BAC 2024, Centres Étrangers Afrique, Jour 1

Cet exercice porte sur la programmation Python, la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.

L'objectif est de faciliter la gestion du système d'information d'un camping municipal. Les informations nécessaires sont stockées dans une base de données relationnelle composée de trois relations. On pourra utiliser les mots-clés SQL suivants : `AND`, `FROM`, `INSERT`, `INTO`, `JOIN`, `ON`, `SELECT`, `SET`, `UPDATE`, `VALUES`, `WHERE`.

Voici le schéma des deux premières relations :

```
Client (id_client, nom, prenom, adresse, ville, pays,
telephone)
Reservation (id_reservation, #id_client, #id_emplacement,
nombre_personne, date_arrivee, date_depart)
```

Dans ce schéma :

- la clé primaire de chaque relation est définie par son attribut souligné ;
- les attributs précédés de # sont les clés étrangères.

La troisième relation est appelée `Emplacement` et elle contient tous les emplacements du camping. Le tableau ci-dessous en donne un extrait.

Emplacement			
`id_emplacement`	`nom`	`localisation`	`tarif_journalier`
1	myrtille	A4	25
2	mirabelle	D1	35
3	mangue	B2	29.90
4	mandarine	B1	25
5	mûre	C3	29.90
6	melon	A2	25

1 Citer deux avantages à utiliser une base de données relationnelle plutôt qu'un fichier texte ou un fichier tableur.

2 Quelle doit être la caractéristique d'un attribut pour pouvoir être utilisé en tant que clé primaire ?

3 Dans la relation `Reservation`, quel est le rôle des clés étrangères `id_client` et `id_emplacement` ?

4 Donner le schéma relationnel de la relation `Emplacement` en précisant la clé primaire et le type de chacun des attributs.

5 À partir de l'extrait du contenu de la relation `Emplacement` donner le résultat de la requête ci-dessous :

```
SELECT id_emplacement, nom, localisation
FROM Emplacement
WHERE tarif_journalier = 25;
```

6 Écrire une requête permettant de donner le nom et le prénom de tous les clients habitant à 'Strasbourg'.

7 Écrire une requête permettant d'ajouter un nouveau client :

- `id_client` : 42 ;
- `nom` : 'CODD' ;
- `prénom` : 'Edgar' ;
- `adresse` : '28 rue des Capucines' ;
- `ville` : 'Lyon' ;
- `pays` : 'France' ;
- `numéro de téléphone` : '0555555555'.

INTERROS

8 Écrire une requête SQL permettant de récupérer les informations ci-dessous concernant la réservation dont l'identifiant `id_reservation` est 18 :

- `Client.nom`
- `Client.prenom`
- `Reservation.nombre_personne`
- `Reservation.date_arrivee`
- `Reservation.date_depart`
- `Emplacement.tarif_journalier`

1 QCM Vérification de connaissances

Énoncé p. 239

1. Réponse **d**. En effet, « SELECT * FROM clients » signifie que l'on veut voir tous les champs de la table « clients » et « ORDER BY ville DESC » signifie que l'on souhaite effectuer un tri sur le contenu de « ville » mais dans un ordre alphabétique inverse (DESC).
2. Réponse **a**. « clients INNER JOIN entrepots » désigne l'intersection des deux tables et « ON clients.ville = entrepots.ville » désigne le critère à prendre en compte pour trouver l'intersection (ici, le nom des villes). « SELECT * » signifie que l'on souhaite afficher tout donc la requête affiche tous les champs des deux tables où le nom des villes coïncide.
3. Réponse **c**. La requête `SELECT * FROM clients,entrepots WHERE clients.ville = entrepots.ville` est explicite : on sélectionne tous les champs des deux tables « clients » et « entrepots » où le contenu des champs « ville » est commun. Elle fait donc exactement la même chose que la requête de la question précédente.
4. Réponse **a**. La requête `SELECT SUM(commandes.total)` renvoie un tableau d'une ligne et d'une colonne dont la valeur est la somme des entrées de la colonne « total ». Il suffit ensuite de regarder la condition (ici, « WHERE date > '2019-09-02' » ainsi que l'alias (ici, « AS somme », qui signifie que le nom de la somme sera accessible via l'appellation de `somme`).

2 V/F Vérification de connaissances

Énoncé p. 240

1. *Faux*. Dans une base de données, une ligne d'une table est appelée un *enregistrement*.
2. *Vrai*. Les étiquettes des colonnes d'une table sont appelées les attributs de la table.
3. *Faux*. Une clé primaire a pour but de repérer de manière unique un enregistrement. On ne peut donc pas prendre un attribut (ici « nom ») qui peut prendre plusieurs fois la même valeur. Il serait ici préférable de prendre « idClient » comme clé primaire, si cet attribut a été créé comme étant un entier auto-incrémenté (par exemple).
4. *Vrai*. En effet, toute clé primaire d'une table, recopiée en tant qu'attribut d'une autre table, est appelé *clé étrangère* de cette dernière.

CORRIGÉS

3 Festival de musique

Énoncé p. 241

Lycée André Theuriet, Civray

1 La liste des titres des représentations est donnée par la requête :

```
SELECT titreRep
FROM Representation
```

2 La liste des titres des représentations ayant lieu sur le lieu nommé « Dionysos » est donnée par la requête :

```
SELECT
    titreRep
FROM
    Representation
WHERE
    lieu = "Dionysos"
```

3 La liste des noms des musiciens et les titres des représentations auxquelles ils participent est donnée par la requête :

```
SELECT
    M.nom , R.titreRep
FROM
    Musicien M
JOIN
    Representation R
ON
    R.idRep = M.idRep
```

4 La liste des titres des représentations, les lieux et les tarifs d'une date précisée (par exemple le 24/12/2020) est donnée par la requête :

```
SELECT
    R.titreRep ,
    R.lieu ,
    P.tarif
FROM
    Programmer P
JOIN
    Representation R
ON
    P.idRep = R.idRep
WHERE
    P.date = "2020-12-24"
```

5 Le nombre des musiciens qui participent à la représentation numéro 7 est donné par la requête :

```
SELECT
    COUNT (*)
FROM
    Musicien
WHERE
    idRep = 7
```

6 Les représentations et leurs dates dont le tarif ne dépasse pas 30 € sont données par la requête :

```
SELECT
    R.idRep ,
    R.titreRep ,
    P.Date
FROM
    Representation R
JOIN
    Programmer P
ON
    R.idRep = P.idRep
WHERE
    P.tarif <= 30
```

4 Les employés du département

Énoncé p. 241

Lycée Henri Alain-Fournier, Bourges

1 La liste des employés ayant une commission :

```
SELECT
    *
FROM
    Employes
WHERE
    COMM = 1
```

2 **MÉTHODE**

Pour obtenir une liste de résultats selon un ordre *croissant* ou *décroissant*, on utilise une requête avec le mot-clé `ORDER BY ... ASC` (pour un ordre croissant) et `ORDER BY ... DESC` (pour un ordre décroissant).

Les noms, emplois et salaires des employés par emploi croissant, et pour chaque emploi, par salaire décroissant :

```
SELECT
    ENOM ,
    PROF ,
    SAL
FROM
    Employes
ORDER BY
    PROF ASC ,
    SAL DESC
```

3 Le salaire moyen des employés :

```
SELECT
    AVG ( SAL )
FROM
    Employes
```

4 Le salaire moyen du département « Production » :

```
SELECT
    AVG ( E.SAL )
FROM
    Employes E
JOIN
    Departements D
ON
    E.DNO = D.DNO
WHERE
    D.DNOM = "Production"
```

5 Notes annuelles des étudiants

Énoncé p. 242

Lycée Sainte-Clotilde, Strasbourg

1 Le nombre total d'étudiants est donné par la requête :

```
SELECT
    COUNT(*)
FROM
    ETUDIANT
```

2 Parmi l'ensemble des notes, la note la plus haute et la note la plus basse sont données par la requête :

```
SELECT
    MIN(Note) AS minimum,
    MAX(Note) AS maximum
FROM
    EVALUER
```

3 Les moyennes de chaque étudiant dans chacune des matières sont données par la requête :

```
SELECT
  ETU.NumEtudiant,
  ETU.Nom,
  ETU.Prenom,
  MAT.LibelleMat,
  MAT.CoeffMat,
  AVG(EVA.Note) AS MoyenneMat
FROM
  ETUDIANT AS ETU
  JOIN EVALUER AS EVA
    ON EVA.NumEtudiant = ETU.NumEtudiant
  JOIN MATIERE AS MAT
    ON MAT.CodeMat = EVA.CodeMat
GROUP BY
  ETU.NumEtudiant,
  ETU.Nom,
  ETU.Prenom,
  MAT.LibelleMat,
  MAT.CoeffMat
ORDER BY
  ETU.Nom
```

4 Les moyennes de la classe par matière peuvent être données par la requête :

```
SELECT
  MAT.CodeMat ,
  MAT.LibelleMat ,
  AVG(EVA.Note) AS MoyClasse
FROM
  MATIERE AS MAT
  INNER JOIN EVALUER AS EVA
    ON EVA.CodeMat=MAT.CodeMat
```

(suite du script page suivante)

```
GROUP BY
  MAT.CodeMat,
  MAT.LibelleMat
ORDER BY
  MAT.LibelleMat
```

5

```
SELECT
    ETU.Nom,
    ETU.Prenom,
    SUM(MAT.CoeffMat * AVG(EVA.Note)) / SUM(MAT.
    CoeffMat) AS moyenne
FROM
    ETUDIANT AS ETU
JOIN
    EVALUER AS EVA ON EVA.NumEtudiant = ETU.
    NumEtudiant
JOIN
    MATIERE AS MAT ON MAT.CodeMat = EVA.CodeMat
GROUP BY
    ETU.Nom,
    ETU.Prenom;
```

6

```
SELECT
    AVG(moyenne)
FROM (
    SELECT
        ETU.Nom,
        ETU.Prenom,
        SUM(MAT.CoeffMat * AVG(EVA.Note)) / SUM(MAT.
    CoeffMat) AS moyenne
    FROM
        ETUDIANT AS ETU
    INNER JOIN
        EVALUER AS EVA ON EVA.NumEtudiant = ETU.
    NumEtudiant
    INNER JOIN
        MATIERE AS MAT ON MAT.CodeMat = EVA.CodeMat
    GROUP BY
        ETU.Nom,
        ETU.Prenom
) AS SousRequete;
```

6 Un cas pratique : site web marchand

Énoncé p. 243

Lycée Jean-Joseph Fourier, Auxerre

1 Un fichier SQL permettant de créer les quatre tables est le suivant :

```
CREATE TABLE users (
    idUser BIGINT PRIMARY KEY NOT NULL
    AUTO_INCREMENT ,
    nom VARCHAR (30) ,
    prenom VARCHAR (30) ,
    email VARCHAR (50) ,
    rue VARCHAR (100) ,
    cp VARCHAR (10) ,
    ville VARCHAR (50) ,
    tel VARCHAR (12) );

CREATE TABLE IF NOT EXISTS modeles (
    idModele BIGINT PRIMARY KEY NOT NULL
    AUTO_INCREMENT ,
    nomModele VARCHAR (20) );

CREATE TABLE IF NOT EXISTS articles (
    idArticle BIGINT PRIMARY KEY NOT NULL
    AUTO_INCREMENT ,
    nom VARCHAR (50) ,
    descriptif VARCHAR (255) ,
    idModele BIGINT ,
    prix DOUBLE ,
    FOREIGN KEY (idModele) REFERENCES modeles(
    idModele) );

CREATE TABLE IF NOT EXISTS panier (
    idPanier BIGINT PRIMARY KEY NOT NULL
    AUTO_INCREMENT ,
    idUser BIGINT,
    idArticle BIGINT ,
  idModele BIGINT ,
    quantite INT ,
    total DOUBLE ,
    FOREIGN KEY (idUser) REFERENCES users(idUser) ,
  FOREIGN KEY (idModele) REFERENCES modeles(
    idModele) ,
    FOREIGN KEY (idArticle) REFERENCES articles(
    idArticle) )
```

La taille des colonnes est ici arbitraire : nous les avons choisi en essayant d'être cohérent.

Pour ce qui est de la colonne « tel », nous avons fait le choix de la déclarer comme chaîne de caractères (pour que le format des numéros soient de la forme `+336XXXXXXXX` ou `+337XXXXXXXX`).

Pour les clés primaires, nous avons fait le choix du format « BIGINT » car nous sommes optimistes... et souhaitons beaucoup d'entrées pour cette base de données, qui signifierait un grand succès du site.

2 Pour cette question, il faut anticiper : il faut avant tout enregistrer les différents modèles cités (Unique, S, M, L, XL et XXL). Ce n'est qu'ensuite que l'on pourra enregistrer les articles. On a alors :

```
INSERT INTO
    modeles (nomModele)
VALUES
    ( 'Unique' ), ( 'S' ),
    ( 'M' ), ( 'L' ),
    ( 'XL' ), ( 'XXL' );

INSERT INTO
    articles (nom,descriptif,idModele,prix)
VALUES
    (   'Mug' ,
        'Magnifique mug personnalisable avec votre
    photo.' ,
        1 ,
        '7.99' ),
    (   'T-Shirt "I Kiffe Beef"' ,
        'T-shirt unique en son genre.' ,
        2 ,
        '24.99' ),
    (   'T-Shirt "I Kiffe Beef"' ,
        'T-shirt unique en son genre.' ,
        3 ,
        '24.99' ),
      (
        'T-Shirt "I Kiffe Beef"' ,
        'T-shirt unique en son genre.' ,
        4 ,
        '24.99'
    ),
```

(suite du fichier page suivante)

```
    (
        'T-Shirt "I Kiffe Beef"' ,
        'T-shirt unique en son genre.' ,
        '5 ,
        '24.99'
    ),
    (
        'T-Shirt "I Kiffe Beef"' ,
        'T-shirt unique en son genre.' ,
        6 ,
        '24.99'
    )
```

3 **Insertion des utilisateurs**

```
INSERT INTO
    users (nom,prenom,email,rue,cp,ville,tel)
VALUES
    (
        'Dupont' ,
        'Jean' ,
        'jdupont@free.fr' ,
        '3 rue des tulipes' ,
        '75000' ,
        'Paris' ,
        '+33601020304'
    ) ,
    (
        'Aimaun' ,
        'Anne' ,
        'amz@free.fr' ,
        '1 rue du glas' ,
        '33000' ,
        'Bordeaux' ,
        '+33799989796'
    )
```

4 **(a)** **Insertion des commandes dans le panier**

```
INSERT INTO
    panier
    (idUser,idArticle,idModele,quantite,total)
```

(suite page suivante)

```
VALUES
    ( '1' , '1' , '2' , '15.98' ) ,
    ( '1' , '2' , '1' , '24.99' ) ,
    ( '1' , '3' , '1' , '24.99' ) ,
    ( '1' , '4' , '1' , '24.99' ) ,
    ( '2' , '1' , '1' , '24.99' ) ,
    ( '2' , '3' , '1' , '24.99' )
```

(b) La requête permettant d'afficher le total du panier de Jean Dupont est la suivante :

```
SELECT
    SUM(total)
FROM
    panier
WHERE
    idUser = 1
```

(c)

```
SELECT
    SUM(PAN.quantite * ART.prix) AS TOTAL
FROM
    panier AS PAN
    JOIN articles AS ART
        ON ART.idArticle = PAN.idArticle
WHERE
    PAN.idUser = 1;
```

7 Voyages dans l'espace

Énoncé p. 244

Lycée Richelieu, Versailles

1 **(a)** Une clé primaire est un attribut dont la valeur permet d'identifier de manière unique un t-uplet de la relation.

(b) La valeur « 3 » a déjà été utilisé pour l'attribut `id_astronaute` de la table `Astronaute`. Nous allons donc avoir une erreur puisque `id_astronaute` est la clé primaire de la table `Astronaute`.

(c) Le schéma relationnel de la table `Fusee` est le suivant :

```
Fusee (id_fusee : INT, modele : TEXT,
    constructeur : TEXT, nb_places : INT)
```

2 **(a)** Le résultat de la requête est : « 2 ». En effet, cette requête compte le nombre d'entrées de la table `Fusee` où l'attribut `constructeur` vaut « SpaceX », et il y a deux entrées concernées.

(b) Une requête qui renvoie le modèle et le constructeur des fusées ayant au moins quatre places est :

```
SELECT modele, constructeur
FROM Fusee
WHERE nb_places > 3
```

(c) Une requête qui renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom est :

```
SELECT nom, prenom
FROM Astronaute
ORDER BY nom
```

3 **(a)** La requête complétée est :

```
INSERT INTO Vol VALUES(5, 3, '12/04/2023');
INSERT INTO Equipe VALUES(5, 1);
INSERT INTO Equipe VALUES(5, 4);
```

(b) Une requête SQL permettant d'obtenir le nom et le prénom des astronautes ayant décollé le 25/10/2022 est :

```
SELECT nom, prenom
FROM Equipe
JOIN Vol ON Vol.id_vol = Equipe.id_vol
JOIN Astronaute ON Astronaute.id_astronaute =
    Equipe.id_astronaute
WHERE DATE = '25/10/2022'
```

8 Collection de CD

Énoncé p. 247

D'après Bac 2024, Asie 2, Jour 2

Partie A

1 L'attribut `titre` ne peut pas être une clé primaire car il est possible qu'il y ait deux titres identiques.

2 La requête donne le résultat suivant :

Welcome too the Jungle	Appetite for Destruction

3 Une requête SQL est la suivante :

```
SELECT titre
FROM Chanson
WHERE album = 'Showbiz' ORDER BY id
```

4 Une requête SQL répondant à la question est la suivante :

```
INSERT INTO Chanson
VALUES (10, 'Megalomania', 'Hullabaloo', 'Muse')
```

5 Une requête SQL répondant à la question est la suivante :

```
UPDATE Chanson
SET titre = 'Welcome to the Jungle'
WHERE titre = 'Welcome too the Jungle'
```

Partie B

1 Le regroupement des données en une seule table entrainerait une duplication des données, d'où l'intérêt d'utiliser trois tables.

2 L'attribut `id_album` est une clé étrangère qui permet de lier la table `Chanson` à la table `Album`.

3 Un schéma relationnel pour cette version de la base de données est :

```
Chanson (id : int, titre : str, #id_album : int)
Album (id : int, titre : str, année : int,
#id_groupe : int)
Groupe (id : int, nom : str)
```

4 Une requête possible est la suivante :

```
SELECT Album.titre
FROM Album
JOIN Chanson ON id_album = Album.id
WHERE Chanson.titre = 'Showbiz'
```

5 Une requête possible est la suivante :

```
SELECT Chanson.titre, Album.titre
FROM Album
JOIN Chanson ON id_album = Album.id
JOIN Groupe ON id_groupe = Groupe.id
WHERE nom = 'Muse'
```

6 Cette requête permet de déterminer le nombre d'albums du groupe Muse présent dans la base de données.

Partie C

1 Les assertions complétées sont :

```
assert ordre_lex("", "a") == True
assert ordre_lex("b", "a") == False
assert ordre_lex("aaa", "aaba") == True
```

2 Le programme complété est le suivant :

```
def ordre_lex(mot1, mot2):
  if mot1 == "":
    return True
  elif mot2 == "":
    return False
  else:
    c1 = mot1[0]
    c2 = mot2[0]
    if c1 < c2:
      return True
    elif c1 > c2:
      return False
    else:
      return ordre_lex(mot1[1:], mot2[1:])
```

3 Une version itérative est la suivante :

```
def ordre_lex(mot1, mot2):
    if mot1 == "":
        return True
    if mot2 == "":
        return False

    i = 0
    while i < len(mot1) and i < len(mot2):
        c1 = mot1[i]
        c2 = mot2[i]
        if c1 < c2:
            return True
        elif c1 > c2:
            return False
        else:
            i = i + 1

    return len(mot1) <= len(mot2)
```

CORRIGÉS

9 Entreprise de location

Énoncé p. 250

Extrait du BAC 2024, Jour 1, Métropole

1. L'attribut `CP` peut être un `INT` ou un `TEXT (STR)`.
2. Dans la mesure où tous les numéros de téléphones figurant dans la table sont tous distincts, cette requête renvoie le nombre de lignes de cette table, donc « 5 ».
3. Afin que l'attribut `Telephone` constitue une clé primaire, il est nécessaire que toutes les agences aient un numéro de téléphone différent.
4. Le schéma relationnel de la table est :

```
couple_voitures_agences (
    # id_agence : INT,
    # id_voiture : INT)
```

5. Une requête qui permet d'enregistrer dans la table `couple_voitures_agences` le fait que le véhicule 2 est associé à l'agence Deep Place est :

```
INSERT INTO couple_voitures_agences
VALUES (5, 2)
```

6. Une requête qui permet d'actualiser la table pour indiquer que le véhicule 2 se trouve maintenant à l'agence Le Carrosse est :

```
UPDATE couple_voitures_agences
SET id_agence = 2
WHERE id_voiture = 2
```

7. Une requête qui permet d'afficher le type, lamarque et le nomde l'agence pour l'ensemble des véhicules est :

```
SELECT type, marque, Agence
FROM couple_voitures_agences
JOIN Agences ON couple_voitures_agences.id_agence =
     Agences.id_agence
JOIN Voitures ON couple_voitures_agences.id_voiture
     = Voitures.id_voiture
```

10 Gestion d'un camping municipal

Énoncé p. 252

D'après BAC 2024, Centres Étrangers Afrique, Jour 1

1. Nous allons donner trois avantages :
   - la gestion efficace des relations entre les données grâce aux clés primaires et étrangères ;

- la non-duplication d'informations ;
- la possibilité d'effectuer des requêtes complexes pour extraire et manipuler les données de manière flexible.

