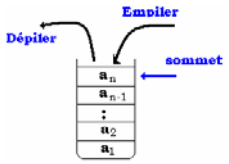


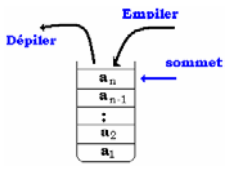
Les Structures De Données

Cours & Activité Pratique

Nom : Prénom : Date :

Table des matières

NSI
NUMÉRIQUE ET SCIENCES INFORMATIQUES



Les Structures De Données

Cours & Activité Pratique

1 Introduction

Qu'est-ce qu'une Structure de Données ?

On pourrait définir le terme « Structure de données » en une phrase :

Une structure de données est une manière d'organiser les données pour les traiter plus facilement.

Une structure de données est une manière d'organiser l'information. Elle permet de garantir certaines propriétés lorsqu'il s'agit d'accéder aux données, et en particulier d'optimiser le temps d'accès à l'information. Elles sont donc particulièrement utiles lorsqu'il s'agit de traiter des collections d'éléments. Et plus les collections sont grosses, plus le gain offert par la structuration des données est conséquent.

Pour prendre un exemple simple de la vie quotidienne, on peut présenter des numéros de téléphone par département, par nom, par profession (comme les Pages jaunes), par numéro téléphonique (comme les annuaires destinés au télémarketing), par rue et/ou une combinaison quelconque de ces classements. À chaque usage correspondra une structure d'annuaire appropriée.
En organisant d'une certaine manière les données, on permet un traitement automatique de ces dernières plus efficace et rapide.

Distinction avec le « type de donnée »

Un type de données, ou simplement type, définit le genre de contenu d'une donnée et les opérations pouvant être effectuées sur la variable correspondante. Par exemple:

- « int » représente un entier, on peut faire des additions, des multiplications. La division est entière et on peut faire des décalages de bits …
- « date » représente une date, l'addition aura un certain sens, on pourra écrire la date sous certaines formes (jj/mm/aaaa ou « 13 Septembre 2020 »)

Les types les plus communs sont :

- int, float, double, char, boolean et le type pointeur
- Entier, réel, chaîne, caractère, booléen, pointeur

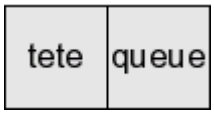
2 Les Structures linéaires

2.1 Les listes

Une liste est une structure de données permettant de regrouper des données de manière à pouvoir y accéder librement. La liste est une structure de données extrêmement utilisée. En particulier, elle a joué un rôle majeur dans le langage LISP (de l'anglais « LISt Processing ») et reste très présente dans les nombreux langages qui s'en sont inspirés.

Définition
La liste L est composée de 2 parties (champs) :

- sa tête qui correspond au dernier élément ajouté à la liste
- sa queue qui correspond au reste de la liste.

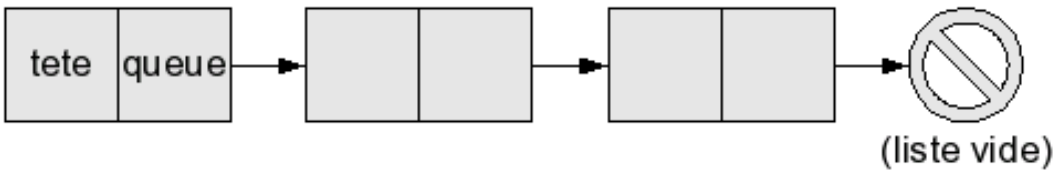


Une liste est :

- soit la liste vide ;
- soit une cellule à deux champs : un champ « tête » contenant un élément, et un champ « queue » contenant l'adresse d'une autre liste.

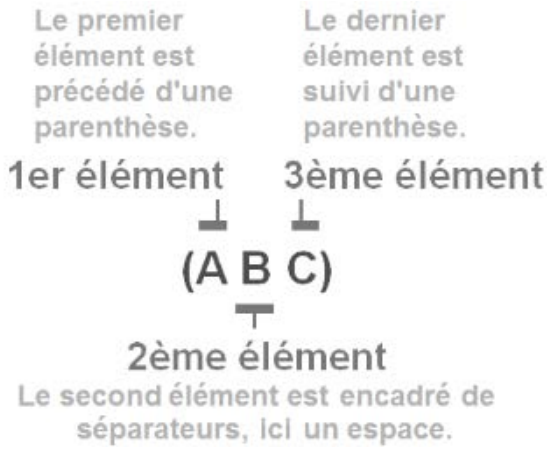
Par conséquent, une liste est "soit vide, soit un élément suivi d'une liste".
Une liste peut donc être la liste vide (0 élément), ou un élément suivi de la liste vide (1 élément), ou un élément suivi d'un élément suivi de la liste vide (2 éléments), et ainsi de suite...