2 Un attribut utilisé en tant que clé primaire doit être unique pour chaque enregistrement de la table et ne doit pas contenir de valeurs nulles.

3 Les clés étrangères `id_client` et `id_emplacement` permettent de créer des relations entre les tables `Reservation`, `Client` et `Emplacement`, assurant ainsi l'intégrité référentielle des données.

4 Le schéma relationnel de la relation `Emplacement` est :

```
Emplacement (
    id_emplacement : INT,
    nom : VARCHAR(50),
    localisation : VARCHAR(10),
    tarif_journalier : DECIMAL(5,2)
)
```

5 Le résultat de la requête est :

id_emplacement	nom	localisation
1	myrtille	A4
4	mandarine	B1
6	melon	A2

6 La requête pour obtenir le nom et le prénom de tous les clients habitant à 'Strasbourg' est :

```
SELECT
  nom, prenom
FROM
  Client
WHERE
  ville = 'Strasbourg';
```

7 La requête pour ajouter un nouveau client est :

```
INSERT INTO Client
  (id_client, nom, prenom, adresse, ville, pays,
    telephone)
VALUES
  (42, 'CODD', 'Edgar', '28 rue des Capucines', '
    Lyon', 'France', '0555555555');
```

8 La requête SQL pour récupérer les informations concernant la réservation dont l'identifiant `id_reservation` est 18 est :

```
SELECT
  c.nom,
  c.prenom,
  r.nombre_personne,
  r.date_arrivee,
  r.date_depart,
  e.tarif_journalier
FROM
  Reservation r
JOIN
  Client c ON r.id_client = c.id_client
JOIN
  Emplacement e ON r.id_emplacement = e.
    id_emplacement
WHERE
  r.id_reservation = 18
```

Chapitre 8

Architecture matérielle et processus

Plan du chapitre

1 Architecture matérielle

1.1 Les composants fondamentaux

Un appareil numérique moderne (smartphone, ordinateur, console de jeux) repose sur des composants essentiels. Ces derniers sont interconnectés pour exécuter des tâches variées.

- **Processeur (CPU)** : responsable de l'exécution des instructions. Son architecture inclut plusieurs cœurs et une mémoire cache optimisant les performances.
- **Mémoire vive (RAM)** : stocke temporairement les données et instructions en cours d'utilisation. Son accès est bien plus rapide que celui du stockage permanent.
- **Stockage** : permet la conservation des données sur des supports variés : SSD (rapide, sans pièce mécanique) et HDD (capacité importante).
- **Carte mère** : connecte et assure la communication entre les composants via des bus de données et des interfaces comme PCIe ou SATA.
- **Périphériques** : dispositifs d'entrée (clavier, souris) et de sortie (écran, imprimante) facilitent l'interaction avec l'utilisateur.
- **Contrôleur vidéo et accélérateur graphique** : nommé généralement GPU (Graphics Processing Unit), il s'agit d'un processeur à part entière et de son microenvironnement minimal (mémoire de travail, bus de communication) qui composent le contrôleur graphique.

1.2 Les SoCs (System on Chip)

Depuis les débuts de l'informatique, les composants électroniques n'ont cessé :

- de gagner en performances (augmentation de la puissance de traitement),
- de se miniaturiser,
- de devenir accessibles au plus grand nombre (diminution des coûts de production).

COURS

La miniaturisation a atteint de telles proportions qu'il est aujourd'hui possible de réunir au sein d'une seule puce électronique tous les composants vitaux d'un ordinateur (CPU, RAM, mémoire de stockage, bus de communications, contrôleurs, GPU, interconnexions,...). Ces « ordinateurs-puce » s'appellent des SoCs.

Définition 1

Un *SoC* (System On Chip) est un système intégré sur une unique puce électronique qui regroupe les différents composants d'un ordinateur tels que le processeur, la mémoire, les interfaces et bus de communication, etc.

Les avantages des SoCs sont multiples.

- *Faible encombrement* (juste une puce électronique), qui leur permet d'être implantés dans des appareils de taille réduite (téléphone, montre,...). L'avènement des smartphones est à l'origine de leurs succès.
- *Faible consommation énergétique d'un SoC due à la miniaturisation des composants.* Il y a moins de pistes que sur une carte mère, les distances sont raccourcies, il n'y a pas besoin de connecteur ni de câble et la tension d'alimentation peut être réduite. On obtient ainsi moins de dissipation thermique et de pertes par effet Joule.
- *Baisse des coûts de fabrication et d'industrialisation* car ils sont produits en très grande quantité et ne nécessitent pas d'assemblage manuel des éléments qui les composent. Tous les composants électroniques sont gravés en une seule fois au sein d'une unique puce.

Il y a cependant quelques inconvénients : les éléments qui composent un SoC étant gravés ensemble, il n'est pas possible de les faire évoluer individuellement. On ne pourra pas changer la « carte réseau » ni la « carte graphique » par exemple.

En conséquence, la durée de vie et la fiabilité d'un SoC sont celles de son maillon le plus faible. Le premier élément d'un SoC à présenter un dysfonctionnement provoquera une altération complète du SoC, qui ne pourra pas être réparé par remplacement de l'élément défaillant.

À RETENIR

- Un processeur multi-cœur exécute autant d'instructions par cycle qu'il a de cœur.
- La consommation électrique et l'échauffement augmentent avec la fréquence d'exécution.
- Tous les programmes en cours d'exécution et les données qu'il manipule sont recopiés et « travaillés » dans la RAM.
- Il existe un grand nombre de contrôleurs spécifiques, dédiés aux fonctionnalités assumées par la machine et au pilotage des périphériques. Ils déchargent le processeur (CPU) des opérations routinières.
- Un SoC (System on Chip) est une puce unique qui embarque tous les éléments de base d'un ordinateur.

2 Processus et ressources partagées

PROBLÉMATIQUE

Comment un système d'exploitation multitâche fait-il pour exécuter de nombreux programmes simultanément alors que la machine ne possède qu'un seul et unique processeur avec un nombre de cœurs très limité ?

2.1 Partage du temps d'exécution

Un ordinateur semble capable d'exécuter simultanément plusieurs programmes : une vidéo en lecture, un téléchargement en cours, un traitement de texte ouvert et une impression en arrière-plan. Pourtant, un processeur ne peut traiter qu'un nombre limité d'instructions en parallèle, correspondant au nombre de ses cœurs.

Pour donner l'illusion du multitâche, le système d'exploitation divise le temps de calcul du processeur en courtes séquences et alterne rapidement entre les programmes en cours. Ce mécanisme assure un partage efficace des ressources.

Afin d'assurer cette gestion, le système d'exploitation ne manipule pas directement les programmes, mais des processus, qui sont les unités d'exécution associées à chaque programme lancé.

2.1.1 - Processus

Définition 2

Un processus est une instance d'un programme en cours d'exécution, accompagnée de son contexte d'exécution : le code du programme, la mémoire allouée, ressources utilisées (fichiers, ports, etc.) et l'état des registres du processeur.

Le système d'exploitation identifie chaque processus par un nombre entier unique appelé *PID* (process identification, en anglais). À chaque processus est associé un *PPID* qui est le PID du processus parent qui a lancé le processus. Sous Linux, la commande `ps` permet d'afficher les processus associés au terminal courant. La commande `kill nnn` envoie un signal pour terminer le processus de PID `nnn`.

2.1.2 - État d'un processus

À un instant donné, un processus peut être dans l'un des états standards suivants :

- **nouveau** (*created* ou *new*) : en cours de création par le système d'exploitation. C'est un état transitoire par lequel le processus ne passe qu'une fois ;
- **prêt** (*ready* ou *runnable*) : il est en attente d'exécution ;
- **élu** ou **en exécution** (*running*) : le processus est en train de s'exécuter ;
- **en attente** (*blocked* ou *waiting*) : suspendu, en attente d'un événement externe ;
- **terminé** (*terminated*) : fin d'exécution. Le système d'exploitation est en train de désallouer les ressources utilisées. C'est aussi un état transitoire.

COURS

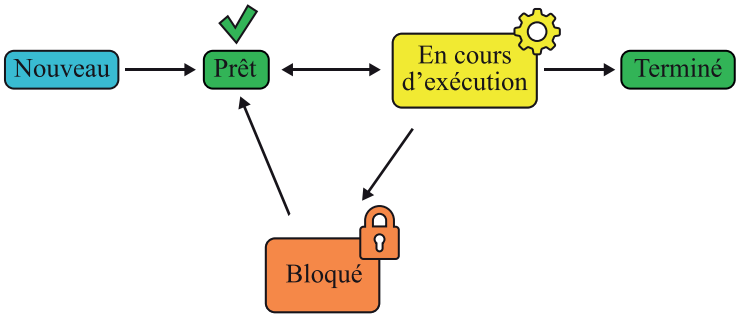


Figure 8.1 – Les différents états d'un processus dans un ordonnancement à court terme

Un processus alterne continuellement entre les états **prêt**, **en exécution** et **en attente**, selon les décisions du système d'exploitation.

2.2 Ordonnancement

L'ordonnancement (*scheduling*) des processus est un mécanisme essentiel au fonctionnement multitâche. Le système d'exploitation attribue le temps d'exécution aux processus en fonction de divers critères.

Définition 3 : ordonnanceur

L'**ordonnanceur** (*scheduler*) est le module du système d'exploitation responsable de la gestion des processus en cours d'exécution. Il décide quel processus obtient l'accès au processeur à un instant donné en fonction de divers critères comme les priorités, le temps d'exécution et l'état des processus (prêt ou élu).

Différents algorithmes d'ordonnancement existent :

- **non préemptifs** : un processus s'exécute jusqu'à son blocage ou sa fin ;
- **préemptifs** : l'ordonnanceur peut interrompre un processus pour exécuter un autre.

2.2.1 - Algorithmes non-préemptifs

Un algorithme d'ordonnancement non préemptif n'effectue aucune réquisition. Les processus rendent la main seulement « de leur plein gré », quand ils passent à l'état bloqué ou terminé, mais ne peuvent être interrompus par le système d'exploitation tant qu'ils sont à l'état élu (en cours d'exécution).

2.2.1.1 - First Come, First Served (FCFS)

C'est la stratégie du « premier arrivé, premier servi », qui fonctionne comme une file (FIFO). Les processus sont placés dans une file d'attente dans leur ordre d'arrivée et le premier lancé est exécuté en continu jusqu'à ce qu'il se termine ou de-

vienne bloqué. Le processus bloqué est replacé dans la file, en dernière position. C'est ensuite le processus prêt suivant dans la file qui s'exécute. Cette stratégie est simple à implémenter mais peut provoquer des attentes longues si un processus long est en tête de file.

2.2.1.2 - Shortest Job First (SJF)

Avec cette stratégie, on évalue la durée supposée de traitement de chacun des processus existants. Le processus qui a la durée d'exécution la plus courte est alors prioritaire, ce qui permet de réduire le temps d'attente moyen des processus. Toutefois, cette approche suppose une estimation fiable de la durée de chaque processus, ce qui n'est pas toujours réalisable.

Définitions 4

On appelle *temps d'attente* d'un processus la durée qui s'écoule entre la soumission du processus (arrivée) et le *début* de son exécution.
On appelle *temps de séjour* (ou temps de réponse) d'un processus la durée qui s'écoule entre la soumission du processus et la *fin* de son exécution.
De plus, on peut calculer un *temps d'attente moyen* et un *temps de séjour moyen* pour l'ensemble des processus.

2.2.2 - Algorithmes préemptifs

Un algorithme d'ordonnancement préemptif peut reprendre la main lors de l'exécution d'un processus et exécuter un autre processus (prioritaire sur le précédent ou plus court, etc.).

2.2.2.1 - Shortest Remaining Time (SRT)

C'est une évolution de la stratégie *Shortest Job First* où l'on tient compte cette fois, non pas de la durée totale (supposée) d'exécution de chaque processus, mais du temps restant estimé : à tout instant, le processus ayant le moins de temps restant à exécuter est choisi en priorité. Ainsi, un processus de durée d'exécution très courte peut devenir prioritaire alors qu'un processus dont il reste davantage de temps de traitement est déjà en train de s'exécuter.

2.2.2.2 - Tourniquet ou Round Robin (RR)

Dans l'algorithme du tourniquet, chaque processus dispose d'un quantum de temps. Chacun des processus est alors exécuté à tour de rôle, avec cette durée. Les processus prêts sont stockés dans une file (FIFO).

C'est l'algorithme le plus couramment utilisé dans les OS multitâches, principalement pour sa simplicité et sa fiabilité. En général, il est complété par un mécanisme d'ordonnancement avec priorité (*cf.* ci-après).

COURS

Définition 5 : quantum de temps

On appelle quantum de temps la plus petite unité de temps qu'un système d'exploitation peut attribuer à un processus.

Cette durée, généralement fixe, définit l'intervalle de temps minimal durant lequel un processus peut s'exécuter avant une éventuelle interruption. Un processus peut se voir allouer plusieurs quanta consécutifs. Typiquement, un quantum de temps dure entre 20 et 50 ms suivant l'OS et l'algorithme d'ordonnancement utilisé.

2.2.3 - Ordonnancement avec priorité

Chaque processus se voit attribuer une valeur de priorité, cette priorité pouvant évoluer de façon dynamique. Les processus de même niveau de priorité sont regroupés dans une file (FIFO) et ordonnancés avec l'algorithme du tourniquet (*Round Robin*). L'ordonnanceur sélectionne alors le processus de priorité la plus élevée et qui est en tête de sa file. Pour éviter qu'un processus prioritaire ne monopolise indéfiniment le processeur, la priorité des processus en cours d'exécution est périodiquement diminuée, tandis que la priorité des processus en attente peut être augmentée.

2.3 Interblocage (deadlock)

Supposons deux processus concurrents *P1* et *P2*. Ils ont accès à deux ressources, A et B, qui ne sont initialement utilisées par aucun processus.

Le processus *P1* accède alors à la ressource A (elle est alors verrouillée pour l'usage exclusif de *P1*). En suite, le processus *P2* accède à la ressource B (qui était libre et qui devient verrouillée).

Il est ensuite possible que le processus *P1* ait besoin d'accéder la ressource B qui est prise par *P2*. Dans ce cas, *P1* est *bloqué* jusqu'à ce que *P2* ait libéré la ressource B.

Supposons que le processus *P2* ait besoin d'accéder à la ressource A. Comme elle est actuellement verrouillée par *P1*, *P2* passe dans l'état *bloqué* jusqu'à ce que *P1* ait libéré la ressource A. Sauf que *P1* est déjà bloqué et ne peut être débloqué tant que *P2* n'a pas terminé.

Cette situation provoque un blocage réciproque de *P1* et de *P2* qui s'attendent mutuellement. On appelle cette situation un interblocage. C'est une situation que l'on cherche évidemment à éviter.

Définition 6 : interblocage

On appelle *interblocage* (deadlock en anglais) un état obtenu en programmation concurrente lorsqu'au moins deux processus s'attendent mutuellement.

Architecture matérielle

1 QCM Sur les SoCs

5 min Corrigé p. 282

Pour chacune des questions suivantes, plusieurs propositions de réponses sont faites dont une seule est réellement pertinente. Laquelle ?

1. Un Soc est :
   - a Une puce électronique ;
   - b Une sorte de micro-ordinateur sur une puce électronique ;
   - c Un système d'exploitation sur une puce électronique ;
   - d Une mémoire intégrée à une puce électronique.
2. La fluidité d'une vidéo sur un appareil à SoC dépend avant tout :
   - a de la mémoire embarquée dans le SoC ;
   - b du nombre de cœurs du CPU ;
   - c de la puce graphique (GPU) ;
   - d des ISP (Image Signal Processor).

Processus et ressources partagées

2 Processus et Linux ★★

15 min Corrigé p. 282

Lycée Richelieu, Rueil-Malmaison

1. Quelle est l'utilité de la commande `ps` ?
2. À quoi sert la commande `kill` ?
3. Avec une ligne de commande dans un terminal, on obtient l'affichage suivant :

```
smt@DEBVM-B3iS-VP0: $ ps o user,pid,ppid,tty,stat,command
```

USER	PID	PPID	TT	STAT	COMMAND
smt	4073	4070	pts/0	S	/bin/bash –login
smt	4114	4073	pts/0	S+	vi monitor.py
smt	4133	4070	pts/1	S	/bin/bash –login
smt	4152	4133	pts/1	S+	python3 monitor.py
smt	4156	4070	pts/2	S	/bin/bash –login
smt	5298	4156	pts/2	R+	ps -o user, pid, ppid,tty,stat, command

Voici un bref résumé de certaines caractéristiques des colonnes affichées :
- `USER` : nom d'utilisateur
- `TT` : numéro de terminal
- `STAT` : état du processus (S : interruptible sleep, R : running). Le signe + indique un processus exécuté en premier plan.

(a) Quel est le PID du processus parent de la commande `vi monitor.py` ?

(b) Le script `monitor.py` contient le code suivant :

```
name = input('Saisir votre nom :')
print('Bonjour', name, '!')
```

Quel est le PID correspondant à l'exécution de ce script Python ?

(c) Après avoir lancé ce programme, l'utilisateur `smt` a saisi son prénom mais n'a pas encore validé. Justifier le statut indiqué par la commande `ps` pour ce processus.

(d) Selon vous, quelles seront les conséquences visibles de l'exécution de la commande suivante ?

```
$ kill 4070
```

3 Processus et ressources

★★ 10 min Corrigé p. 282

Lycée Fresnel, Caen

Une machine exécute de manière concurrente quatre processus notés P1, P2, P3 et P4.

Le tableau ci-dessous donne l'état à un instant donné des différents processus qui peuvent soit mobiliser (M) des ressources (notées R1, R2, R3, R4 et R5), soit être en attente (A) des ressources, soit ne pas les solliciter (-).

Une ressource ne peut être mobilisée que par un seul processus à la fois. Si un autre processus demande une ressource déjà mobilisée, il passe en attente.

	R1	R2	R3	R4	R5
P1	M	-	-	A	-
P2	-	-	-	M	A
P3	-	M	A		-
P4	A	-	M	-	M

1. À partir du tableau ci-dessus, démontrer que, à cet instant, les processus s'attendent mutuellement.
2. Comment s'appelle cette situation ?
3. On suppose que le processus P1 libère la ressource R1. Donner un ordre possible d'exécution des processus.