La forme de la structure « Liste »
La liste peut être représentée selon la forme suivante :



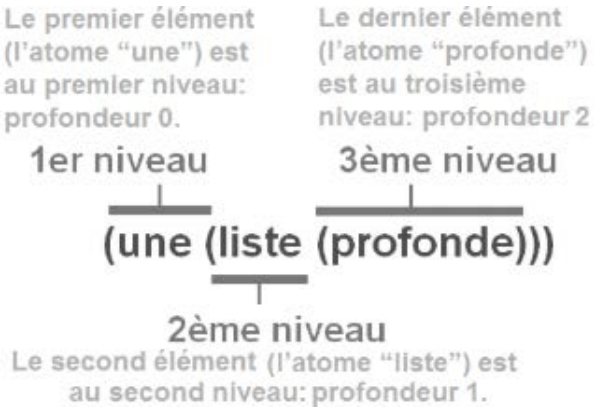
Longueur d'une liste L

Une liste L est une donnée composée d'objets dénombrables. Il est possible de faire ou non la distinction entre les objets eux-mêmes. Par conséquent chaque objet compte pour 1. La somme des objets d'une liste constitue sa longueur.



Profondeur d'une liste

Elle est calculée en fonction des éléments contenus dans la liste. Les parenthèses pourront aider à calculer la profondeur d'une liste. Ainsi la première parenthèse dénote la profondeur 0. La parenthèse suivante indique la profondeur 1 et ainsi de suite selon les imbrications des parenthèses.



Voici quelques opérations qui peuvent être effectuées sur une liste :

- créer une liste vide (L=vide() ; on a créé une liste L vide)
- tester si une liste est vide (estVide(L) renvoie vrai si la liste L est vide)
- ajouter un élément en tête de liste (ajouteEnTete (*elt* , L) avec L une liste et « *elt* » l'élément à ajouter)
- supprimer la tête « *t* » d'une liste L et renvoyer cette tête *t* (supprEnTete(L))
- Compter le nombre d'éléments présents dans une liste (compte(L) renvoie le nombre d'éléments présents dans la liste L)
- Obtenir une nouvelle liste à partir d'une liste et d'un élément (L1 = cons(*elt* , L)). On peut enchaîner les « cons » et obtenir ce genre de structure : cons(*elt1*, cons(*elt2* , cons(*elt3* , L)))

Prenons quelques exemples :

L=vide() => on a créé une liste vide
estVide(L) => renvoie Vrai (True)
ajoutEnTete(7, L) => la tête de la liste L contient maintenant l'élément 7, sa queue contient une liste vide
estVide(L) => renvoie Faux (False)
ajoutEnTete(4, L) => la tête de la liste L correspond à 4, la queue contient l'élément 7 suivi d'une liste vide
ajoutEnTete(6, L) => la tête de la liste L correspond à 6, la queue contient l'élément 4 suivi de l'élément 7 suivi d'une liste vide
t = supprEnTete(L) => la variable t vaut 6, la tête de L correspond à 4 et la queue contient l'élément 7 suivi d'une liste vide
L1 = vide()
L2 = cons(8, cons(5, cons(3, L1))) => La tête de L2 correspond à 8 et la queue contient l'élément 5 suivi de l'élément 3 suivi de la liste vide L1

A faire vous-même I

Soit l'enchainement d'instructions :

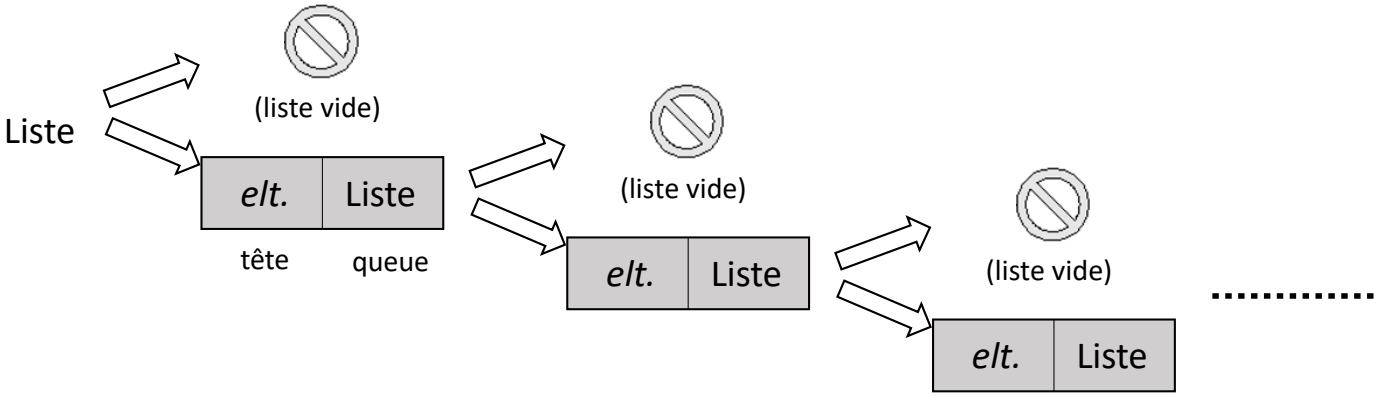
- L1 = vide()
- ajoutEnTete(13, L1)
- estVide(L1)
- ajoutEnTete(11, L1)
- ajoutEnTete(9, L1)
- L2 = cons(8, cons(6, cons(5, cons (4, cons(2, cons(0,L1))))))

Expliquez clairement le contenu de chaque étape ci-dessous:

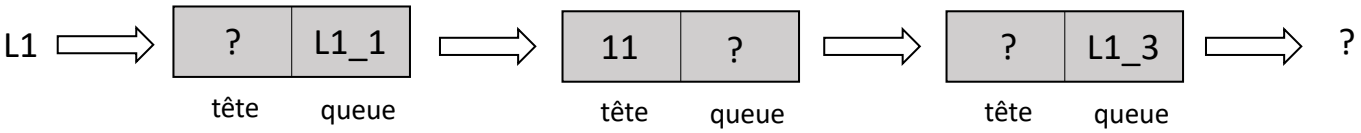
L1 = vide() => On cree la liste L1 vide
ajoutEnTete(13,L1) => La tete de la liste L1 contient l'élément 13, sa queue contient une liste vide
estVide(L1) => renvoie False
ajoutEnTete(11,L1) => La tete de la liste L1 contient l'élement 11 suivie de l'élément 13 suivie d'une liste vide
ajoutEnTete(9,L1) => La tete de la liste L1 contient l'élement 9 suivie de l'élément 11 suivie de lélément 13 suivie d'une liste vide
L2 = cons(8,cons(6,cons(5,cons(4,cons(2,cons(0,L1)))))) => La liste L2 contient en tete l'élément 8 et est suivie de l'élément 6,5,4,2,0,suivie de la liste L1

A faire vous-même II

Nous avons vu qu'une une liste était "soit vide, soit un élément suivi d'une liste". Il est intéressant de noter que cette définition est récursive. En effet, nous pouvons nous en convaincre avec le schéma suivant :



En reprenant la liste L1 du AFVM I, compléter le schéma suivant :




```
L1 => 13 L1 1 => 11 L1 2 => 9 L1 3 => liste vide
```

A faire vous-même III

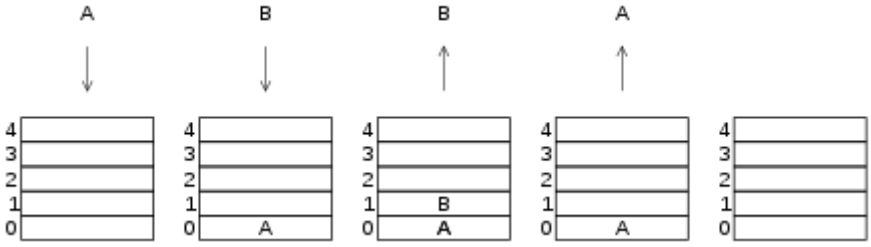
Réalisez ci-dessous la représentation schématique de la liste L2 (afvm I) :

L2 => 8 L2_1 => 6 L2_2 => 5 L2_3 => 4 L2_4 => 2 L2_5 => 0 L2_6 =>
13 L1_1 => 11 L1_2 => 9 L1_3 => liste vide

2.2 Les piles

Définition

En informatique, une pile (en anglais stack) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (en anglais LIFO pour last in, first out), ce qui veut dire, qu'en général, le dernier élément, ajouté à la pile, sera le premier à en sortir. *(Wikipedia)*