Ordonnancement

4 Ordonnancement de processus

★★ 15 min Corrigé p. 283

Lycée Richelieu, Rueil-Malmaison

On considère cinq processus à ordonnancer, notés A, B, C, D et E. Les durées d'exécution et leurs dates d'arrivée respectives sont données dans le tableau ci-dessous. Les durées et dates sont exprimées en unités de temps (multiples de cycles d'horloge), une unité étant assimilable à un quantum de temps.

Processus	Durée d'exécution	Date d'arrivée
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

1. Quelle est la durée totale, en unités de temps, nécessaire pour l'exécution de ces cinq processus ?
2. Décrire en quelques lignes la différence entre les algorithmes Shortest Job First (SJF) et Shortest Remaining Time (SRT).
3. Donner le schéma d'exécution des algorithmes d'ordonnancement suivants :
   - First-Come First-Served (FCFS) ;
   - Shortest Job First (SJF) ;
   - Shortest Remaining Time (SRT) ;
   - Round Robin (RR) avec un quantum de 2.
4. Quelle est la différence entre le temps d'attente et le temps de séjour d'un processus ?
5. Calculer le temps d'attente moyen pour chacun de ces quatre algorithmes. Pour cette question et la suivante, il peut être intéressant de présenter au moins une partie des calculs faits. Un résultat juste dont le calcul n'est pas fourni n'entraîne pas de perte de points.
6. Calculer le temps de séjour moyen pour chacun de ces quatre algorithmes.
7. Quelle politique d'ordonnancement donne le meilleur résultat, c'est-à-dire celui correspondant à la durée minimale d'attente moyenne par processus ?

INTERROS

5 Algorithme d'ordonnancement

★★★ 30 min Corrigé p. 284

Lycée Fresnel, Caen

On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en unités de temps.

Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une liste d'attente en fonction de leur instant d'arrivée. On se propose d'ordonnancer ces quatre processus avec la méthode suivante :

- Parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la durée restante est la plus courte ;
- Le processeur exécute le processus élu durant une unité de temps puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;
- En cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;
- Tout ceci jusqu'à épuisement des processus en liste d'attente.

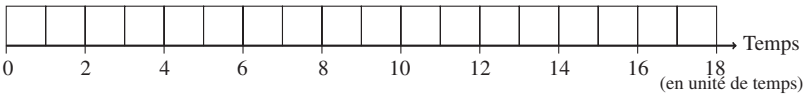
1 Quelle est la durée totale, en unité de temps, nécessaire à l'exécution des quatre processus.

2 En suivant l'algorithme ci-dessus, déterminer l'ordonnancement des 4 processus de l'exemple précédent.

3 On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée. Calculer la moyenne des temps d'exécution des quatre processus.

On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté précédemment mais au lieu de s'exécuter durant une seule unité de temps, le processeur exécutera à chaque fois le processus choisi durant deux unités de temps. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

4 Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



5 Calculer la nouvelle moyenne des temps d'exécution des 4 processus et indiquer si cet ordonnancement est plus performant que le précédent.

On se propose de programmer l'algorithme du premier ordonnanceur. Chaque processus sera représenté par une liste comportant autant d'éléments que de durées (en unités de temps).

Pour simuler la date de création de chaque processus, on ajoutera en fin de liste de chaque processus autant de chaînes de caractères vides que la valeur de leur date de création.

```
p1=['1.8','1.7','1.6','1.5','1.4','1.3','1.2','1.1'
    ]
p2=['2.6','2.5','2.4','2.3','2.2','2.1','','']
p3=['3.2','3.1','','','']
p4=['4.2','4.1','','','','','','','','']
liste_proc = [p1, p2, p3, p4]
```

La fonction `choix_processus` est chargée de sélectionner le processus dont le temps restant d'exécution est le plus court parmi les processus en liste d'attente.

6 Quel est le contenu de la liste d'attente à $t = 10$ (premier algorithme) ?

7 Recopier et compléter la fonction `choix_processus` ci-dessous. Le code peut contenir plusieurs lignes.

```
def choix_processus(liste_attente):
  """ Renvoie l'indice du processus le plus court
  parmi ceux présents dans liste d'attente"""
  if liste_attente != []:
    mini = len(liste_attente[0])
    indice = 0
    ...
    return indice
```

Une fonction `scrutation` (non étudiée) est chargée de parcourir la liste `liste_proc` de tous les processus et de renvoyer la liste d'attente des processus en fonction de leur arrivée. À chaque exécution de `scrutation`, les processus présents (sans chaînes de caractères vides en fin de liste) sont ajoutés à la liste d'attente. La fonction supprime pour les autres un élément de chaîne de caractères vides.

8 Recopier et compléter les différentes instructions de la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
def ordonnancement(liste_proc):
  """Exécute l'algorithme d'ordonnancement
  liste_proc -- liste des processus
  Renvoie la liste d'exécution des processus"""
  execution = []
  attente = scrutation(liste_proc, [])
  while attente != []:
    indice = choix_processus(attente)
    ... # À FAIRE (plusieurs lignes de code) ...
    attente = scrutation(liste_proc, attente)
  return execution
```

CORRIGÉS

1 QCM Sur les SoCs

Énoncé p. 277

1 Réponse **b**. Un SoC est un système intégré sur une puce qui regroupe l'essentiel d'un ordinateur : processeurs, mémoire, périphériques d'interface, etc.

2 Réponse **c**. En effet, les performances graphiques dépendent avant tout de la puce graphique. C'est elle qui gère les calculs des images donc il est nécessaire d'avoir un bon GPU pour que les calculs se fassent rapidement.

2 Processus et Linux

Énoncé p. 277

Lycée Richelieu, Rueil-Malmaison

1 La commande `ps` permet d'afficher la liste des processus en cours d'exécution (sans option, elle n'affiche que les processus lancés par l'utilisateur courant depuis le terminal courant).

2 La commande `kill` permet de terminer l'exécution d'un processus.

3 **(a)** Le PID du processus parent de `vi monitor.py` est son PPID : 4073.

(b) Le script Python `monitor.py` est exécuté par la commande `python3 monitor.py`, son PID est donc 4152.

(c) Le processus est en attente d'une saisie de l'utilisateur, il n'est donc plus en cours d'exécution mais « en attente ».

(d) La commande `kill 4070` va terminer le processus parent des trois terminaux (interpréteur de commandes bash), qui sont eux-mêmes les processus parents des trois autres commandes listées (`vi monitor.py`, `python3 monitor.py` et `ps`).

Par conséquent, les trois terminaux se fermeront.

3 Processus et ressources

Énoncé p. 278

Lycée Fresnel, Caen

1 Chacun des processus est en attente d'une ressource (donnée) qui est déjà mobilisée par un autre processus : P1 est en attente de R4, or R4 est déjà mobilisée par P2. P2 est en attente de R5 qui est mobilisée par P4, etc.

2 Cette situation où chaque processus est en attente d'une ressource mobilisée par un autre processus est appelée interblocage.

3 Lorsque R1 est libérée, P4 passe en exécution. S'il libère ensuite R3 et R5, alors P3 et P2 s'exécutent ensuite. Enfin, P1 attendait R4 qui est libérée par P2. D'où un ordre possible : P4 P3 P2 P1.

4 Ordonnancement de processus

Énoncé p. 279

Lycée Richelieu, Rueil-Malmaison

1 L'exécution de ces cinq processus nécessite au total $3+6+4+2+1 = 16$ cycles CPU.

2 La différence essentielle entre ces deux algorithmes est que SRT est préemptif, contrairement à SJF. L'avantage d'un algorithme préemptif est qu'il peut reprendre la main après unité de temps pour modifier le processus à exécuter.

3 Pour ces questions, en l'absence d'une spécification plus précise des algorithmes SRT et RR dans ce cours, il peut exister plusieurs schémas d'exécution possibles pour SRT entre les dates 6 et 10 (incluse) et pour RR à partir de la date 4. Ce corrigé présente une possibilité parmi plusieurs valides.

Date	0	1	2	3	4	5	6	7	8	9	10	11	12
FCFS	A	A	A	B	B	B	B	B	B	C	C	C	C
SJF	A	A	A	B	B	B	B	B	B	E	D	D	C
SRT	A	A	A	B	C	C	C	C	E	D	D	B	B
RR	A	A	B	B	A	B	B	C	C	B	B	C	C

Date	13	14	15	$< t_A >$	$< t_S >$
FCFS	D	D	E	4,4	7,6
SJF	C	C	C	3,2	6,4
SRT	B	B	B	1,2	5,8
RR	D	D	E	3,8	8,4

4 Le temps de séjour correspond au temps d'attente (durée écoulée entre la date d'arrivée et la date d'exécution) additionné du temps d'exécution (durée écoulée entre la date de début de l'exécution et la date de fin d'exécution).

5 On a le tableau suivant :

Algorithme	$t_A(A)$	$t_A(B)$	$t_A(C)$	$t_A(D)$	$t_A(E)$	$< t_A >$
FCFS	0	2	5	7	8	4,4
SJF	0	2	8	4	2	3,2
SRT	0	2	0	3	1	1,2
RR	0	1	3	7	8	3,8
Algorithme	$t_S(A)$	$t_S(B)$	$t_S(C)$	$t_S(D)$	$t_S(E)$	$< t_S >$
FCFS	3	8	9	9	9	7,6
SJF	3	8	12	6	3	6,4
SRT	3	15	4	5	2	5,8
RR	5	10	9	9	9	8,4

6 L'ordonnancement pour lequel le temps d'attente moyen est minimal est l'algorithme préemptif SRT.

CORRIGÉS

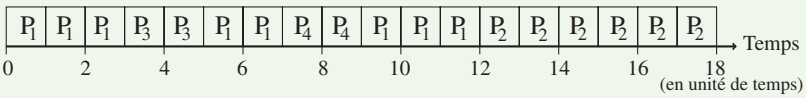
5 Algorithme d'ordonnancement

Énoncé p. 280

Lycée Fresnel, Caen

1 On additionne les quatre durées, il faut donc $\Delta t = 8 + 6 + 2 + 2 = 18$ unités de temps.

2 À $t = 0$, seul P1 est arrivé, il est donc exécuté. À $t = 2$, P1 et P2 sont dans la liste et leur durée restante est la même. C'est P1 qui est élu car le plus ancien. À $t = 3$, P3 arrive avec une durée restante plus courte que les deux autres, il est donc exécuté durant deux unités de temps. À $t = 5$, P1 a la durée restante la plus courte. Il est exécuté jusqu'à $t = 7$ où P4 arrive et est exécuté. À $t = 10$, il reste P1 (durée restante : 5) et P2 (durée restante 6). P1 est donc exécuté jusqu'à ce qu'il termine et enfin P2 est exécuté. On en déduit le chronogramme suivant :



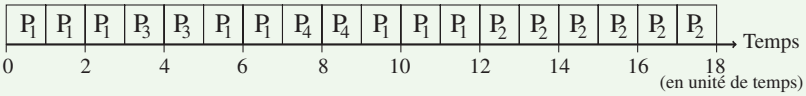
3

Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Instant de terminaison	12	18	5	9
Temps d'exécution	12	16	2	2

D'où un temps d'exécution moyen :

$$\Delta t_{moy} = \frac{12 + 16 + 2 + 2}{4} = 8$$

4



5

Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Instant de terminaison	10	18	6	12
Temps d'exécution	10	16	3	5

D'où le nouveau temps d'exécution moyen :

$$\Delta t_{moy} = \frac{10 + 16 + 3 + 5}{4} = 8{,}5$$

Cet algorithme d'ordonnancement est donc moins performant que le précédent.

6 La liste d'attente contient uniquement les processus arrivés et non encore terminés. À $t = 10$, il reste P1 et P2, d'où :

```
>>> liste_attente
[
 ['1.8','1.7'],
 ['2.6','2.5','2.4','2.3','2.2','2.1']
]
```

7

```
def choix_processus(liste_attente):
  """ Renvoie l'indice du processus le plus court
  parmi ceux présents dans liste d'attente"""
  if liste_attente != []:
    mini = len(liste_attente[0])
    indice = 0
    for i in range(1, len(liste_attente)):
      if len(liste_attente[i] < mini:
        indice = i
        mini = len(liste_attente[i]
    return indice
```

8