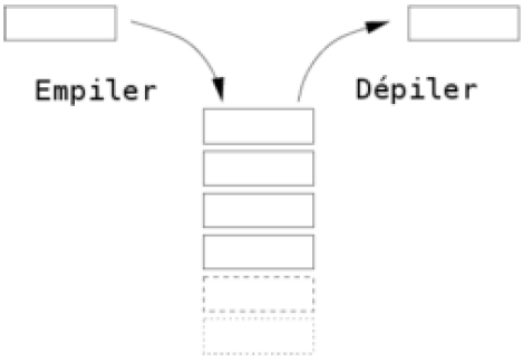
Communément, on compare le fonctionnement de cette structure à celui d'une pile d'assiettes.

Quelques applications des piles

- Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant le bouton « Afficher la page précédente ».
- L'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile. Par exemple, « (4 + (5 + 6)) * 3 » s'écrit « 4 5 6 add add 3 mul »
- La fonction « Annuler la frappe » (en anglais « Undo ») d'un traitement de texte

Voici la liste des opérations que l'on peut réaliser sur une pile :

- savoir si une pile est vide (pile_vide?)
- empiler un nouvel élément sur la pile (push)
- récupérer l'élément au sommet de la pile tout en le supprimant. On dit que l'on dépile (pop)
- accéder à l'élément situé au sommet de la pile sans le supprimer de la pile (sommet)
- connaitre le nombre d'éléments présents dans la pile (taille)



Nous allons simuler le fonctionnement d'une pile avec la liste python. Nous pouvons ainsi :

- savoir si une pile est vide (pile_vide?) : renvoie vrai si la pile est vide, faux sinon
- « empiler » un nouvel élément sur la pile p en utilisant la fonction append()
- « dépiler » : récupérer l'élément au sommet de la pile p tout en le supprimant. On utilise la fonction pop()
- accéder à l'élément situé au sommet de la pile avec p[-1]
- connaitre le nombre d'éléments présents dans la pile (taille) avec la fonction « built-in » len()

A faire vous-même IV

Soit la pile p_1 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 24, 25]. A l'aide de l'IDLE Python, saisir les instructions qui permettent de répondre aux questions suivantes :

a. Initialiser p_1 en initialisant une liste Python. Quel est le sommet de p_1 ?

```
>>> p1 = [2,3,5,7,11,13,17,19,23,24,25]
>>> p1[-1]
25
```

b. Dépiler les deux premiers éléments de p_1 ?

```
>>> p1.pop()
25
>>> p1.pop()
24
```

c. Selon vous, quel est maintenant le sommet de p_1 ? Vérifiez-le en utilisant une instruction Python ?

```
23
>>> p1[-1]
23
```

d. On souhaite combler p_1 avec la liste des nombres premiers inférieurs à 40. A l'aide d'un « itérateur » et d'une boucle « for », empiler les nombres premiers afin d'atteindre l'objectif.

```
for nb in range(24,40):
    i,premier = 2,True
    while i < nb and premier:
        if nb%i == 0: premier = False
        i+=1
    if premier:p1.append(nb)
```

e. Selon vous, quel est à ce stade le sommet de p_1 ? Vérifiez-le en utilisant une instruction Python ?

```
37
>>> p1[-1]
37
```

f. Selon vous, quelle est à ce stade la taille de p1 ? Vérifiez-le en utilisant une instruction Python ?

```
>>> len(p1)
12
```

g. On souhaite maintenant dépiler p_1 afin d'obtenir la liste des nombres premiers inférieurs à 20. Comment atteindre cet objectif avec un bloc d'instructions (2 lignes) ? Ecrire un programme Python qui permette d'afficher p_1 au format pile :

```
 -------
|   19  |
 -------
|   17  |
 -------
|   13  |
 -------
|   11  |
 -------
|   7   |
 -------
|   5   |
 -------
|   3   |
 -------
|   2   |
 -------
```

```
while p1[-1] > 20:
    p1.pop()

for i in range(1,len(p1)+1):
    print(" ------- ")
    if len(str(p1[-i])) == 2:
        print(f"|  {p1[-i]}   |")
    else:
        print(f"|   {p1[-i]}   |")
print(" ------- ")
```