```
def ordonnancement(liste_proc):
  """Exécute l'algorithme d'ordonnancement
  liste_proc -- liste des processus
  Renvoie la liste d'exécution des processus"""
  execution = []
  attente = scrutation(liste_proc, [])
  while attente != []:
    indice = choix_processus(attente)
    """retrait de la liste d'attente du dernier
    élément du processus le plus court"""
    proc_elu = attente[indice][0]
    """ si ce processus est terminé alors
    retrait du processus de la liste d'attente"""
    if len(attente[indice]) == 0:
      attente.pop(indice)
    # ajoute l'élément à la liste d'exécution
    execution.append(proc_elu)
    attente = scrutation(liste_proc, attente)
  return execution
```

Chapitre 9

Réseaux et sécurité

Plan du chapitre

1 Protocoles de routage

1.1 Protocole IP

Internet est constitué d'un ensemble interconnecté de réseaux, au sein duquel les données doivent pouvoir circuler entre deux machines, indépendamment des réseaux locaux auxquels elles sont connectées. Ce transport des données repose sur la couche réseau du modèle TCP/IP, qui correspond à la couche 3 du modèle OSI. Son rôle est rempli par le protocole IP (Internet Protocol) dont les deux fonctions principales sont :

- l'*adressage* des machines pour les identifier de façon unique sur un réseau IP ;
- le *routage* des paquets entre les réseaux, en les acheminant de proche en proche vers leur destination, sans garantie de livraison.

1.1.1 - Adresses réseau

Une adresse IP est attribuée à chaque interface réseau d'un *hôte* (ou *machine* : ordinateur, routeur, objet connecté, etc.) sur un réseau qui utilise le protocole IP. Ainsi, un hôte possédant plusieurs interfaces réseau (Wi-Fi, Ethernet, etc.) sera identifié par plusieurs adresses IP distinctes, chacune correspondant à une interface spécifique.

Définition 1 : Adresse IP

Une *adresse IP* est un identifiant numérique unique attribué à chaque interface réseau d'un hôte connecté à un réseau informatique utilisant le protocole IP. Elle permet de localiser et d'acheminer les paquets vers l'interface concernée.

Elle peut être permanente (adresse IP statique) ou temporaire (allouée dynamiquement par un serveur DHCP).

Pour simplifier, nous parlerons souvent de l'« adresse IP d'une machine », bien qu'il s'agisse en réalité de l'adresse IP de l'interface qu'elle utilise sur un réseau donné.

COURS

Dans un réseau informatique, on distingue deux types d'adresses utilisées pour identifier une machine :

- l'*adresse IP*, qui est une adresse logique attribuée dynamiquement ou statiquement et qui permet de localiser un appareil dans un réseau IP.
- l'*adresse MAC*, qui est une adresse physique unique inscrite directement dans la carte réseau.

Définition 2 : Adresse MAC

Une *adresse MAC* (*Media Access Control*) correspond à une adresse physique, utilisée au niveau de la couche accès réseau. C'est un identifiant numérique unique de 48 bits (6 octets), représenté en hexadécimal (exemple : `2a-6e-64-7f-a9-e0`). Elle identifie de manière unique une carte réseau.

Lorsqu'un appareil veut envoyer des paquets sur un réseau local, il doit connaître l'adresse MAC du destinataire. Des mécanismes comme ARP (pour IPv4) permettent d'établir cette correspondance entre adresse IP et adresse MAC.

On distingue deux versions d'adresses IP :

- IPv4, codées sur 4 octets (32 bits), présentées sous la forme *décimale pointée* `a.b.c.d` où `a`, `b`, `c` et `d` sont les valeurs en base 10 des quatre octets. Exemple : 172.20.18.13 ;
- IPv6, codées sur 16 octets (128 bits), souvent représentées en notation hexadécimale. Exemple : `2a01:c9c0:a300:0800:0000:0004`.

L'espace d'adressage IPv4 est limité à environ 4,3 milliards d'adresses (2^{32}), ce qui pose problème avec l'augmentation du nombre d'appareils connectés. IPv6 offre un espace bien plus vaste (2^{128}), permettant une adresse unique pour chaque machine. Dans la suite du cours, nous nous concentrerons sur les adresses IPv4.

1.1.2 - Structure d'une adresse IP

Une adresse IP est constituée de deux parties :

- l'*adresse réseau* déterminée par la partie gauche de l'adresse IP, soit les n premiers bits de poids fort : identifie le réseau auquel appartient l'hôte ;
- le *numéro d'hôte* déterminé par la partie droite de l'adresse IP, soit les $32 - n$ bits de poids faible : identifie une machine au sein de ce réseau.

MÉTHODE

Pour bien visualiser la séparation entre la partie réseau et la partie hôte, il est conseillé d'écrire l'adresse IP sous la forme `w.x.y.z` où `w`, `x`, `y` et `z` sont les valeurs *binaires* des quatre octets.

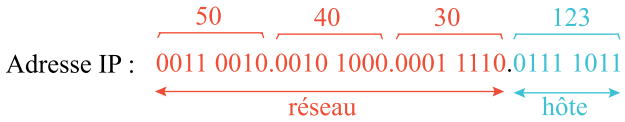


Figure 9.1 – Structure d'une adresse IPv4

Un *réseau local* (ou *sous-réseau*) est un ensemble de machines connectées physiquement et partageant une partie commune de leur adresse IP.

Définition 3 : Masque de réseau

Le *masque de réseau* (ou *masque de sous-réseau*, en anglais *subnet mask*), constitué d'une suite de 32 bits, possède des bits à 1 dans la partie réseau (bits de poids fort) et des bits à 0 dans la partie hôte. Il permet de délimiter la séparation entre l'adresse réseau et l'adresse hôte.

Deux adresses IP appartiennent au même réseau local si elles ont en commun les bits de la partie réseau définie par le masque.

Définition 4 : Notation CIDR

La *notation CIDR* (Classless Inter-Domain Routing) est une notation abrégée permettant d'exprimer la taille de la partie réseau d'une adresse IP. Elle s'écrit sous la forme `a.b.c.d/n`, où n indique le nombre de bits utilisés pour coder l'adresse réseau.

Par exemple, l'adresse 50.40.30.123/24 signifie que les 24 premiers bits définissent l'adresse du réseau et les 8 bits restants identifient les hôtes.

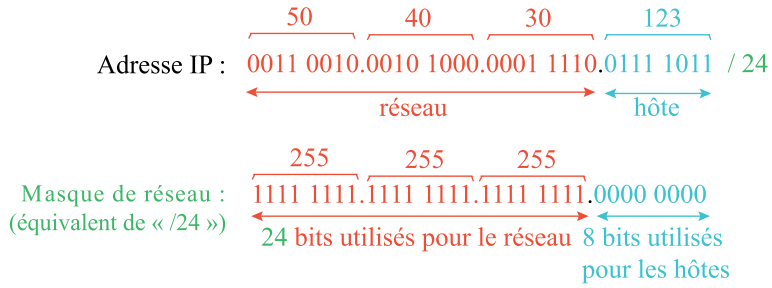


Figure 9.2 – Illustration du masque de réseau

Une fois la séparation réseau/hôte définie, on peut identifier deux adresses particulières dans chaque réseau :

Définition 5 : Adresse de réseau

L'*adresse de réseau* est l'adresse où tous les bits de la partie hôte sont mis à 0. Elle permet d'identifier le réseau et ne peut être attribuée à un hôte.

COURS

MÉTHODE

L'adresse de réseau peut être calculée en effectuant un ET bit à bit entre l'adresse IP et le masque de réseau.

Définition 6 : Adresse de diffusion

L'*adresse de diffusion* (broadcast) est l'adresse où tous les bits de la partie hôte sont mis à 1. Elle permet d'envoyer un message à toutes les machines du réseau simultanément.

Ces adresses sont réservées et ne peuvent pas être attribuées aux machines du réseau.

La taille de la partie hôte ($32 - n$ bits) permet de calculer le nombre d'adresses IP disponibles pour les hôtes de ce réseau. Sachant que deux adresses sont réservées, le nombre de machines adressables d'un réseau `a.b.c.d/n` est : $2^{32-n} - 2$.

Lorsqu'une machine doit envoyer un paquet à destination d'une adresse IP, elle commence par vérifier si cette adresse appartient à son réseau local :

- si c'est le cas, elle envoie le paquet directement au destinataire (avec une éventuelle résolution d'adresse MAC par ARP) ;
- sinon, elle l'envoie à une *passerelle*, un routeur connecté à au moins un autre réseau, qui se charge de relayer le paquet vers sa destination finale.

À RETENIR

La configuration d'un hôte sur un réseau local nécessite trois informations essentielles :

- *l'adresse IP de la machine* : identifie l'interface de l'hôte sur le réseau local.
- *le masque de réseau* : détermine quelle partie de l'adresse IP correspond au réseau et quelle partie identifie l'hôte. La combinaison de l'adresse IP et du masque permet de déterminer l'adresse du réseau local.
- *l'adresse IP de la passerelle* : c'est l'adresse d'un routeur présent sur le réseau local. Il sert d'intermédiaire pour envoyer les paquets destinés à des machines situées en dehors du réseau local.

POUR ALLER PLUS LOIN

Sur Internet, chaque machine connectée doit posséder une adresse IP unique. Cependant, avec la pénurie d'adresses IPv4, de nombreuses machines au sein d'un réseau local partagent une seule adresse IP publique grâce à un mécanisme appelé *NAT* (Network Address Translation).

...

POUR ALLER PLUS LOIN

Le NAT est souvent utilisé sur les routeurs domestiques et d'entreprise :

- il permet à plusieurs appareils du réseau local d'accéder à Internet en utilisant une seule adresse IP publique ;
- il protège indirectement les machines internes en cachant leurs adresses IP privées.

Avec IPv6, le NAT est moins nécessaire car l'espace d'adressage est suffisamment grand pour attribuer une adresse publique unique à chaque machine.

1.2 Principe du routage

PROBLÉMATIQUE

Comment les paquets IP trouvent-ils leur chemin sur un réseau aussi vaste et hétérogène qu'Internet ? Quelles stratégies permettent aux routeurs d'acheminer efficacement les paquets jusqu'à leur destination ?

1.2.1 - Routeur

Définition 7 : Routeur

Un routeur est un équipement connecté à au moins deux réseaux distincts, dont le rôle est d'acheminer les paquets d'un réseau à un autre en respectant certaines règles de routage.

Un routeur possède plusieurs *interfaces*, chacune étant connectée à un réseau spécifique. Chaque interface doit disposer d'une adresse IP correspondant au réseau auquel elle est rattachée.

Lorsqu'un paquet arrive sur l'une des interfaces d'un routeur, celui-ci doit « décider » où l'envoyer en fonction de l'adresse IP de destination. C'est là qu'intervient la *table de routage*, qui indique à chaque routeur comment transmettre les paquets vers le réseau approprié.

1.2.2 - Tables de routage

Définition 8 : Table de routage

La *table de routage* est une structure de données utilisée par un routeur pour stocker les informations sur les réseaux qu'il peut atteindre. Chaque entrée de la table associe un réseau de destination à l'interface ou au routeur intermédiaire à utiliser pour y accéder.

COURS

Lorsqu'un routeur reçoit un paquet IP, il consulte sa *table de routage* pour déterminer, en fonction de l'adresse IP destinataire, la meilleure route à emprunter. Si la destination est directement connectée au routeur, il transmet le paquet sur l'interface correspondante. Sinon, il l'envoie vers un autre routeur qui se trouve plus proche de la destination.

Les tables de routage peuvent être mises à jour de deux manières :

- routage statique : les entrées sont configurées manuellement par un administrateur. Ce mode est fiable mais peu flexible, car il nécessite une mise à jour manuelle à chaque modification du réseau.
- routage dynamique : la table de routage est mise à jour automatiquement à l'aide de protocoles de routage. Cette approche permet aux routeurs d'échanger des informations et d'adapter les itinéraires en fonction de l'état du réseau.

Considérons l'architecture réseau suivante qui présente quatre réseaux locaux interconnectés par le biais de quatre routeurs (R1, R2, R3 et R4) dont un (R4) est connecté à Internet (R4 peut être une box ADSL ou fibre par exemple).

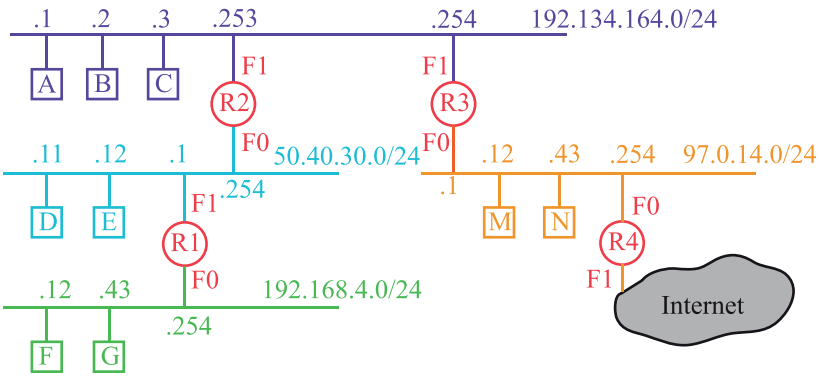


Figure 9.3 – Réseau à quatre routeurs

Lorsqu'un routeur démarre, sa table de routage ne contient par défaut que les réseaux directement connectés à ses interfaces. En l'absence de configuration, il ne sait pas encore comment joindre les autres réseaux.

Chaque entrée (ligne) de la table de routage correspond à un réseau et contient :

- l'*adresse du réseau* avec son *masque*, par exemple : 192.134.164.0/24 ;
- la *destination*, qui indique où envoyer le paquet :
  - si le réseau de destination est directement connecté, la destination est l'interface locale (ou port) à utiliser ;
  - si le réseau de destination est distant, la destination est l'adresse IP de la passerelle, c'est-à-dire l'adresse d'un autre routeur à qui transmettre le paquet, en précisant également l'interface par laquelle l'envoyer.

- la *métrique*, qui permet de quantifier la « distance » au destinataire final. La méthode de mesure utilisée dépend de l'algorithme de routage utilisé. Elle vaut 0 si le réseau est directement connecté au routeur.

Les tables de routages des quatre routeurs sont donc initialement :

Routeur	Réseau	Destination	Métrique
R1	192.168.4.0/24	Interface locale F0	0
	50.40.30.0/24	Interface locale F1	0
R2	50.40.30.0/24	Interface locale F0	0
	192.134.164.0/24	Interface locale F1	0
R3	97.0.14.0/24	Interface locale F0	0
	192.134.164.0/24	Interface locale F1	0
R4	97.0.14.0/24	Interface locale F0	0

Ces informations ne suffisent pas à router l'ensemble du réseau décrit ci-dessus. Par exemple, les machines F et A ne peuvent pas communiquer ensemble :

- La machine F envoie un paquet à destination de la machine A d'adresse 192.134.164.1. Le réseau local de F est 192.168.4.0 : A n'appartient pas à ce réseau. Elle envoie donc le paquet à sa passerelle par défaut, le routeur R1 à l'adresse 192.168.4.254 ;
- Le routeur R1 recherche une route pour atteindre le réseau 192.134.164.0/24. Or, sa table de routage ne contient aucune entrée correspondant à cette destination. Faute d'une route par défaut ou d'un protocole de routage dynamique, le paquet est abandonné.

Deux solutions permettent de résoudre ce problème :

- ajouter une route par défaut pour rediriger les paquets inconnus vers un routeur de sortie (comme R4, qui est connecté à Internet).
- utiliser un protocole de routage dynamique, comme RIP ou OSPF, permettant aux routeurs de partager automatiquement leurs tables de routage.

La mise à jour de la table de routage doit se faire périodiquement pour pouvoir prendre en compte les modifications du réseau (panne d'un routeur, ajout d'une liaison, ajout d'un routeur, etc.).

À part pour des petits réseaux, cette opération peut difficilement se faire de manière centralisée. Elle repose sur des algorithmes distribués (chaque routeur les exécute indépendamment des autres). Les échanges d'informations entre routeurs sont décrits dans des protocoles de routage.

1.3 Protocole RIP

1.3.1 - Principe

Le protocole RIP (*Routing Information Protocol*) est un des protocoles de routage dynamique mis en œuvre au niveau des routeurs afin d'acheminer les paquets.

COURS

Avec ce protocole, la métrique correspond au *nombre de sauts* (hops) entre le routeur et le réseau de destination.

Les routeurs transmettent automatiquement à leurs voisins les informations issues de leurs tables de routage (typiquement, une itération toutes les 30 secondes).

À la réception de ces informations, le routeur effectue les analyses suivantes :

- s'il trouve une nouvelle route (c'est-à-dire un nouveau réseau), il incrémente la métrique de 1 et si celle-ci est inférieure à 15, il l'ajoute à sa propre table de routage ;
- si la route existe déjà dans sa table :
  - soit la métrique reçue est inférieure à celle qu'il connaît : il supprime alors l'ancienne route et la remplace par la nouvelle ;
  - soit la métrique reçue est supérieure : si cette information vient du même routeur, alors il met à jour la métrique dans sa table.

La table contient donc, pour chaque réseau connu, l'adresse du routeur voisin dont la métrique est la plus petite. À chaque nouvelle itération, la table est ainsi mise à jour dynamiquement à chaque nouvelle itération du protocole.

Voici le résultat de l'application du protocole RIP sur l'architecture (figure 9.3) à l'issue des échanges de tables. La topologie réseau n'étant pas modifiée sur le temps analysé, les tables sont uniquement complétées au fur et à mesure des échanges RIP.

Routeur	Réseau	Destination	Métrique
R1	192.168.4.0/24	Interface locale F0	0
	50.40.30.0/24	Interface locale F1	0
	192.134.164.0/24	50.40.30.254 (R2)	1
	97.0.14.0/24	50.40.30.254 (R2)	2
R2	50.40.30.0/24	Interface locale F0	0
	192.134.164.0/24	Interface locale F1	0
	192.168.4.0/24	50.40.30.1 (R1)	1
	97.0.14.0/24	192.134.164.254 (R3)	1
R3	97.0.14.0/24	Interface locale F0	0
	192.134.164.0/24	Interface locale F1	0
	50.40.30.0/24	192.134.164.253 (R2)	1
	192.168.4.0/24	192.134.164.253 (R2)	2
R4	97.0.14.0/24	Interface locale F0	0
	192.134.164.0/24	97.0.14.1 (R3)	1
	50.40.30.0/24	97.0.14.1 (R3)	2
	192.168.4.0/24	97.0.14.1 (R3)	3

1.3.2 - Limites

Le protocole RIP présente des limites car :

- les routes présentant plus de 15 sauts sont ignorées ;
- la notion de « plus court chemin » fondée sur le nombre de sauts ne tient pas compte de la nature réelle des routes empruntées, et en particulier du débit qui les caractérise.

1.4 Protocole OSPF

Le protocole *OSPF* (pour Open Shortest Path First) est un autre type de protocole de routage. Il tient compte du débit des liaisons entre routeurs afin d'établir la route la plus rapide. Il ne tient pas compte du nombre de sauts.

Définitions 9

Le *débit* d'une liaison est la quantité d'information effectivement transmise par unité de temps (en nombre de bits par unité de temps, le plus souvent exprimée en kbit/s, Mbit/s, etc.)

La *bande passante* représente le débit maximal théorique d'une liaison (fonction de ses caractéristiques physique : câble, fibre optique, wi-fi, etc.).

Dans OSPF, on calcule le *coût* de chaque liaison entre deux routeurs. Ce coût est d'autant plus élevé que la bande passante est faible.

Définition 10 : Coût d'une liaison

Par convention, on utilise la relation suivante :

$$\text{coût} = \frac{\text{débit maximal de référence}}{\text{débit de la liaison}} = \frac{10^8}{\text{débit}}$$

les débits étant exprimés en bit/s. Le coût est sans unité.

Le coût d'une liaison tient lieu de métrique pour le protocole OSPF.

Le *coût d'une route* entre deux routeurs qui ne sont pas liés directement est la somme des coûts de toutes les routes à traverser.

Remarque : Le coût calculé est encodé par les routeurs sous la forme d'un entier compris entre 1 et 65 535. Ainsi, pour des liaisons à haute vitesse (supérieures à 100 Mbit/s), le coût calculé (inférieur à 1) est assimilé à 1. Pour pouvoir prendre en compte ces liaisons à haute vitesse, la bande passante de référence utilisée pour le calcul (10^8) peut être augmentée (10^9 ou 10^{10} par exemple). Si ce n'est pas le cas, sauf mention contraire dans l'énoncé, on conservera les valeurs calculées.

COURS

Le fonctionnement du protocole OSPF se décompose en deux étapes.

1. diffusion, entre voisins, de l'information sur le voisinage : les routeurs s'échangent des messages afin de reconstituer la topologie de leur zone ;
2. chaque routeur utilise un algorithme afin de calculer la meilleure route à emprunter pour joindre chacun des autres routeurs de la zone. Après calcul, chaque route est mémorisée dans la table de routage. Typiquement, le réseau peut être vu comme un graphe pondéré sur lequel on fait fonctionner un algorithme de recherche de plus court chemin (*Shortest Path First*) (*cf.* Dijkstra).

Exemple : considérons un réseau constitué de cinq routeurs, dont les débits entre chaque routeur sont :

- 1 Mbit/s entre R1 et R4 ;
- 10 Mbit/s entre R3 et R4, R3 et R5, R4 et R5 ;
- 100 Mbit/s entre R1 et R3, R1 et R2, R2 et R3 ;
- 1 Gbit/s entre R2 et R5.

.

Dans le protocole OSPF, chaque routeur peut se construire le graphe complet des liaisons inter-routeur :

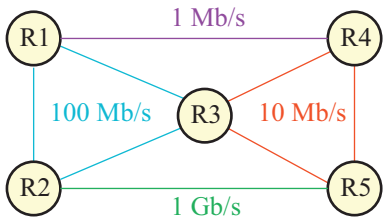


Figure 9.4 – Graphe correspondant au réseau

Considérons le cas où R1 a besoin de router un paquet vers R4 :

- S'il passe directement par son interface, ce qui serait le cas en appliquant le protocole RIP (métrique égale à 0) la transmission se fera avec un débit de 1 Mbit/s.
- S'il passe par R3 (route R1 – R3 – R4), la transmission sera dix plus rapide (le débit de la transmission globale sera limité par le débit le plus bas rencontré sur la route) : 10 Mbit/s sur R3 – R4.
- Dans cette configuration, les autres routes possibles (R1 – R3 – R5 – R4, R1 – R2 – R3 – R4, R1 – R2 – R5 – R4, R1 – R2 – R5 – R3 – R4) sont toutes limitées par le débit de 10 Mbit/s entre R3 (ou R5) et R4.

Le calcul de coût conduit au graphe pondéré suivant :

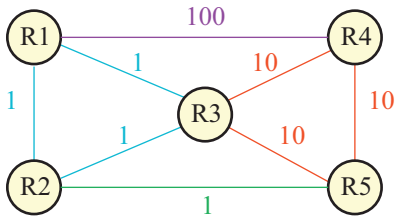


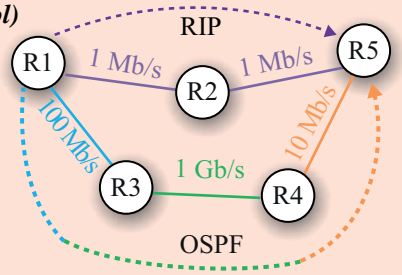
Figure 9.5 – Graphe représenté par chaque routeur

Dans le protocole OSPF, la route choisie minimise le coût total. Ainsi, c'est bien la route R1 – R3 – R4, de coût $1 + 10 = 11$ qui sera privilégiée.

À RETENIR

RIP (Routing Information Protocol)
Plus court chemin déterminé par le nombre de sauts
R1 → R2 → R5

OSPF (Open Short Path First)
Plus court chemin déterminé par le coût des routes
R1 → R3 → R4 → R5



2 Sécurisation des communications

Sur Internet, la protection des données sensibles échangées est un enjeu majeur. Lorsqu'un message transite sur un réseau, il peut être intercepté par un tiers malveillant. Le chiffrement permet de rendre ces messages inexploitables sans la clé de déchiffrement correspondante.

Définitions 11

La *cryptographie* est l'ensemble des techniques permettant de chiffrer et déchiffrer des messages. Le message à transmettre est en général appelé *message clair* ou « *en clair* ». Le message clair, une fois transformé par la technique de chiffrement est appelé *message chiffré*.
La *cryptanalyse* est l'ensemble des techniques visant à déchiffrer un message sans en être le destinataire légitime et donc sans en avoir la clé (on parle aussi de « *casser* » ou *décrypter* un message), voire sans connaître la méthode de chiffrement utilisée.

COURS

Depuis 1977 et l'article présentant le système RSA, on a l'habitude d'utiliser des personnages fictifs pour désigner les différents intervenants : Alice veut transmettre un message à son ami Bob. Elle ne souhaite pas qu'Ève, qui peut écouter les échanges, puisse déchiffrer les messages.

Une méthode de chiffrement repose sur un *algorithme* (de chiffrement et de déchiffrement) et sur une *clé*. La clé est une donnée (par exemple un nombre ou un texte) permettant de chiffrer et de déchiffrer un message en utilisant l'algorithme.

2.1 Chiffrement symétrique

Le chiffrement symétrique repose sur le *partage d'une clé* entre Alice et Bob. Cette clé partagée est *secrète*. Idéalement, elle doit être échangée hors du réseau ou être transmise de manière sécurisée (ce qui déplace le problème). La clé utilisée pour chiffrer est la même que celle utilisée pour déchiffrer, d'où le terme « *symétrique* ».

2.1.1 - Chiffre de César

Le chiffre de César est une méthode très ancienne fondée sur le décalage au sein d'un alphabet. Il permet de substituer une lettre par une autre lettre de l'alphabet, décalée dans cet alphabet d'une distance fixe (un nombre de lettres).

Dans le cas de l'alphabet latin (26 lettres), une clé de 4 correspond à une substitution de A par E, B par F, ..., V par Z, W par A, etc. (permutation circulaire de l'alphabet). Une clé égale à 0 (ou 26) ne modifie pas le texte.

Exemple :
Un décalage de 4 transforme « NSI » en « RWM ».

Bien que simple à implémenter, ce chiffrement est facilement cassable par une attaque exhaustive (énumération de toutes les possibilités) car il n'existe que 25 clés possibles.

2.1.2 - ROT13

L'algorithme ROT13 est un chiffrement de César dont la clé vaut 13. On en déduit qu'appliquer deux fois cet algorithme sur un texte en clair est équivalent à un décalage de 26, ce qui restitue le texte initial.

$$\text{NSI} \xrightarrow{\text{ROT13}} \text{AFV} \xrightarrow{\text{ROT13}} \text{NSI}$$

2.1.3 - Chiffre de Vigenère

Le chiffre de Vigenère est une méthode améliorée de chiffrement par décalage, qui utilise le chiffrement de César non plus sur une seule clé numérique mais en utilisant successivement plusieurs clés, de manière cyclique.

On peut alors présenter la clé sous forme d'une chaîne. Ainsi, la clé `'ALLO'` sera appliquée de telle manière que :

- le premier caractère du texte en clair est décalé de « A » soit 1 caractère ;
- le deuxième caractère est décalé de « L » soit 12 caractères ;
- idem pour le troisième (L : 12) ;
- le quatrième est décalé de A : 1 caractère ;
- le cinquième utilise à nouveau la première lettre de la clé et applique un décalage de A : 1 ;
- ainsi de suite...

Exemple : ici, le texte en clair « INFORMATIQUE » est chiffré à l'aide de la clé « NSI ».

Lettre en clair	I	N	F	O	R	M	A	T	I	Q	U	E
Clé	N	S	I	N	S	I	N	S	I	N	S	I
Décalage	14	19	9	14	19	9	14	19	9	14	19	9
Lettre chiffrée	W	G	O	C	K	V	O	M	R	E	N	N

D'où :

$$\text{INFORMATIQUE} \xrightarrow{NSI} \text{WGOCKVOMRENN}$$

Cette technique rend plus difficile les attaques par force brute ou par analyse statistique.

2.1.4 - Chiffrement XOR

Le chiffrement XOR repose sur l'opérateur *ou exclusif* (XOR), noté $\oplus$, appliqué bit à bit entre le message clair et la clé.

$$M \oplus K = C, \text{ et donc } C \oplus K = M$$

On rappelle sa table de vérité :

A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

Si la clé est de longueur plus petite que le message clair, la clé est répétée autant de fois que nécessaire.

Remarque :
En Python, l'opérateur ^ (*accent circonflexe*) permet de réaliser l'opérer un *ou exclusif* entre deux entiers.

COURS

2.1.5 - Chiffrement par substitution

Plutôt que de passer par une permutation circulaire des lettres de l'alphabet, on peut convenir de table de traduction qui à chaque lettre de l'alphabet associe une autre lettre.

Lettre en clair :	A	B	C	D	E	F	G	H	I	J	K	L	M
Lettre chiffrée :	S	E	T	R	O	G	Q	B	P	I	F	L	W

Lettre en clair :	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Lettre chiffrée :	H	X	C	Y	Z	V	U	J	A	K	N	D	M

$$\text{NSI} \xrightarrow{\text{table}} \text{HVP}$$

Alice et Bob doivent tous les deux disposer de la table de correspondance, qui constitue la clé secrète. Cette table peut être représentée sous la forme d'une chaîne de caractères :

```
SETROGQBPIFLWHXCYZVUJAKNDM
```

Un espion qui intercepterait le message secret sans connaître la clé peut tenter une attaque par analyse statistique si le message est suffisamment long et s'il connaît la langue utilisée en clair.

2.1.6 - Algorithmes modernes

Les algorithmes symétriques modernes, comme *AES* (Advanced Encryption Standard) ou *ChaCha20*, offrent une meilleure sécurité grâce à des clés longues (128 à 256 bits).

POUR ALLER PLUS LOIN

Le principe de Kerckhoffs (hors programme), connu depuis le XIX^{e} siècle, stipule que pour qu'un algorithme de chiffrement soit sûr, il ne doit pas reposer sur le secret de l'algorithme de chiffrement. Les cryptographes travaillent donc avec des algorithmes majoritairement publiés.

2.2 Chiffrement asymétrique

PROBLÉMATIQUE

Les algorithmes de chiffrement symétrique reposent toujours sur un principe fondamental : une clé secrète doit être partagée entre les interlocuteurs avant tout échange de données chiffrées. Si Alice et Bob ne disposent pas d'un canal de communication déjà sécurisé, un attaquant interceptant leurs communications pourrait la récupérer et déchiffrer leurs messages. Pour contourner cette limitation, une approche différente doit être adoptée.

2.2.1 - Principe

Le chiffrement asymétrique repose sur une *paire de clés* :

- une *clé publique*, qui peut être librement diffusée ;
- une *clé privée*, qui doit être gardée secrète.

Ces deux clés sont mathématiquement liées : un message chiffré avec l'une ne peut être déchiffré qu'avec l'autre. Ainsi, si Alice utilise la clé publique de Bob pour chiffrer un message, seul Bob, qui détient la clé privée correspondante, pourra le déchiffrer. Ce système permet d'échanger des informations de manière sécurisée sans nécessiter de clé secrète partagée au préalable.

Ce type de chiffrement repose sur des « *fonctions mathématiques à sens unique* ». Il s'agit de fonctions faciles à calculer dans un sens, mais pratiquement impossibles à inverser sans une information secrète (la clé privée). Connaître une clé publique ne permet donc pas de retrouver la clé privée associée.

Toutefois, le chiffrement asymétrique est bien plus coûteux en calcul que le chiffrement symétrique. Il est donc rarement utilisé pour chiffrer de grandes quantités de données, mais principalement pour établir un canal sécurisé afin d'échanger une clé de chiffrement symétrique. Une fois cette clé secrète transmise de manière sécurisée, la communication peut se poursuivre avec un algorithme symétrique (AES par exemple), plus rapide et mieux adapté aux flux temps réel (voix, vidéo) ou aux échanges volumineux.

2.2.2 - Chiffrement RSA

En 1977, Rivest, Shamir et Adleman ont proposé un système de chiffrement asymétrique connu sous le nom de RSA, fondé sur la difficulté de factoriser de grands nombres premiers.

Chaque interlocuteur génère de son côté sa propre paire de clés (K_p, K_s) :

- *une clé publique* K_p : diffusée librement, elle est utilisée par quiconque souhaite envoyer un message chiffré à son propriétaire. Un message clair m devient un message chiffré $c = K_p(m)$.
- *une clé privée* K_s : conservée secrète par le destinataire, elle lui permet de retrouver le message initial en appliquant l'opération inverse :

$$K_s(c) = K_s(K_p(m)) = m$$

L'efficacité de ce chiffrement repose sur une fonction à sens unique : la multiplication de deux très grands nombres premiers est facile à réaliser, mais la factorisation du produit obtenu est extrêmement difficile. Ainsi, la connaissance de la clé publique K_p ne permet pas d'en déduire la clé privée K_s.

COURS

Exemple : Alice génère sa paire de clés (K_p^A, K_s^A) et Bob génère de son côté (K_p^B, K_s^B). Alice souhaite recevoir de Bob un message qu'elle seule pourra déchiffrer. Ève peut intercepter toutes leurs communications.

- Alice diffuse sa clé publique K_p^A.
- Bob utilise la clé publique K_p^A pour chiffrer son message clair m : il génère un message chiffré $c = K_p^A(m)$.
- Bob envoie c à Alice.
- Alice utilise sa clé privée pour déchiffrer le message de Bob : $m = K_s^A(K_p^A(c))$.
- Pendant ce temps, Ève a intercepté K_p^A mais aussi $c = K_p^A(m)$. Heureusement pour Alice et Bob, elle ne peut déduire ni m à partir de $K_p^A(m)$ ni K_s^A à partir de K_p^A.

Remarque :
Bob peut aussi utiliser sa clé privée pour signer un message (chiffrement avec sa clé privée). Ainsi, toute personne possédant la clé publique pourra vérifier que le message provient bien de Bob, assurant son authenticité.

Briser RSA par force brute reviendrait à factoriser un très grand nombre, c'est-à-dire retrouver les deux nombres premiers utilisés pour générer la clé publique. Actuellement, les clés doivent mesurer au moins *2 048 bits* pour garantir une sécurité suffisante, bien que certaines applications utilisent déjà des clés de *4 096 bits* pour anticiper les progrès du calcul informatique.

2.3 Sécurisation des échanges sur le web

Le protocole HTTPS est un protocole HTTP sécurisé où HTTP est complété par une surcouche qui sert aux échanges entre navigateurs et serveurs web. La problématique de HTTPS est de fournir une navigation sécurisée sur des sites authentifiés par certificat, sans la ralentir (par des calculs cryptographiques trop lourds ou par un échange de données significativement plus volumineuses du fait du chiffrement).

Pour ce faire, on utilise le protocole TLS (Transport Layer Security) qui vient compléter HTTP pour former HTTPS, en combinant chiffrement symétrique et asymétrique.

La connexion à un site web sécurisé par HTTPS commence par une requête classique (HTTP) vers le serveur. Le serveur répond et démarre la phase initiale, propre à TLS (sur un port dédié, le port 443, différent du port 80 propre à HTTP), appelée « *poignée de main TLS*. En voici une décomposition (voir figure 9.6) :

- le serveur commence par envoyer au client son certificat, sa clé publique K_p^s, la signature de son certificat par une autorité de certification ;
- le client vérifie la signature du certificat (en utilisant la clé publique de l'autorité de certification) ;

- le client génère aléatoirement une clé de chiffrement symétrique K (*clé de session*) et utilise la clé publique du serveur pour la chiffrer : $K_p^s(K)$;
- le serveur reçoit la clé chiffrée et la déchiffre à l'aide de sa clé privée car $K = K_s^s(K_p^s(K))$. Il dispose alors de la clé de chiffrement symétrique K. Les données échangées par le protocole HTTP peuvent dès lors être chiffrées pour toute la durée de la session, par exemple avec l'algorithme AES.

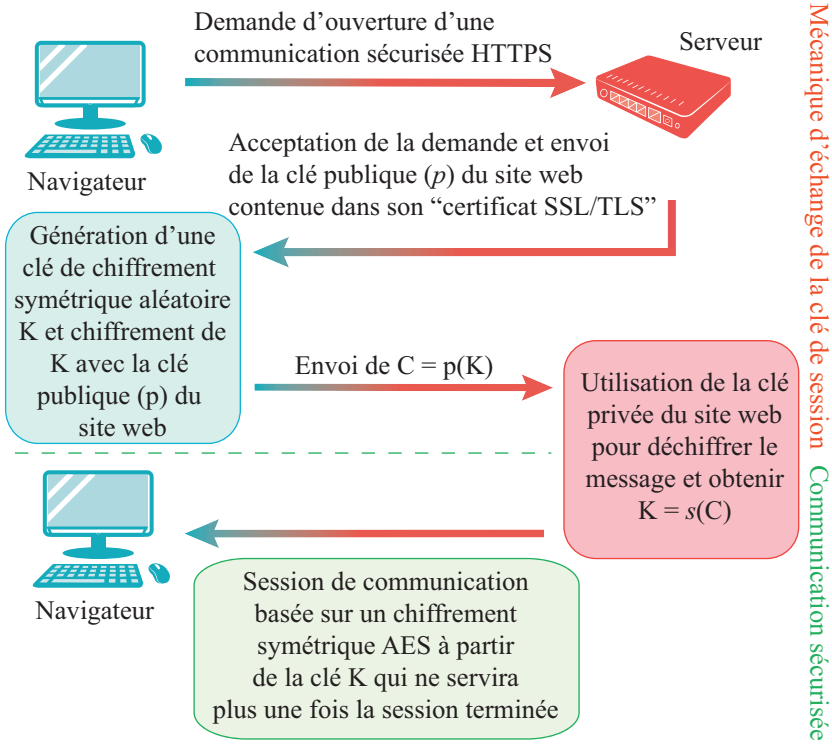


Figure 9.6 – Utilisation des chiffrements symétrique et asymétrique sur le web

À RETENIR

Chiffrement symétrique (exemple : AES) : il utilise une clé partagée, secrète, pour chiffrer et déchiffrer. La taille de la clé doit être suffisamment grande pour parer les attaques par force brute. La clé doit être échangée de façon sûre et nécessite une méthode sécurisée (chiffrement asymétrique).

Chiffrement asymétrique (exemple : RSA) : chacun génère sa paire clé privée-clé publique. Le lien entre les deux clés repose sur l'existence de fonctions mathématiques en pratique impossible à inverser.

Sécurisation d'une communication (exemple : HTTPS) : elle s'appuie sur le chiffrement asymétrique pour la signature et l'authentification de l'émetteur et permet l'échange sécurisé d'une clé de chiffrement symétrique.

INTERROS

Protocoles réseau

1 Réseau du lycée

★★ 15 min Corrigé p. 313

1 Les machines de la salle informatique d'un lycée sont reliées au réseau pédagogique dont l'adresse est `172.20.0.0`. Le masque de sous-réseau est `255.255.0.0`.

(a) Quelle est l'adresse de diffusion sur ce réseau ?

(b) Un des ordinateurs de la salle a pour adresse `172.20.204.9`. Donner son adresse en notation CIDR.

(c) Déterminer le nombre total d'hôtes adressables sur ce réseau.

2 Le réseau administratif dispose de son propre réseau, d'adresse `172.10.160.0/20`.

(a) Déterminer le masque de sous-réseau du réseau administratif.

(b) Quelle est l'adresse de diffusion ?

(c) Donner le nombre maximum de machines pouvant être connectées à ce réseau.

(d) La machine d'adresse `172.10.176.1` fait-elle partie de ce réseau ? Justifier.

2 ARPANET

★★★ 30 min Corrigé p. 313

Lycée Richelieu, Rueil-Malmaison

Cet exercice porte sur les protocoles réseau.

ARPANET, partie ouest représentée sur la figure 9.7, est le premier réseau à transfert de paquets de données conçu aux États-Unis. En 1971, ce réseau contenait 23 nœuds dont 8 dans la partie ouest :

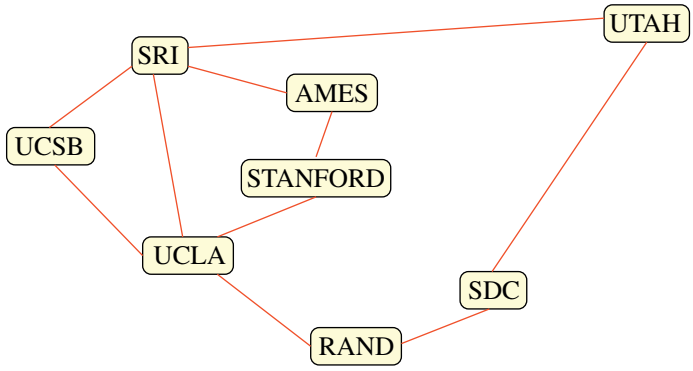


Figure 9.7 – ARPANET en 1971, partie ouest.

- SRI : Stanford Research Institute
- AMES : Ames Research (NASA)
- UCSB : Université de Santa Barbara
- UCLA : Université de Los Angeles
- STANFORD : Université, Silicon Valley
- RAND : Research And Development
- SDC : System Development Corporation (Santa Monica)
- UTAH : Université de Salt Lake City

On schématise ce réseau par un ensemble de routeurs (appelés A, B, C, D, E, F, G et H) chacun associé directement à un réseau du même nom :

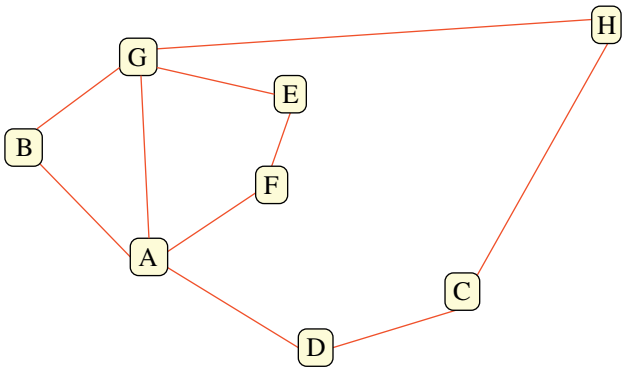


Figure 9.8 – Schématisation du réseau.

On s'intéresse au protocole RIP, qui minimise le nombre de sauts entre routeurs. Si on l'applique au nœud A de la figure 9.8, on obtient la table de coûts suivante :

Nœud A	
Destination	Coût
B	1
C	2
D	1
E	2
F	1
G	1
H	2

1 Écrire la table des coûts des nœuds B et F.

2 En appliquant le protocole RIP, donner tous les chemins possibles d'un paquet de données partant du nœud F à destination du nœud H.

INTERROS

3 L'armée ajoute le nœud Z correspondant sur le réseau. Sa table de coûts est la suivante :

Nœud Z	
Destination	Coût
A	3
B	3
C	1
D	2
E	3
F	4
G	2
H	1

Tracer le réseau en ajoutant ce nœud Z pour qu'il respecte cette table de routage.

4 On utilise maintenant le protocole OSPF qui minimise la somme des coûts de la transmission entre deux nœuds. Le réseau avec les coûts est illustré par la figure 9.9.

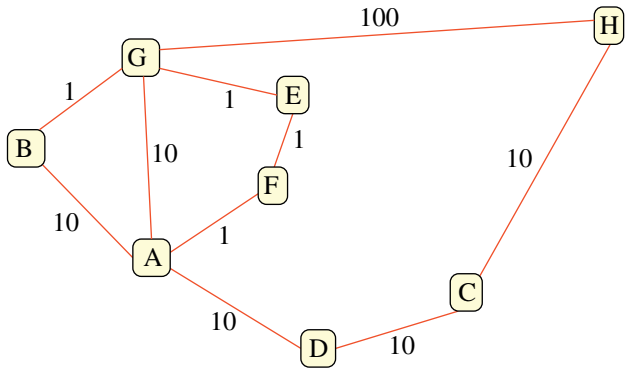


Figure 9.9 – Réseau avec coûts.

Avec le protocole OSPF, donner en justifiant le chemin pris par un paquet d'origine B à destination du nœud H.

3 Réseaux et graphes

★★★ 20 min Corrigé p. 314

D'après Métropole J1 Septembre 2024, exercice 2

Le graphe représenté page ci-contre schématise l'architecture du réseau d'une entreprise. Les sommets représentent les routeurs et les arêtes représentent les liaisons.

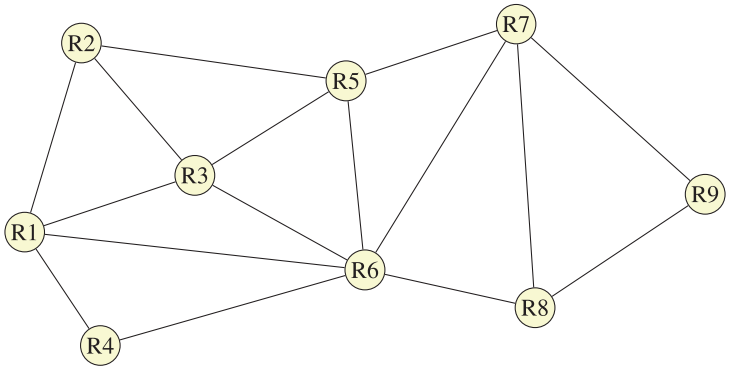


Figure 9.10 – Graphe non pondéré

1 On suppose que le protocole de routage RIP est utilisé.

(a) Recopier et compléter, en rajoutant autant de lignes que nécessaire, la table de routage simplifiée suivante du routeur R1.

Destination	Suivant	Nombre de sauts
R2	R2	1
R3		

Un paquet doit passer du routeur R1 au routeur R9.

(b) Donner l'un des chemins empruntés par le paquet ainsi que le nombre de sauts.

On appelle M la matrice d'adjacence du graphe de la figure 9.10. Les sommets sont rangés par ordre croissant des numéros des routeurs (R1, R2, ..., R9).

(c) Donner l'écriture en Python de cette matrice d'adjacence sous la forme d'une liste de listes.

Le degré d'un sommet est le nombre d'arêtes dont ce sommet est une extrémité.

(d) Recopier et compléter les lignes 3, 5 et 6 de la fonction `degre` qui prend en paramètre la matrice d'adjacence d'un graphe donné sous forme d'une liste de listes et qui renvoie la liste des degrés de tous les sommets du graphe rangés dans le même ordre que les sommets de la matrice d'adjacence.