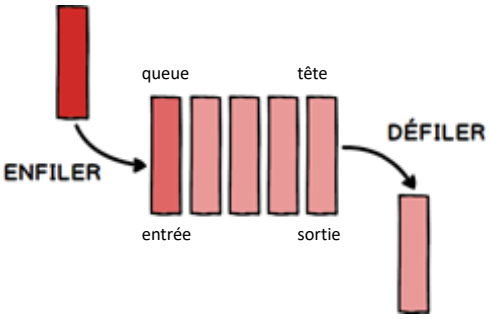
h. On souhaite maintenant dépiler p_1 jusqu'à ce qu'elle soit vide. Ecrire le bloc d'instructions le bloc d'instructions (2 lignes) qui permette atteindre cet objectif. Enfin, vérifiez que p_1 est bien vide :

```
while p1 != []:
    p1.pop()

>>> len(p1)
0
```

2.3 Les files

Définition
Une file (*queue* en anglais) est une structure de données basée sur le principe « Premier entré, premier sorti », en anglais FIFO (First In, First Out), ce qui signifie que les premiers éléments ajoutés à la file seront les premiers à être récupérés. Le fonctionnement ressemble à une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file.

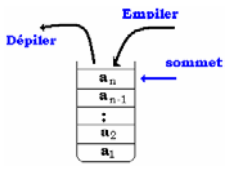


Quelques applications des files

- Il est courant d'utiliser les files pour mémoriser temporairement des transactions qui doivent attendre pour être traitées.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps « machine » à chaque tâche, sans n'en privilégier aucune.
- Un algorithme de parcours en largeur utilise une file pour mémoriser les nœuds visités.

Voici la liste des opérations que l'on peut réaliser sur une file :

- savoir si une file est vide (file vide?)
- enfiler un nouvel élément dans la file
- récupérer l'élément situé en tête de la file tout en le supprimant. On dit que l'on défile
- accéder à l'élément situé en tête de la file sans le supprimer de la file
- connaitre le nombre d'éléments présents dans la file (taille de la file)



Les Structures De Données

Cours & Activité Pratique

En Python, les listes ne simulent pas correctement la structure de donnée « file ». En effet, les objets ne peuvent être enfilés en temps constant qu'en fin de liste. Ils doivent donc être défilés en début de liste. Or l'insertion en début de liste ne se fait pas en temps constant (car Python doit décaler les autres valeurs).

Cependant, il existe un autre type de donnée qui permet de simuler une file: « deque ». Il est accessible via le module « collections ».

Les « deques » sont une généralisation des files (deque se prononce "dèque" et est l'abréviation de l'anglais double-ended queue). Ils permettent d'ajouter et retirer des éléments par leurs deux extrémités.
Voici quelques fonctions utilisables avec le type « deque » :

- d1=deque() : création d'une file vide
- d2=deque((13,15,26,)) : création d'une file à partir d'un tuple
- len(d2) permet de savoir si la file est vide
- d.pop() renvoie l'élément de tête de file
- d.appendleft(*elt*) ajoute un élément *elt* en queue de file
- d.append(*elt*) ajoute un élément *elt* en tête de file

Les « queues » et « tête » de file sont interchangeables car il existe aussi les fonctions suivantes qui s'exécutent en temps constant :

- d.popleft()
- d.append(*elt*)

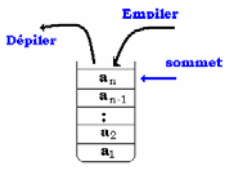
A faire vous-même V

a. A l'aide de l'IDLE Python, importer le type deque avec la commande :

```
>>> from collections import deque
```

b. Afficher l'aide sur le type. Quelles sont les fonctions utilisables ?

```
append(),appendleft(),clear(),copy(),count(),extend(),extendleft(),index(),
insert(),pop(),popleft(),remove(),reverse(),rotate()
```



Les Structures De Données

Cours & Activité Pratique

Soit la chaine de caractères 'mule'. A l'aide de l'IDLE Python, saisir les instructions qui permettent de répondre aux questions suivantes :

c. Initialiser la file d_1 à partir de la chaine de caractères 'mule' puis afficher d1

```
d = deque("mule")
print(d)
```

d. A l'aide d'une itération, afficher la file d_1 en colonne et en majuscules comme ceci :

```
M
U
L
E
```

```
for i in range(len(d)):
    print(d[i].capitalize())
```

e. On souhaite maintenant afficher le verbe « émuler » en colonne et en majuscules comme ceci :

```
E
M
U
L
E
R
```

Comment doit-on modifier la file d_1 ? Après avoir modifié le contenu de la file, écrire le programme python qui affiche la file comme ci-dessus.

```
d.appendleft("e")
d.append("r")
```

f. On souhaite maintenant afficher le verbe « simuler » en colonne et en majuscules comme ceci :

```
S
I
M
U
L
E
R
```