```
1 def degre(matrice):
2   d = []
3   for ... in ...:
4     cpt = 0
5     for ... in ...:
6       cpt = cpt + ...
7     d.append(cpt)
8   return d
```

(e) Donner la liste renvoyée par `degre(M)`.

On appelle chaîne eulérienne d'un graphe non orienté un chemin qui passe une et une seule fois par toutes les arêtes du graphe. Un graphe connexe admet une chaîne eulérienne si et seulement si le graphe possède, au plus, deux sommets de degré impair.

Afin de vérifier l'état physique de la fibre optique installée sur le réseau, un robot inspecte toute la longueur de la fibre optique afin de s'assurer qu'elle ne présente pas de détérioration apparente.

(f) En utilisant le résultat de la question précédente et en admettant que le graphe est connexe, indiquer si le robot peut parcourir l'ensemble du réseau en suivant les fibres optiques et en empruntant chaque fibre optique une et une seule fois.

2 On considère maintenant le même réseau, représenté par un graphe pondéré figure 9.11 dans lequel le poids de chaque arête représente la bande passante (ou débit maximal) de chaque liaison, en mégabits par seconde (Mb/s). On utilise le protocole de routage OSPF pour lequel le coût C de chaque liaison est calculé par la formule :

$$C = \frac{10^8}{BP}$$

avec BP la bande passante de la liaison, en bits par seconde.

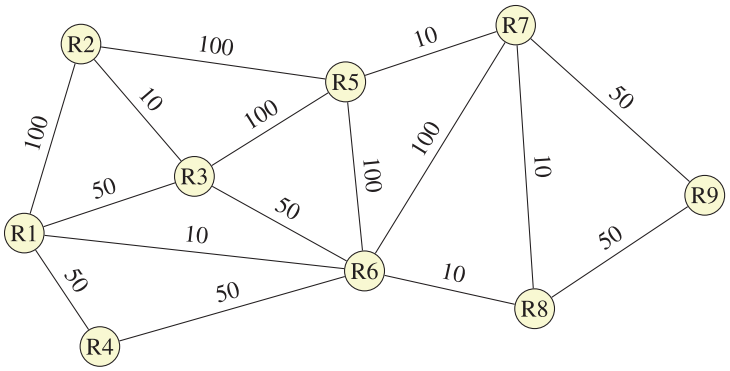


Figure 9.11 – Graphe pondéré

(a) Reproduire le graphe pondéré en indiquant le coût de chaque arête.

(b) Déterminer la route empruntée par un paquet partant du routeur R1 à destination du routeur R9. Préciser le coût de ce trajet.

Sécurisation des communications

4 ROT13

★ 10 min Corrigé p. 316

Lycée Henri Alain-Fournier, Bourges

Écrire une fonction Python `chiffrement_rot13(message)` permettant de chiffrer un message à l'aide de l'algorithme de chiffrement ROT13. Pour cela,

- on s'assurera que toutes les lettres du message sont en minuscule ;
- si le message comporte plusieurs mots, on le découpera ;
- on écrira une fonction `rot13(mot)` chiffrant un mot à l'aide de l'algorithme demandé ;
- dans la fonction `chiffrement_rot13(message)`, on fera appel à la fonction `rot13(mot)`.

Chiffrer alors le message : « cheval indomptable ».

5 Chiffrement à l'aide de XOR

★★ 30 min Corrigé p. 317

Lycée Sainte-Clotilde, Strasbourg

1 On donne le programme suivant :

```
def getBytes(msg):
  return [ord(c) for c in msg]

phrase  = "My name is Bond, James Bond..."
liste = getBytes(phrase)
print(phrase)
print(liste)
```

(a) Qu'affiche-t-il ?

(b) Son résultat peut-il être considéré comme une version chiffrée du texte initial ? Expliquez.

2 Écrire la fonction réciproque `getText(liste)` qui, à partir du résultat de `getBytes(msg)`, retourne le texte initial `msg`.

3 On veut maintenant utiliser une clé secrète pour encoder les valeurs de la liste résultat de getBytes() afin qu'il ne soit pas possible à un intervenant de retrouver le texte initial par `getText()` sans connaître la clé.

Pour ce faire, on décide d'utiliser l'opération « OU exclusif » (XOR) entre chaque octet du texte et celui de la clé. XOR est une opération binaire (notée ^ en Python) qui s'effectue sur chacun des bits des deux opérandes (voir page suivante).

XOR	0	1
0	0	1
1	1	0

Tableau 9.1 – Table du « OU exclusif »

Par exemple,

56 ^ 83 → 0011 1000 ^ 0101 0011 = 0110 1011 → 107.

Supposons, pour le moment, que la clé est plus longue que le texte :

Clé = « Ceci est une clé de 41 caractères de long »

(a) Écrire une fonction qui reçoit un texte et une clé en arguments et retourne les octets du texte combinés aux octets respectifs de la clé via l'opération XOR. Le résultat sera fourni sous forme d'une liste d'octets.

(b) Appliquer cette fonction au texte « message urgent » avec la clé « cryptographie005 ».

Afficher la liste des octets ainsi cryptés obtenue et comparer avec l'utilisation de la fonction `getBytes()` de la question 1.

(c) Constater qu'une fois l'opération XOR appliquée, il n'est pas possible de récupérer la phrase d'origine avec `getText()`.

(d) Pour décrypter le texte, il faut appliquer l'opération inverse du XOR à la liste des octets chiffrés. Donc, il faut disposer de la clé secrète.

L'opération XOR est involutive, c'est-à-dire que :

si a XOR b = c alors c XOR a = b et c XOR b = a.

Pour l'inverse, il suffit donc de l'appliquer une nouvelle fois, sur les octets chiffrés, avec la même clé.

Constater qu'une fois l'opération XOR appliquée à nouveau, le texte peut être déchiffré.

4 On veut maintenant pouvoir utiliser une clé de longueur quelconque (typiquement plus courte que le message à transmettre). Il n'y aura alors pas assez d'octets dans la clé pour calculer le XOR en une seule fois sur tout le message. On va donc répéter la clé, autant de fois que nécessaire, pour chiffrer le message en entier.

Par exemple,

- `Clé: "Key"`
- `Message: "Ceci est le message"`
  `"KeyKeyKeyKeyKeyKeyK"`
-
```
Message chiffré = [ ord("C") ^ ord("K"),
                    ord("e") ^ ord("e"),
                    ord("c") ^ ord("y"),
                    ord("i") ^ ord("K"),
                    ... ]
```

(a) Écrire une fonction `computeXorRepeatingKey(text,key)` qui effectue ce calcul et l'appliquer sur une phrase.

(b) Utiliser la même fonction, avec la même clé pour déchiffrer le résultat précédent et vérifier que la reconstruction de la phase conduit bien à celle de départ.

6 Échange de clés de Diffie-Hellman

★★★ 20 min Corrigé p. 319

Alice et Bob souhaitent s'échanger une clé K destinée à chiffrer leurs échanges par chiffrement symétrique. Malheureusement, ils ne disposent pas de canal sécurisé pour partager leur clé.

Voici le procédé proposé par Diffie et Hellman (*cf.* figure 9.12) :

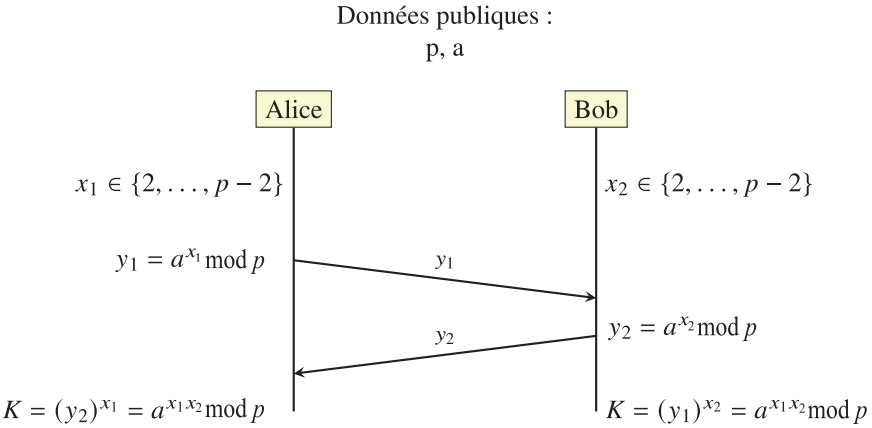


Figure 9.12 – Échange de clés selon Diffie-Hellman

1. Alice et Bob choisissent ensemble un grand nombre premier p, ainsi qu'un entier positif a strictement inférieur à p. Ce choix peut être fait via le réseau et ces deux valeurs a et p peuvent être divulguées publiquement.
2. Alice choisit secrètement un nombre x_1. Elle calcule ensuite $y_1 = a^{x_1} \bmod p$. Ce type de calcul est très facile à réaliser par la machine, même avec des grands nombres.
3. Bob, de son côté, choisit secrètement un nombre x_2 et calcule $y_2 = a^{x_2} \bmod p$.
4. Alice et Bob s'échangent alors leurs valeurs de y_1 et y_2, qui peuvent là encore être interceptées.
5. Alice calcule alors $y_2^{x_1}$, ce qui donne $(a^{x_2})^{x_1} = a^{x_1 x_2} \bmod p$. On convient d'appeler K ce nombre et de s'en servir comme clé secrète pour échanger avec Bob.

INTERROS

6 En effet, Bob, de son côté, calcule $y_1^{x_2}$, ce qui donne $(a^{x_1})^{x_2} = a^{x_1 x_2} \bmod p$, ce qui lui permet de retrouver K de son côté. Finalement, ils disposent chacun de leur côté de la clé secrète sans jamais l'avoir faite transiter sur le réseau.

Remarque : pour simplifier les calculs suivants, on travaillera sur des petits nombres.

Alice et Bob décident, à deux, d'un nombre premier $p = 29$ puis d'un autre nombre entier positif a, qui n'est pas forcément premier mais qui est strictement inférieur à p. Ils choisissent : $a = 3$. De plus, Alice choisit un nombre secret $x_1 = 5$ et Bob choisit un nombre secret $x_2 = 6$.

1 Calculer les valeurs des nombres y_1 et y_2 qu'ils calculent à l'aide de la relation $y = a^x \bmod p$.

2 En déduire la clé $K = a^{x_1 x_2} \bmod p$.

3 Une propriété mathématique énonce qu'il est impossible de trouver x_1 (ou x_2 ou $a^{x_1 x_2}$) connaissant a, p et y_1 (y_2) (pour de grandes valeurs). Recenser les données collectées par Ève et indiquer si elle est en mesure de retrouver la clé K.

4 Quelle conséquence peut-avoir le choix d'un nombre $p = 29$ aussi petit, si la clé K calculée est utilisée ensuite comme clé de chiffrement ?

1 Réseau du lycée

Énoncé p. 304

1 **(a)** L'adresse de diffusion se retrouve en plaçant tous les bits de la partie hôte à 1. À l'aide du masque, on déduit que la partie hôte occupe les 16 bits de poids faible. D'où : `172.20.255.255`.

(b) L'analyse du masque nous indiquant que la partie réseau est codée sur les 16 bits de poids fort, on note l'adresse : `172.20.204.9/16`.

(c) Il y a au total $2^{16} - 2 = 65\,534$ machines adressables.

2 Le réseau administratif dispose de son propre réseau, d'adresse `172.10.160.0/20`.

(a) La notation CIDR utilisée indique que les 20 bits de poids fort de l'adresse codent pour la partie réseau. Le masque de sous-réseau est donc construit en mettant les 20 bits de poids fort à 1 et les 12 autres à 0, ce qui donne deux premiers octets de valeur 255, le troisième octet de valeur binaire `1111 0000` soit :

$$2^7 + 2^6 + 2^5 + 2^4 = 128 + 64 + 32 + 16 = 240$$

et le quatrième de valeur 0 : `255.255.240.0`.

(b) L'adresse de diffusion reprend la partie réseau de l'adresse et place tous les bits de la partie hôte à 1. D'où : `172.10.175.255`. Le troisième octet a pour valeur binaire : `1010 1111` (les 4 premiers bits sont issus de la partie réseau, les quatre derniers appartiennent à la partie hôte et sont à 1).

(c) Le nombre d'hôtes adressables est :

$$2^{32-20} - 2 = 2^{12} - 2 = 4\,096 - 2 = 4\,094.$$

(d) La machine d'adresse `172.10.176.1` appartient au réseau si sa partie réseau est identique à celle de l'adresse réseau `172.10.160.0`. La comparaison des deux premiers octets est immédiate. Étant donné que la partie réseau couvre les 20 bits de poids fort, il faut comparer aussi les 4 bits de poids fort du troisième octet : $160 = 1010\,0000$ pour le réseau et $176 = 1011\,0000$. Les 4 bits de poids fort sont différents, la machine n'appartient pas au réseau.

2 ARPANET

Énoncé p. 304

Lycée Richelieu, Rueil-Malmaison

1 On a les tables suivantes :

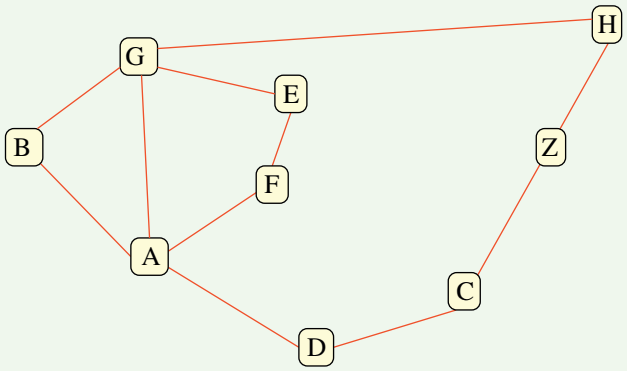
Nœud B	
Destination	Coût
A	1
C	3
D	2
E	2
F	2
G	1
H	2

Nœud F	
Destination	Coût
A	1
B	2
C	3
D	2
E	1
G	2
H	3

2 En partant du nœud F, nous avons les possibilités suivantes :

- F – E – G – H
- F – A – G – H

3 En insérant le nœud Z, on a le réseau suivant :



4 Le chemin qui a le plus faible coût :

$$B - G - E - F - A - D - C - H$$

avec un coût de 34.

À titre comparatif, par exemple, le chemin B–A–D–C–H a un coût de $10 + 10 + 10 + 10 = 40$, ce qui est plus grand que 34.

3 Réseaux et graphes

Énoncé p. 306

D'après Métropole J1 Septembre 2024, exercice 2

1 **(a)** On dresse autant de lignes que de routeurs accessible via le réseau. R2, R3, R6 et R4 sont directement reliés à R1, ils ont donc une métrique de 1. R5 est accessible avec une métrique de 2 (deux sauts), indifféremment via R2, R3 ou R6.

Destination	Suivant	Nombre de sauts
R2	R2	1
R3	R3	1
R4	R4	1
R5	R2	2
R6	R6	1
R7	R6	2
R8	R6	2
R9	R6	3

(b) Le protocole RIP minimise le nombre de sauts. R9 sera donc atteint par le chemin R1 – R6 – R7 (ou R8) – R9.

(c) Matrice d'adjacence sous forme de liste de liste :

```
M =[[0, 1, 1, 1, 0, 1, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0, 0, 0],
    [1, 1, 0, 0, 1, 1, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 1, 1, 1, 0, 0],
    [1, 0, 1, 1, 1, 0, 1, 1, 0],
    [0, 0, 0, 0, 1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 1, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 1, 1, 0]]
```

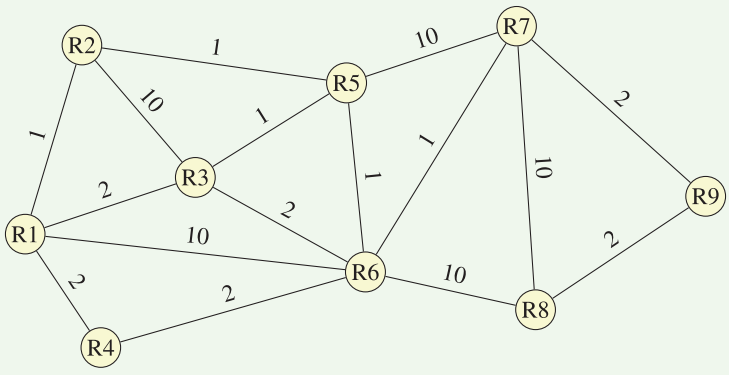
(d)

```
def degre(matrice):
  d = []
  for sommet in matrice:
    cpt = 0
    for arete in sommet:
      cpt = cpt + arete
    d.append(cpt)
  return d
```

(e) `degre(M)` retourne `[4, 3, 4, 2, 5, 6, 4, 3, 2]`.

(f) Le robot doit inspecter toutes les liaisons (parcourir toutes les arêtes du graphe) une et une seule fois. Un tel chemin existe s'il existe une chaîne eulérienne dans ce graphe. Une telle chaîne existe si et seulement si le graphe possède au plus deux sommets de degré impair. Il y a trois sommets (R2, R5, R8) de degrés impair. On en déduit donc qu'une telle chaîne n'existe pas. Le robot ne pourra donc pas parcourir le réseau en respectant les conditions imposées.

2 **(a)** Le graphe pondéré par les coûts des liaisons calculées est le suivant :



(b) Le plus court chemin est : R1 – R2 – R5 – R6 – R7 – R9. Son coût total est : $1 + 1 + 1 + 1 + 2 = 6$.

CORRIGÉS

4 ROT13

Énoncé p. 309

Lycée Henri Alain-Fournier, Bourges

Une proposition de programme est donnée page suivante.

Dans la fonction `rot13(mot)`,

- on commence par initialiser une chaîne de caractères `r` vide, qui contiendra le résultat attendu ;
- on parcourt chaque lettre du mot : on détermine son code ASCII avec `ord(lettre)` et on lui ajoute 13. Ensuite, on soustrait le code ASCII du caractère 'a' afin d'associer 'a' à 0, 'b' à 1, etc. Le résultat de ces opérations est pris modulo 26 car l'alphabet est considéré comme un « cycle » de longueur 26. On obtient ainsi le rang de la lettre chiffrée. Il suffit ensuite d'ajouter `ord('a')` au résultat pour obtenir le code ASCII de la lettre chiffrée. Ensuite, on utilise la commande `chr(code ASCII)` pour obtenir le caractère souhaité, que l'on ajoute à la chaîne `r`.

Dans la fonction `chiffrement_rot13(message)`, on transforme le message de sorte qu'il ne contienne que des minuscules, puis on parcourt la liste des mots composant le message (obtenue à l'aide de la méthode `split`) afin d'appliquer à chacun d'eux la fonction `rot13(mot)`.

Le message « cheval indomptable » est alors chiffré en :

« puriny vaqbzcgnoyr »

```
def rot13(mot):
    r = ''
    for lettre in mot:
        r += chr((ord(lettre) + 13 - ord('a'))%26
        + ord('a'))

    return r

def chiffrement_rot13(message):
    r = ''
    message = message.lower()
    for mot in message.split(' '):
        r += rot13(mot) + ' '

    return r

print(chiffrement_rot13('cheval indomptable'))
```

5 Chiffrement à l'aide de XOR

Énoncé p. 309

Lycée Sainte-Clotilde, Strasbourg

1 **(a)** La fonction `getBytes(msg)` retourne la liste des valeurs décimales de chaque octet qui compose le texte reçu `msg`. Par exemple, si le texte reçu est « My » alors la fonction renvoie la liste `[77,121]` car 77 est le code ASCII de « M » et 121 celui de « y ».

Ainsi, le programme proposé affiche la liste de tous les codes ASCII des caractères qui composent le message informé.

(b) Le résultat du programme est effectivement une manière d'encoder (écrire différemment) le texte initial via un encodage réversible. Techniquement, on peut donc dire qu'il s'agit d'un chiffrement dont la sécurité repose sur la connaissance de l'algorithme utilisé (remplacement de chaque lettre par la valeur décimale de l'octet correspondant).

L'algorithme utilisé étant extrêmement simple, le chiffrement serait immédiatement « cassé ».

2 Une fonction possible est la suivante :

```
def getText(liste):
  return "" . join([chr(b) for b in liste])
```

Il suffit ici de parcourir la liste informée en argument de la fonction et de trouver, à l'aide de la fonction `chr()`, le caractère qui correspond au code ASCII de l'élément de la liste.

3 **(a)** Voici une proposition de fonction :

```
def computeXor(text,key):
  textBytes   = getBytes(text)
  keyBytes  = getBytes(key)
  return [textBytes[i] ^ keyBytes[i] for i in
    range(len(textBytes))]
```

(b) En utilisant la fonction `computeXor`, la liste suivante est retournée :

```
[14, 23, 10, 3, 21, 8, 2, 82, 20, 2, 15, 12, 11, 68]
```

alors qu'avec la fonction `getBytes`, on obtient la liste suivante :

```
[109, 101, 115, 115, 97, 103, 101, 32, 117, 114, 103,
101, 110, 116]
```

Bien entendu, nous n'obtenons pas du tout la même liste ; la première liste est bien plus « mystérieuse » que la seconde.

(c) Avec `print( getText(xorBytes) )`, où `xorBytes` est la liste retournée par la fonction `computeXor()`, nous avons une belle surprise...

Selon l'interpréteur que l'on utilise, l'affichage diffère mais une chose est sûre : ce n'est pas le message d'origine que nous voyons !

(d) Avec le programme suivant :

```
phrase    = "message urgent"
key     = "cryptographie005"
xorBytes  = computeXor(phrase,key)
print(xorBytes)
crypText = getText(xorBytes)
xorBytes2 = computeXor(crypText,key)
print(xorBytes2)
decrypt   = getText(xorBytes2)
print(decrypt)
```

on s'aperçoit en effet que le message initial apparaît en clair.

4 **(a)** Voici une fonction possible :

```
def computeXorRepeatingKey(textBytes,key) :
  keyBytes  = getBytes(key)
  return [textBytes[i] ^ keyBytes[i % len(
    keyBytes)] for i in range(len(textBytes))]

text      = "My name is Bond, James Bond... I'm
    trying to find what is written into this
    message"
key       = "privateKey007"
textBytes     = getBytes(text)
encodedBytes  = computeXorRepeatingKey(textBytes
    ,key)

print("Phrase:\r\n{}".format(text))
print("Octets non encodés:\r\n{}".format(
    textBytes))
print("Octets encodés:\r\n{}".format(
    encodedBytes))
```

(b) On peut utiliser le programme suivant :

```
decodedBytes  = computeXorRepeatingKey(
    encodedBytes,key)
decodedText   = getText(decodedBytes)

print("Phrase:\r\n{}".format(text))
print("Octets encodés:\r\n{}".format(
    encodedBytes))
print("Octets décodés:\r\n{}".format(
    decodedBytes))
print("Restitution de la phrase:\r\n{}".format(
    decodedText))
```

Nous avons ici décidé d'afficher les listes des octets encodés et décodés, mais nous pouvons afficher directement la seule phrase déchiffrée, qui correspond bien à la phrase initiale.

Téléchargez le programme complet.

6 Échange de clés de Diffie-Hellman

Énoncé p. 311

1 On calcule $y_1 = a^{x_1} \bmod p = 3^5 \bmod 29 = 243 mod 29 = 11$.
Sur la console Python : `(3**5)%29`)
$y_2 = a^{x_2} \bmod p = 3^6 \bmod 29 = 243 \bmod 29 = 4$.

2 On en déduit la clé : $K = a^{x_1 x_2} \bmod p = 3^{5\times 6} \bmod 29 = 9$.

3 Ève connaît : a, p, y_1 et y_2. Ce qu'elle connaît ne lui permet pas de retrouver K car x_1 et x_2 ne peuvent être déduits de a, p et y_1 (y_2). Cela reviendrait à résoudre le problème du logarithme discret, réputé difficile à résoudre pour des grands nombres. Toutefois, avec de telles petites valeurs, une attaque par force brute est envisageable. En pratique, pour garantir la sécurité, on utilise des valeurs très grandes.

4 Avec un nombre premier p petit, l'attaque par force brute demeure possible car quel que soit le résultat de $a^{x_1 x_2}$, le modulo 29 ne donne que 29 possibilités différentes.

Chapitre 10

Baccalauréat blanc

Voir nos conseils de méthode « Spécial Bac » dans la partie « Méthodes de travail » en début d'ouvrage.

La durée de l'épreuve est de 3 h 30 min. Le sujet comporte trois exercices.

1 Environnement Linux

★★ 60 min Corrigé p. 333

D'après bac 2024 Étranger J2

Cet exercice, noté sur 6 points, porte sur la programmation objet, la récursivité, les arbres et les systèmes d'exploitation.

Dans cet exercice, on travaille dans un environnement Linux. On considère l'arborescence de fichiers de la figure 1.

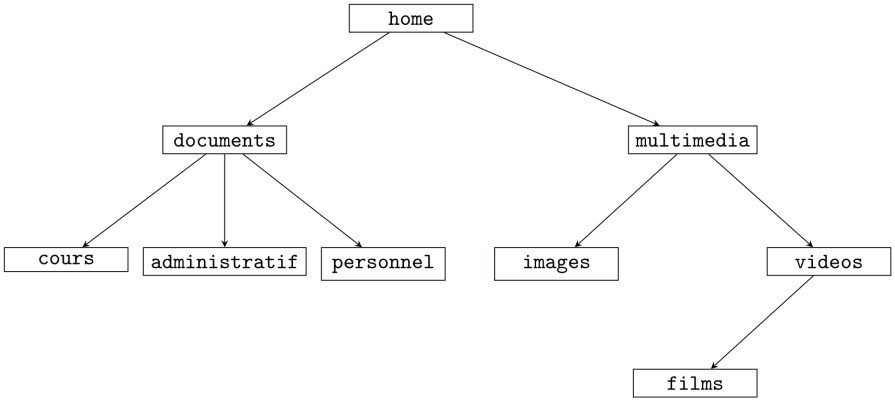


Figure 10.1 – Arborescence de fichiers

Partie A : arborescence de fichiers

1. Le répertoire courant est `home`. Donner une commande permettant de connaître le contenu du dossier `documents`.

   On suppose que l'on se trouve dans le dossier `cours` et que l'on exécute la commande `mv ../../multimedia /home/documents`
2. Indiquer la modification que cela apporte dans l'arborescence de la figure 1.

SUJET

On considère le code suivant :

```
class Arbre:
  def __init__(self, nom, g, d):
    self.nom = nom
    self.gauche = g
    self.droite = d

  def est_vide(self):
    return self.gauche is None and self.droite is
    None

  def parcours(self):
    print(self.nom)
    if self.gauche != None:
      self.gauche.parcours()
    if self.droite != None:
      self.droite.parcours()
```

3 Donner une raison qui justifie que le code précédent ne permet pas de modéliser l'arborescence de fichiers de la figure 1.

4 Donner le nom du parcours réalisé par le code précédent.

5 Donner la liste des dossiers dans l'ordre d'un parcours en largeur de l'arborescence. On ne demande pas d'écrire ce parcours en Python.

Partie B : implémentation de la classe Dossier

1 Pour pouvoir modéliser l'arborescence de fichiers de la figure 1, on propose l'implémentation suivante. L'attribut `fils` est une variable de type `list` contenant tous les dossiers fils. Cette liste est vide dans le cas où le dossier est vide.

```
class Dossier:
  def __init__(self, nom, liste):
  self.nom = nom
  self.fils = liste # liste d'objets de la classe
    Dossier
```

2 Écrire le code Python d'une méthode `est_vide` qui renvoie `True` lorsque le dossier est vide et `False` sinon.

3 Écrire le code Python permettant d'instancier la variable `var_multimedia` de la classe `Dossier` représentant le dossier `multimedia` de la figure 1. Attention : cela nécessite d'instancier tous les nœuds du sous-arbre de racine `multimedia`.

4 Recopier et compléter sur votre copie le code Python de la méthode `parcours` suivante qui affiche les noms de tous les descendants d'un dossier en utilisant l'ordre préfixe.

```
def parcours(self):
  print(...)
  for f in ...:
    ...
```

5 Justifier que cette méthode `parcours` termine toujours sur une arborescence de fichiers.

6 Proposer une modification de la méthode `parcours` pour que celle-ci effectue plutôt un parcours suffixe (ou postfixe).

7 Expliquer la différence de comportement entre un appel à la méthode `parcours` de la classe `Dossier` et une exécution de la commande UNIX `ls`.

On considère la variable `var_videos` de type `Dossier` représentant le dossier `videos` de la figure 1. On souhaite que le code Python `var_videos.mkdir("documentaires")` crée un dossier `documentaires` vide dans le dossier `var_videos`.

8 Écrire le code Python de la méthode `mkdir`.

9 Écrire en Python une méthode `contient(self, nom_dossier)` qui renvoie `True` si l'arborescence de racine `self` contient au moins un dossier de nom `nom_dossier` et `False` sinon.

10 Avec l'implémentation de la classe `Dossier` de cette partie, expliquer comment il serait possible de déterminer le dossier parent d'un dossier donné dans une arborescence donnée. On attend ici l'idée principale de l'algorithme décrite en français. On ne demande pas d'implémenter cet algorithme en Python.

11 Proposer une modification dans la méthode `__init__` de la classe `Dossier` qui permettrait de répondre à la question précédente beaucoup plus efficacement et expliquer votre choix.

2 Groupe de personnes

★★ 60 min Corrigé p. 335

D'après bac 2024 Amérique Nord J1

Cet exercice, noté sur 6 points, porte sur les graphes.

Dans cet exercice, on modélise un groupe de personnes à l'aide d'un graphe.

Le groupe est constitué de huit personnes (Anas, Emma, Gabriel, Jade, Lou, Milo, Nina et Yanis) qui possèdent entre elles les relations suivantes :

- Gabriel est ami avec Jade, Yanis, Nina et Milo ;
- Jade est amie avec Gabriel, Yanis, Emma et Lou ;

SUJET

- Yanis est ami avec Gabriel, Jade, Emma, Nina, Milo et Anas ;
- Emma est amie avec Jade, Yanis et Nina ;
- Nina est amie avec Gabriel, Yanis et Emma ;
- Milo est ami avec Gabriel, Yanis et Anas ;
- Anas est ami avec Yanis et Milo ;
- Lou est amie avec Jade.

Partie A : matrice d'adjacence

On choisit de représenter cette situation par un graphe dont les sommets sont les personnes et les arêtes représentent les liens d'amitié.

1 Dessiner sur votre copie ce graphe en représentant chaque personne par la première lettre de son prénom entourée d'un cercle et où un lien d'amitié est représenté par un trait entre deux personnes.

Une matrice d'adjacence est un tableau à deux entrées dans lequel on trouve en lignes et en colonnes les sommets du graphe.

Un lien d'amitié sera représenté par la valeur `1` à l'intersection de la ligne et de la colonne qui représentent les deux amis alors que l'absence de lien d'amitié sera représentée par un `0`.

2 Recopier et compléter l'implémentation de la déclaration de la matrice d'adjacence du graphe.

```
# sommets : G, J, Y, E, N, M, A, L
matrice_adj = [[0, 1, 1, 0, 1, 1, 0, 0], # G
               [......................], # J
               [......................], # Y
               [......................], # E
               [......................], # N
               [......................], # M
               [......................], # A
               [......................]] # L
```

On dispose de la liste suivante qui identifie les sommets du graphe :

```
sommets = ['G', 'J', 'Y', 'E', 'N', 'M', 'A', 'L']
```

On dispose d'une fonction `indice(l, s)` qui prend en paramètres une liste de sommets `l` et un nom de sommet `s` et qui renvoie l'indice du sommet `s` dans la liste `l` s'il est présent et `None` sinon.

3 Indiquer quel seront les retours de l'exécution des instructions suivantes :

```
>>> indice(sommets, 'G')
>>> indice(sommets, 'Z')
```

4 Recopier et compléter le code de la fonction `nb_amis(L, m, s)` qui prend en paramètres une liste de noms de sommets `L`, une matrice d'adjacence `m` d'un graphe et un nom de sommet `s` et qui renvoie le nombre d'amis du sommet `s` s'il est présent dans `L` et `None` sinon.

```
def nb_amis(L, m, s):
  pos_s = ...
  if pos_s == None:
    return ...
  amis = 0
  for i in range(len(m)):
    amis += ...
  return ...
```

5 Indiquer quel est le retour de l'exécution de la commande suivante :

```
>>> nb_amis(sommets, matrice_adj, 'G')}
```

Partie B : dictionnaire de listes d'adjacence

1 Dans un dictionnaire Python `{c : v}`, indiquer ce que représentent `c` et `v`.

On appelle `graphe` le dictionnaire de listes d'adjacence associé au graphe des amis. On rappelle que Gabriel est ami avec Jade, Yanis, Nina et Milo.

```
graphe = {'G' : ['J', 'Y', 'N', 'M'],
   'J' : ...
   ...
    }
```

2 Recopier et compléter le dictionnaire de listes d'adjacence `graphe` sur votre copie pour qu'il modélise complètement le groupe d'amis.

3 Écrire le code de la fonction `nb_amis(d, s)` qui prend en paramètres un dictionnaire d'adjacence `d` et un nom de sommet `s` et qui renvoie le nombre d'amis du nom de sommet `s`. On suppose que `s` est bien dans `d`.

Par exemple :

```
>>> nb_amis(graphe, 'L')
1
```

Milo s'est fâché avec Gabriel et Yanis tandis qu'Anas s'est fâché avec Yanis. Le dictionnaire d'adjacence du graphe qui modélise cette nouvelle situation est donné ci-dessous :

```
graphe = {'G' : ['J', 'N'],
    'J' : ['G', 'Y', 'E', 'L'],
    'Y' : ['J', 'E', 'N'],
```

SUJET

```
    'E' : ['J', 'Y', 'N'],
    'N' : ['G', 'Y', 'E'],
    'M' : ['A'],
    'A' : ['M'],
    'L' : ['J']
   }
```

Pour établir la liste du cercle d'amis d'un sommet, on utilise un parcours en profondeur du graphe à partir de ce sommet. On appelle cercle d'amis de `Nom` toute personne atteignable dans le graphe à partir de `Nom`.

4 Donner la liste du cercle d'amis de Lou.

Un algorithme possible de parcours en profondeur de graphe est donné ci-dessous :

```
visités = liste vide des sommets déjà visités
fonction parcours_en_profondeur(d, s)
   ajouter s à la liste visités
   pour tous les sommets voisins v de s :
      si v n'est pas dans la liste visités :
         parcours_en_profondeur(d, v)
   retourner la liste visités
```

5 Recopier et compléter le code de la fonction `parcours_en_profondeur(d, s)` qui prend en paramètres un dictionnaire d'adjacence `d` et un sommet `s` et qui renvoie la liste des sommets issue du parcours en profondeur du graphe modélisé par `d` à partir du sommet `s`.

```
visites = []

def parcours_en_profondeur(d, s):
  ...
  for v in d[s]:
    ...
      parcours_en_profondeur(d, v)
  ...
```

3 Gestion d'un hôpital

★★ 80 min Corrigé p. 337

D'après bac 2024 Polynésie J2

Cet exercice, noté sur 8 points, porte sur les protocoles réseau, les bases de données relationnelles et les requêtes SQL, l'algorithmique et la programmation en Python.

Cet exercice est composé de trois parties indépendantes.

Partie A : le réseau informatique d'un hôpital

Dans un hôpital, le service informatique a créé le réseau ci-dessous :

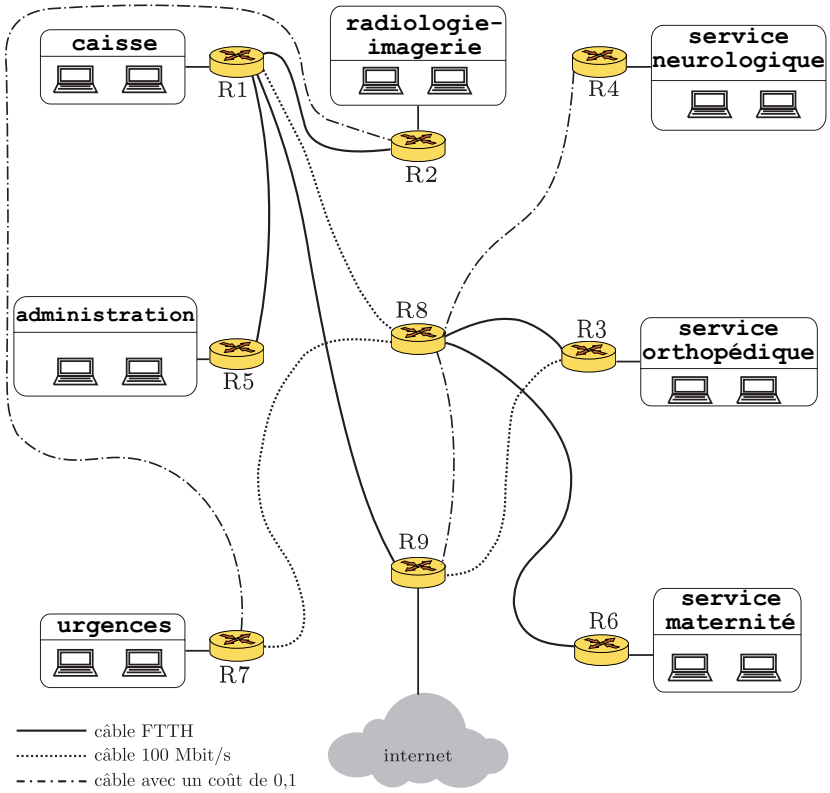


Figure 10.2 – Réseau informatique de l'hôpital

Dans un premier temps, on s'intéresse au protocole RIP, qui minimise le nombre de sauts entre routeurs.

1 Donner les chemins possibles d'un paquet de données partant du service de neurologie à destination du service d'imagerie en utilisant le protocole RIP.

2 Recopier et compléter la table des coûts du nœud R2 selon le protocole RIP.

Nœud R2	
Destination	Coût
R1	1
...	...

SUJET

On s'intéresse maintenant au protocole de routage OSPF. Ce dernier cherche à minimiser la somme des coûts des liaisons entre les routeurs empruntés par un paquet. Le coût c d'une liaison est donné par :

$$c = \frac{10^8}{d}$$

où d est la bande passante en bit/s de la liaison.

La bande passante des liaisons FTTH (fibre optique : Fiber To The Home) est de 10 Gbit/s et celle des liaisons FastEthernet est de 100 Mbit/s.

3 Calculer le coût d'une liaison de communication par la technologie FTTH.

Sur la figure 3, les liaisons sont représentées par un style de trait différents en fonction de leur débit. La légende y est indiquée en bas à gauche.

4 Avec le protocole OSPF, donner le chemin pris par un paquet partant du service de neurologie à destination du service d'imagerie.

Partie B : le dossier médical d'un patient

Lorsqu'un patient se présente dans un hôpital, on lui demande sa carte de sécurité sociale (carte Vitale) ainsi qu'une pièce d'identité. En effet, les secrétaires des différents services doivent mettre à jour les données du dossier médical du patient. Ainsi le dossier permet aux médecins de l'hôpital d'avoir accès aux fichiers contenant les divers examens effectués par le patient (les clichés des IRM ou radios, les résultats de ses prises de sang, etc.).

Pour gérer et partager toutes ces données, le service informatique de l'hôpital a créé une base de données dont le modèle relationnel est donné par le schéma présenté sur la figure 3. Sur ce schéma, un attribut souligné indique qu'il s'agit d'une clé primaire et un dièse # avant un attribut indique qu'il s'agit d'une clé étrangère.

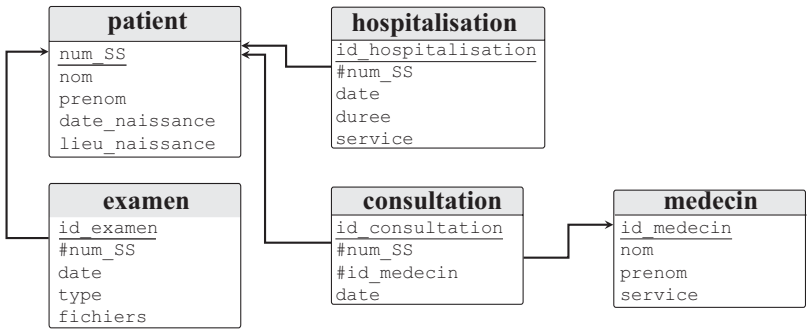


Figure 10.3 – Base de données de l'hôpital

Toutes les clés primaires de cette base de données sont de type `INT` (sauf `num_SS` qui est de type `TEXT`), les dates sont de type `DATE` (`JJ/MM/AAAA`) et tous les autres attributs sont de type `TEXT`.

L'énoncé de cet exercice utilise tout ou une partie des mots clefs du langage SQL suivants : `SELECT`, `DISTINCT`, `FROM`, `WHERE`, `JOIN ... ON`, `UPDATE ... SET`, `DELETE`, `INSERT INTO ... VALUES`, `LIKE`, `BETWEEN`.

La commande `LIKE 'a%'` permet de rechercher toutes les chaînes de caractères qui commencent par un « a » et `LIKE '%a'` celles qui se terminent par un « a ».

Patient				
num_SS	Nom	prenom	date_naissance	lieu_naissance
2 95 07 75 019 089 45	Baujean	Emma	03/07/1995	Paris
1 01 12 14 118 326 20	Tardus	Kylian	16/12/2001	Caen
2 12 10 31 555 673 33	Delpuis	Sarah	23/10/2012	Toulouse
1 90 03 49 007 512 42	Montpart	Vincent	30/03/1990	Angers

1. Décrire simplement le résultat obtenu avec la requête SQL ci-dessous :