Comment doit-on modifier la file d_1 ? Après avoir modifié le contenu de la file, écrire le programme python qui affiche la file comme ci-dessus.

```
d.popleft()
d.appendleft("i")
d.appendleft("s")
```

g. Nous considérons maintenant la chaine de caractères ‘kay’. A l’aide de l’IDLE Python, initialiser la file d_2 à partir de la chaine ‘kay’.

```
d2 = dque("kay")
```

h. Ajouter à d_2 la chaine de caractères qui permettra d'obtenir la chaine « kayak » puis afficher d_2 en colonne et en majuscules

```
d2.append("a")
d2.append("k")
for i in range(len(d2)):
    print(d2[i].capitalize())
```

i. A l'aide de la méthode « reversed() », inverser la tête et la queue de la file d_2 et mettez le résultat dans la file d_3. Afficher en colonne et majuscules la file d_3. Est-elle différente de d_2 ? Pourquoi ?

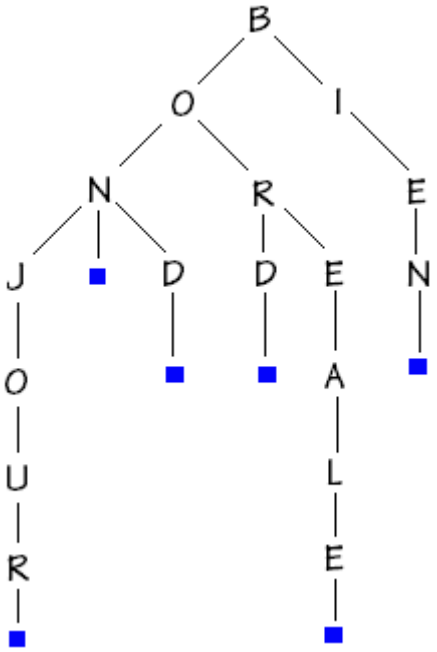
```
L'affichage est le meme car kayak est un palindrome
```

3 Les structures hiérarchiques

3.1 Les Arbres

Qu'est-ce que c'est ?

Prenons un exemple. Soient les mots BON, BONJOUR, BORD, BOND, BOREALE, BIEN. Il est possible de les ranger ainsi dans une structure d'arbre :



On appelle cette SDD un arbre lexicographique.

Définitions

Un arbre est une SDD constituée de sommets (que l'on appelle aussi nœuds), et d'arêtes (que l'on appelle aussi branches) qui relient les sommets entre eux. Dans les SDD de type « arbre », il n'y a qu'un seul chemin possible d'un nœud à un autre (il est impossible de tourner en rond).

Le nœud de départ, duquel partent les premières branches, s'appelle la ***racine*** de l'arbre.

Dans l'exemple ci-dessus, l'arbre est représenté avec la racine en haut et des branches qui descendent. Les nœuds issus d'un nœud situé immédiatement au-dessus, s'appellent les fils (descendants) et le nœud source s'appelle le père (parent).

Le nombre de chemins connectés à un nœud s'appelle le degré. Puisque dans un arbre, il n'y a qu'un seul parent par nœud (sauf pour la racine), le degré auquel on retire la valeur « 1 » donne le nombre de fils.

Les nœuds qui ne sont reliés qu'à un seul nœud sont appelés les feuilles (ce sont des nœuds de degré 1, ils n'ont pas de fils). Lorsqu'on représente un arbre verticalement avec la racine en haut, les feuilles sont tout en bas. Il suffit de retourner l'arbre pour avoir une analogie parfaite avec les arbres de la nature.

On peut nommer les nœuds. Ces noms sont appelés « étiquette ». Lorsque tous les nœuds ont une étiquette, on dit que le graphe est étiqueté. L'étiquette peut indiquer le nom du nœud mais cela peut aussi être une valeur qui donne des informations sur le nœud.

La profondeur ou ***hauteur*** d'un arbre est le nombre de branches que contient le chemin le plus long de l'arbre en partant de la racine et en descendant. C'est la distance entre la racine et le nœud. C'est le rang du dernier niveau de l'arbre, la racine étant le niveau 0, ses descendants au niveau 1, etc...

La profondeur d'un nœud est le niveau où se trouve le nœud (ou encore le nombre de branches à utiliser en partant de la racine et en descendant pour atteindre le nœud).