```
SELECT nom,prenom FROM patient WHERE num_SS LIKE '1%';
```

2. Écrire une requête SQL permettant d'afficher le numéro de Sécurité Sociale des patients hospitalisés dans le service intitulé orthopédique durant l'année 2023.
3. Écrire une requête SQL permettant d'afficher le type et la date de chaque examen de la patiente Mme Baujean Emma.
4. Écrire une requête SQL permettant d'afficher le nom et le prénom de tous les patients du médecin M. ARNOS Pierre.

Partie C : la sécurité des mots de passe d'un médecin

Pour utiliser les ordinateurs de cet hôpital, tout le personnel doit saisir son identifiant puis son mot de passe. Pour davantage de sécurité, ce dernier doit être fort et changé régulièrement.

On appelle « mot de passe fort » une chaîne de caractères composée :

- d'au minimum 12 caractères ;
- d'au moins 2 majuscules ;
- d'au moins 2 chiffres ;
- d'au moins 2 symboles parmi #@ ! ?%<>=€$+-*/&.

Par exemple, un médecin utilise le mot de passe fort : `@20!HôPiTaL&24#`.

On dispose des méthodes :

- `isalpha` qui permet de tester si un caractère est une lettre ;
- `isupper` qui permet de tester si un caractère est une majuscule ;
- `isdigit` qui permet de tester si un caractère est un chiffre.

SUJET

```
>>> 'A'.isalpha()
True
>>> 'a'.isupper()
False
>>> '5'.isdigit()
True
```

De plus, on affecte tous les symboles dans une variable globale `liste_symboles` de type `str` :

```
liste_symboles = '#@!?%<>=€$+-*/&'
```

1 On considère une fonction `mdp_fort` qui prend en paramètre une chaîne de caractères `mdp` et qui renvoie `True` si `mdp` est un mot de passe fort et `False` sinon. Compléter le script de cette fonction `mdp_fort` en recopiant les lignes 2, 10, 12 et 14 sur votre copie :

```
1  def mdp_fort(mdp):
2    if ... :
3      return False
4    majuscules = 0
5    chiffres = 0
6    symboles = 0
7    for caractere in mdp :
8      if caractere.isupper():
9        majuscules += 1
10     if ... :
11       chiffres += 1
12     if ... :
13       symboles += 1
14   if ... :
15     return False
16   return True
```

Pour aider le personnel, le service informatique a mis en place une fonction `creation_mdp` permettant de générer aléatoirement un mot de passe. Elle prend en paramètres quatre entiers de type `int` :

- la longueur `n` du mot de passe ;
- le nombre `nbr_m` de majuscules qu'il doit contenir au minimum ;
- le nombre `nbr_c` de chiffres qu'il doit contenir au minimum ;
- le nombre `nbr_s` de symboles qu'il doit contenir au minimum.

Cette fonction renvoie une chaîne de caractères respectant les conditions précédentes.

2 Compléter le script de cette fonction `creation_mdp` en recopiant les lignes 9 et de 14 à 20 sur votre copie.

```
1  def creation_mdp(n, nbr_m, nbr_c, nbr_s):
2    mdp = ''
3    caracteres='abcdefghijklmnopqrstuvwxyz' + \
4        'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
5             '#@!?%<>=\u20AC$+-*/&'
6    majuscules = 0
7    chiffres = 0
8    symboles = 0
9    while ... :
10     # la variable 'c' contient un caractère
11     # choisi aléatoirement dans la variable
12     # 'caracteres'
13     c = choice(caracteres)
14     if ... :
15       ...
16     if ... :
17       ...
18     if ... :
19       ...
20     mdp = ...
21   return mdp
```

Si un personnel n'utilise pas le générateur de mot de passe, il a tendance à y insérer des mots de la langue française. Par exemple, le mot de passe `@20!HôPiTaL&24#` est fort mais on y trouve le mot « hôpital ». Si un mot de passe ne contient pas de mots de 4 lettres ou plus de la langue française ou étrangère, on dit que c'est un « mot de passe extra fort ». On dispose de la variable `dicoFR` (de type `list`) qui contient tous les mots de la langue française en minuscules.

On dispose également de la fonction `transforme` qui prend en paramètre une chaîne de caractères et qui renvoie une nouvelle chaîne de caractère en transformant toutes les lettres majuscules en minuscules. Par exemple :

```
>>> transforme('@20!HôPiTaL&24#')
@20!hôpital&24#
```

Pour simplifier, on suppose que les mots présents dans un mot de passe sont entiers (pas d'abréviation) et sont séparés par des chiffres ou des symboles. On considère une fonction `recherche_mot` qui prend en paramètre un mot de passe `mdp` (de type `str`) et qui renvoie des mots présents (de type `str`) dans `mdp`.

SUJET

Exemple :

```
>>> recherche_mot('@20!NeUrO&24#')
['neuro']
>>> recherche_mot('Chef!14Neuro@85!iRm')
['chef','neuro','irm']
```

3 Compléter le script de cette fonction `recherche_mot` en recopiant les lignes 5, 6, 15, 16, 18 sur votre copie.

```
1  def recherche_mot(mdp):
2    mot = transforme(mdp)
3    trouve = []
4    i = 0
5    while ... :
6      if ... : # si le caractère est un chiffre
7        i = i + 1
8      elif mot[i] in liste_symboles:
9        i = i + 1
10     else:
11       # si le caractère est une lettre, on prend
12       # les lettres qui la suivent jusqu'au moment
13       #  où on trouve un chiffre ou un symbole
14       chaine = ''
15       while ... :
16         chaine = ...
17         i = i + 1
18       ...
19   return trouve
```

4 Écrire une fonction `mdp_extra_fort` qui prend en paramètre une chaîne de caractères `mdp` et qui renvoie `True` si `mdp` est un mot de passe extra fort et `False` sinon.

1 Environnement Linux

Énoncé p. 321

D'après bac 2024 Étranger J2

Partie A : arborescence de fichiers

1. La commande `ls` permet de lister le contenu d'un répertoire. Le répertoire `documents` est un répertoire fils du répertoire courant. Une commande possible est donc :
   `ls documents`
2. La commande `mv` effectue le déplacement (*move*) de l'origine (en chemin relatif) `../../multimedia` (on remonte donc de deux répertoires parents et on redescend vers `multimedia`) vers la destination (en chemin absolu) `/home/documents`. L'ensemble du répertoire multimedia et ses fils se retrouvent donc comme fils du répertoire `documents`.
3. La classe `Arbre` présentée ici permet de décrire un arbre binaire, chaque nœud ne pouvant posséder qu'au plus deux sous-arbres (droite et gauche). Or l'arborescence de fichiers présente des répertoires qui peuvent contenir plus de deux répertoires fils (tel `documents` qui en possède initialement 3). Donc la classe présentée ne convient pas pour cette modélisation.
4. Il s'agit d'un parcours en profondeur car la méthode `parcours` affiche le nœud rencontré puis s'appelle récursivement sur ses sous-arbres gauche, puis droit. Cet ordre correspond à un parcours préfixe.
5. Un parcours en largeur affiche les nœuds, en commençant par la racine, puis niveau par niveau, par distance croissante à la racine : `home`, `documents`, `multimedia`, `cours`, `administratif`, `personnel`, `images`, `videos`, `films` (pour l'arborescence initiale, figure 1).

Partie B : implémentation de la classe Dossier

1.
```
def est_vide(self):
  return self.fils == []
```

2.
```
var_films = Dossier('films', [])
var_videos = Dossier('videos', [var_films])
var_images = Dossier('images', [])
var_multimedia = Dossier('multimedia', [var_images,
     var_videos])
```

ou

```
var_multimedia = Dossier('multimedia',
    [Dossier('images', []),
    Dossier('videos', [Dossier('films', [])])
    ])
```

CORRIGÉ

3

```
def parcours(self):
  print(self.nom)
  for f in self.fils:
    f.parcours()
```

4 Chaque répertoire dispose toujours d'un nombre fini de répertoires fils. De plus, il existe toujours, dans un répertoire vide (feuille de l'arbre). Donc le parcours termine toujours (arrêt des appels récursifs).

5

```
def parcours(self):
  for f in self.fils:
    f.parcours()
  print(self.nom)
```

6 La commande `ls` n'affiche que les répertoires fils et les fichiers contenus dans le répertoire courant. La méthode `parcours` affiche récursivement le contenu de tous les répertoires fils. En cela, elle se rapproche de la commande `ls -R`.

7

```
def mkdir(self, nom):
  var_fils = Dossier(nom, [])
  self.fils.append(var_fils)
```

8

```
def contient(self, nom_dossier):
  if self.nom == nom_dossier:
    return True
  for fils in self.fils:
    return fils.contient(nom_dossier)
  return False
```

9 Il s'agit ici de définir une méthode récursive prenant en argument le nom du dossier (`nom_dossier`) dont on cherche le répertoire parent. On parcourt tous les éléments `fils` de la liste `self.fils`.

- Si l'on trouve `nom_dossier` parmi cette liste, alors on retourne `self.nom` (ou `self` si c'est l'instance qui doit être retournée plutôt que son nom).
- Sinon, on appelle récursivement la méthode sur l'instance `fils`.

Une implémentation possible (non demandée) est la suivante :

```
def parent(self, nom_dossier):
  for fils in self.fils:
    if fils.nom == nom_dossier:
      return self
    else:
      parent = fils.parent(nom_dossier)
      if parent:
        return self.parent
  return None
```

10 On peut stocker, dans chaque instance de `Dossier`, l'instance du répertoire parent (`None` si racine). Cela occupe plus de place en mémoire mais permet une recherche plus efficace.

```
def __init__(self, nom, liste, parent=None):
self.nom = nom
self.fils = liste # liste d'objets de la classe
  Dossier
self.parent = parent
# met à jour les parents des répertoires fils
for fils in self.fils:
  fils.parent = self
```

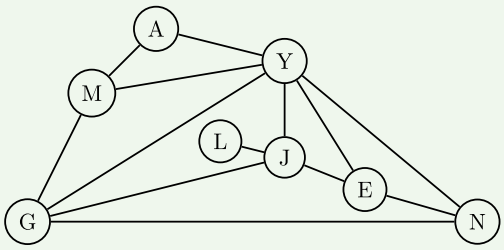
2 Groupe de personnes

Énoncé p. 323

D'après bac 2024 Amérique Nord J1

Partie A : matrice d'adjacence

1



2

```
# sommets G, J, Y, E, N, M, A, L
matrice_adj = [
        [0, 1, 1, 0, 1, 1, 0, 0], # G
        [1, 0, 1, 1, 0, 0, 0, 1], # J
        [1, 1, 0, 1, 1, 1, 1, 0], # Y
        [0, 1, 1, 0, 1, 0, 0, 0], # E
        [1, 0, 1, 1, 0, 0, 0, 0], # N
        [1, 0, 1, 0, 0, 0, 1, 0], # M
        [0, 0, 1, 0, 0, 1, 0, 0], # A
        [0, 1, 0, 0, 0, 0, 0, 0]] # L
```

3

```
>>> indice(sommets, 'G')
0
>>> indice(sommets, 'Z')
None
```

CORRIGÉ

4

```
def nb_amis(L, m, s):
  pos_s = indice(L, s)
  if pos_s is None:
    return None
  amis = 0
  for i in range(len(m)):
    amis += m[i][pos_s]
  return amis
```

5 Gabriel a 4 amis, ce qui se représente dans la matrice d'adjacence par une liste `[0, 1, 1, 0, 1, 1, 0, 0]` (soit 4 occurrences de `'1'`) :

```
>>> nb_amis(sommets, matrice_adj, 'G')}
4
```

Partie B : dictionnaire de listes d'adjacence

1 Dans `{c : v}`, `c` est la clé et `v` la valeur qui lui est associée.

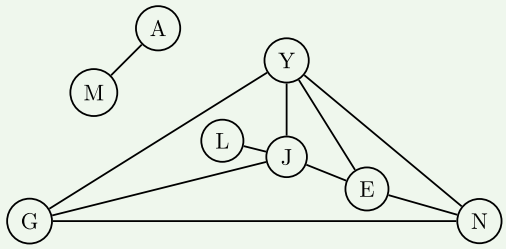
2 Le dictionnaire de listes d'adjacence est :

```
graphe = {'G' : ['J', 'Y', 'N', 'M'],
                'J' : ['G', 'Y', 'E', 'L'],
                'Y' : ['G', 'J', 'E', 'N', 'M', 'A'],
                'E' : ['J', 'Y', 'N'],
                'N' : ['G', 'Y', 'E'],
                'M' : ['G', 'Y', 'A'],
                'A' : ['Y', 'M'],
                'L' : ['J']}
```

3 Avec un dictionnaire de liste d'adjacence, dès lors que `s` est dans `d`, permet de trouver le nombre d'amis de `s` en renvoyant la longueur de la liste associée à la clé `s` :

```
def nb_amis(dict_adj, sommet):
  return len(dict_adj[sommet])
```

4 Désormais, les sommets `'M'` et `'A'` ne sont plus liés qu'entre eux et ne sont plus atteignables par les autres (*cf.* nouveau graphe ci-dessous).



Ainsi, le cercle d'amis de Lou est constitué de Jade, Yanis, Emma, Gabriel et Nina.

5

```
visites = []

def parcours_en_profondeur(d, s):
  visites.append(s)
  for v in d[s]:
    if v not in visites:
      parcours_en_profondeur(d, v)
  return visites
```

3 Gestion d'un hôpital

Énoncé p. 326

D'après bac 2024 Polynésie J2

Partie A : le réseau informatique d'un hôpital

1 En utilisant le protocole RIP, le chemin entre R4 et R2 minimise le nombre d'intermédiaires.

On a donc :

R4 – R8 – R1 – R2 ou R4 – R8 – R7 – R2.

2 On remplit la table des coûts du nœud R2 en comptant le nombres de routeurs à joindre pour que le paquet chemine jusqu'à chaque routeur.

Nœud R2	
Destination	Coût
R1	1
R3	3
R4	3
R5	2
R6	3
R7	1
R8	2
R9	2

3 Le débit d'une liaison FTTH est : $d = 10 \times 10^9$, d'où :

$$c = \frac{10^8}{10 \times 10^9} = 10^{-2} = 0{,}01.$$

Or, les coûts inférieurs à 1 dans OSPF sont arrondis à 1.

La réponse est donc $c = 1$.

CORRIGÉ

4 Les liaisons FTTH ont un coût de 0,01. Les liaisons FastEthernet ont un coût de 1. Les autres ont un coût de 1. Le protocole OSPF minimise le coût total du chemin. On trouve alors R8 – R9 – R1 – R2 avec un coût total de 4.

Partie B : le dossier médical d'un patient

1 Le résultat de la requête contient les noms et prénoms de tous les patients dont le numéro de Sécurité Sociale commence par un 1 (hommes).

2 On réalise une jointure (interne) entre les tables `patient` et `hospitalisation`. Les doublons éventuels (un même patient hospitalisé plusieurs fois durant l'année dans ce service) avec `DISTINCT`. Les dates n'étant pas stockées au format `TEXT`, il ne semble pas adapté d'utilier `LIKE` mais plutôt des tests sur `hospitalisation.date` avec l'opérateur `BETWEEN`.

```
SELECT DISTINCT patient.num_SS
FROM patient
JOIN hospitalisation ON patient.num_SS =
    hospitalisation.num_SS
WHERE hospitalisation.service = 'orthopédique'
AND hospitalisation.date BETWEEN '01/01/2023' AND '
    31/12/2023'
```

3 Là aussi, on utilise une jointure interne. Les doublons ne sont pas éliminés.

```
SELECT examen.type, examen.date
FROM examen
JOIN patient ON examen.num_SS = patient.num_SS
WHERE patient.nom = 'Baujean'
AND patient.prenom = 'Emma'
```

4 La jointure interne est réalisée entre les trois tables, avec élimination des doublons.

```
SELECT DISTINCT patient.nom, patient.prenom
FROM patient
JOIN consultation ON patient.num_SS = consultation.
    num_SS
JOIN medecin ON consultation.id_medecin = medecin.
    id_medecin
WHERE medecin.nom = 'ARNOS'
AND medecin.prenom = 'Pierre'
```

Partie C : la sécurité des mots de passe d'un médecin

1 La fonction renvoie `False` dès qu'une des conditions n'est pas remplie, `True` sinon. Pour cela, elle teste la longueur de la chaîne `mdp` puis utilise 3 variables pour compter les caractères (lettres, chiffres et symboles).

```
def mdp_fort(mdp):
  if len(mdp) < 12:
    return False
  majuscules = 0
  chiffres = 0
  symboles = 0
  for caractere in mdp :
    if caractere.isupper():
      majuscules += 1
    if caractere.isdigit():
      chiffres += 1
    if caractere in liste_symboles:
      symboles += 1
  if majuscules<2 or chiffres<2 or symboles<2:
    return False
  return True
```

2 On reprend le même principe que précédemment, avec l'utilisation de compteurs. La chaîne retournée est le résultat d'une concaténation caractère par caractère, qui s'arrête lorsque toutes les conditions sont réunies.

```
def creation_mdp(n, nbr_m, nbr_c, nbr_s):
  mdp = ''
  caracteres='abcdefghijklmnopqrstuvwxyz' + \
      'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
            '#@!?%<>=\u20AC$+-*/&'
  majuscules = 0
  chiffres = 0
  symboles = 0
  while len(mdp) < n or majuscules < nbr_m or
    chiffres < nbr_c or symboles < nbr_s:
    c = choice(caracteres)
    if c.isupper():
      majuscules += 1
    if c.isdigit():
      chiffres += 1
    if c in liste_symboles:
      symboles += 1
    mdp = mdp + c
  return mdp
```

CORRIGÉ

3 La chaîne `mdp` est parcourue caractère par caractère : si le caractère est un chiffre ou symbole, on passe au caractère suivant. Sinon, on constitue une chaîne par concaténation, tant que le caractère est une lettre. La fin de la chaîne ou l'apparition d'un chiffre ou d'un symbole arrête la concaténation et la chaîne construite est ajoutée dans la liste à retourner.

```
def recherche_mot(mdp):
  mot = transforme(mdp)
  trouve = []
  i = 0
  while i < len(mot):
    if mot[i].isdigit() : # si le caractère est un
    chiffre
      i = i + 1
    elif mot[i] in liste_symboles:
      i = i + 1
    else:
      chaine = ''
      while i < len(mot) and mot[i].isalpha():
        chaine = chaine + mot[i]
        i = i + 1
      trouve.append(chaine)
  return trouve
```

4 À l'aide de la fonction `recherche_mot`, les mots à tester sont extraits de la chaîne `mdp`. Si au moins un de ces mots extraits est de taille supérieure ou égale à 4 et qu'il est présent dans le dictionnaire, alors on retourne `False`. Après avoir testé tous les mots, on retourne `True` car aucun mot de 4 lettres au moins n'a été trouvé dans le dictionnaire utilisé.

```
def mdp_extra_fort(mdp):
  for mot in recherche_mot(mdp):
    if len(mot) > 3 and mot in dicoFR:
      return False
  return True
```

Annexe

Liste des vidéos