Certains ouvrages attribuent à la racine une profondeur de valeur 1. Dans ce cas :

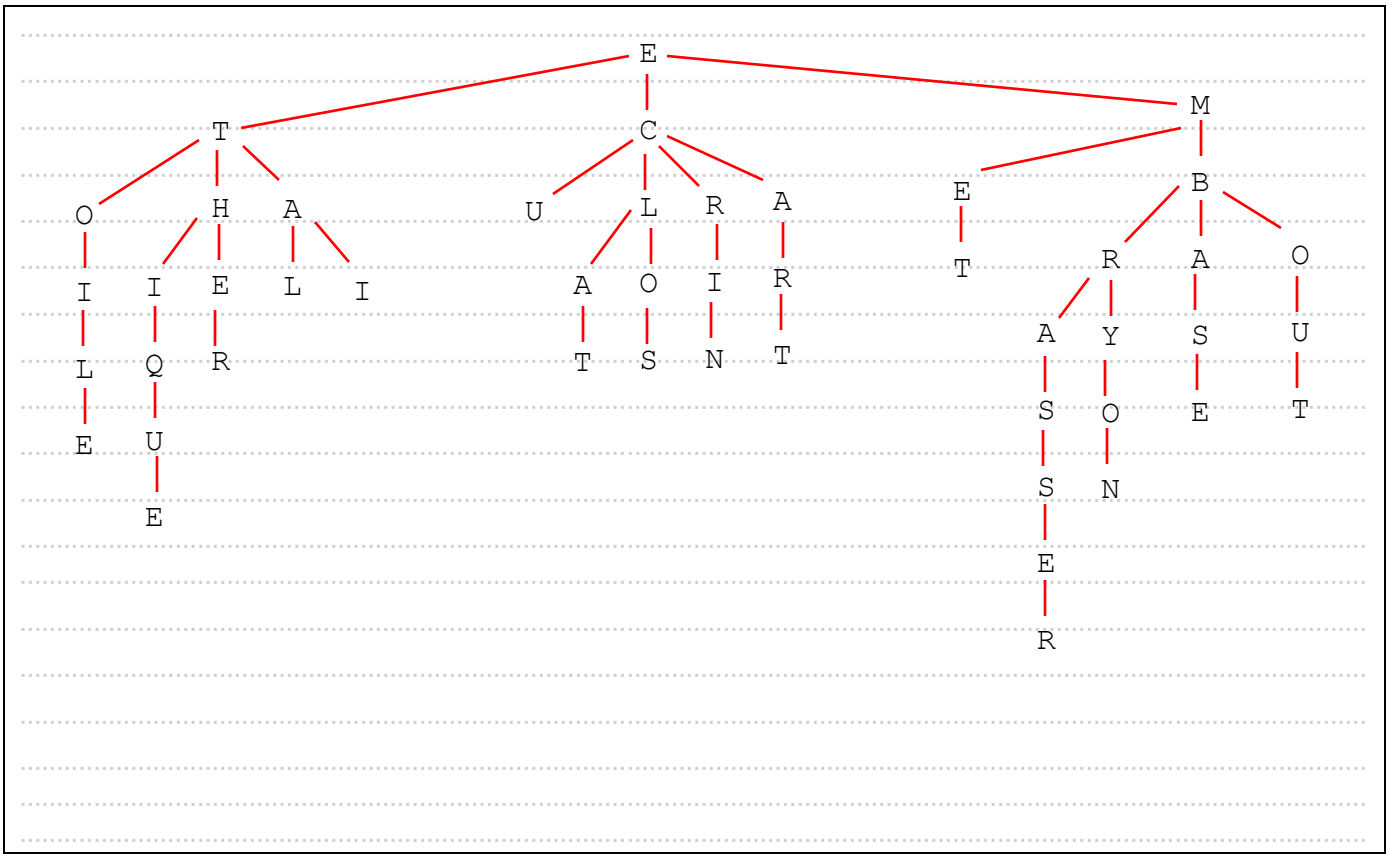
- la profondeur d'un nœud ou d'une feuille est égale à la profondeur de son prédécesseur plus 1.
- la hauteur de l'arbre est la profondeur maximale des nœuds de l'arbre.

On appelle chemin d'un nœud, la suite de nœuds par lesquels il faut passer en partant de la racine.

Enfin, remarquons que l'on peut aussi voir la SDD « arbre » comme une structure ***récursive*** : les fils d'un nœud sont des arbres qui sont eux-mêmes constitués d'arbres...

A Faire Vous-même I

a. Construire un arbre lexicographique aves les mots suivants : ECLOS, EMBASE, ECU, ETHER, EMBOUT, ETOILE, EMET, ETHIQUE, ECRIN, ETAL, EMBRYON, ECLAT, ETAI, EMBRASSER, ECART



b. Quelle est la racine de l'arbre ?

E

c. Combien y a-t-il de nœuds de profondeur 4 ? Quels sont-ils ? Classez-les par ordre alphabétique.

11
ALNQRSSTTUY

d. Cet arbre possède-t-il des nœuds de degré 4 ? de degré 5 ? Quels sont-ils ?

Noeuds de degré 4 : T B
Il y a un noeuds de degré 5 (C)

e. Quels sont les descendants du nœud L de niveau 2 ? Quel est leur(s) niveau(x) ?

A et O sont de niveau 3

f. Quelle est la hauteur de l'arbre ?

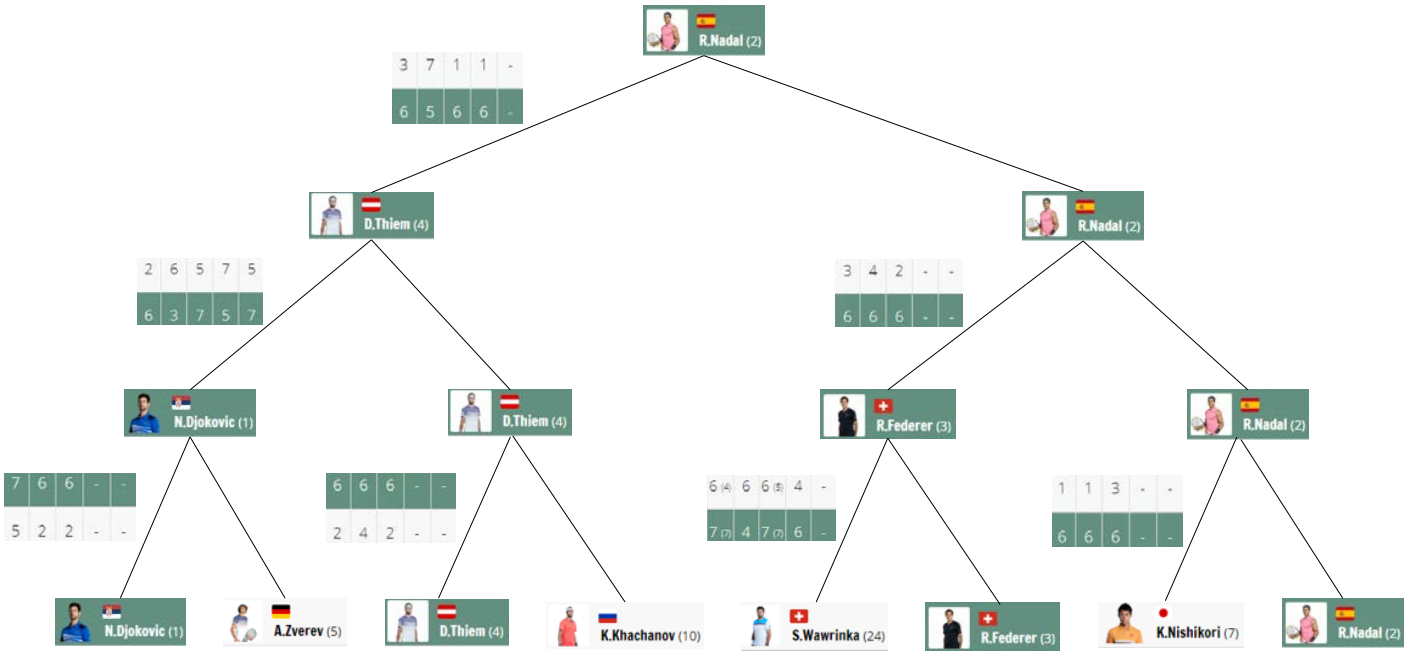
8

3.2 Les arbres binaires

Définition

C'est un arbre avec une racine. Le degré maximum de chaque nœud est 3. En d'autres termes, chaque nœud a au plus deux descendants: le fils gauche et le fils droit.

Prenons un exemple : ci-dessous la représentation des résultats des phases finales du tournoi de Roland Garros 2019.



A Faire Vous-même II

a. Quelle est la hauteur de l'arbre ?

3

b. Quel nombre maximum de branches part de chaque nœud ?

2 branche descendent de chaque noeud

c. L'arbre possède combien de feuilles ? Nommez-les.

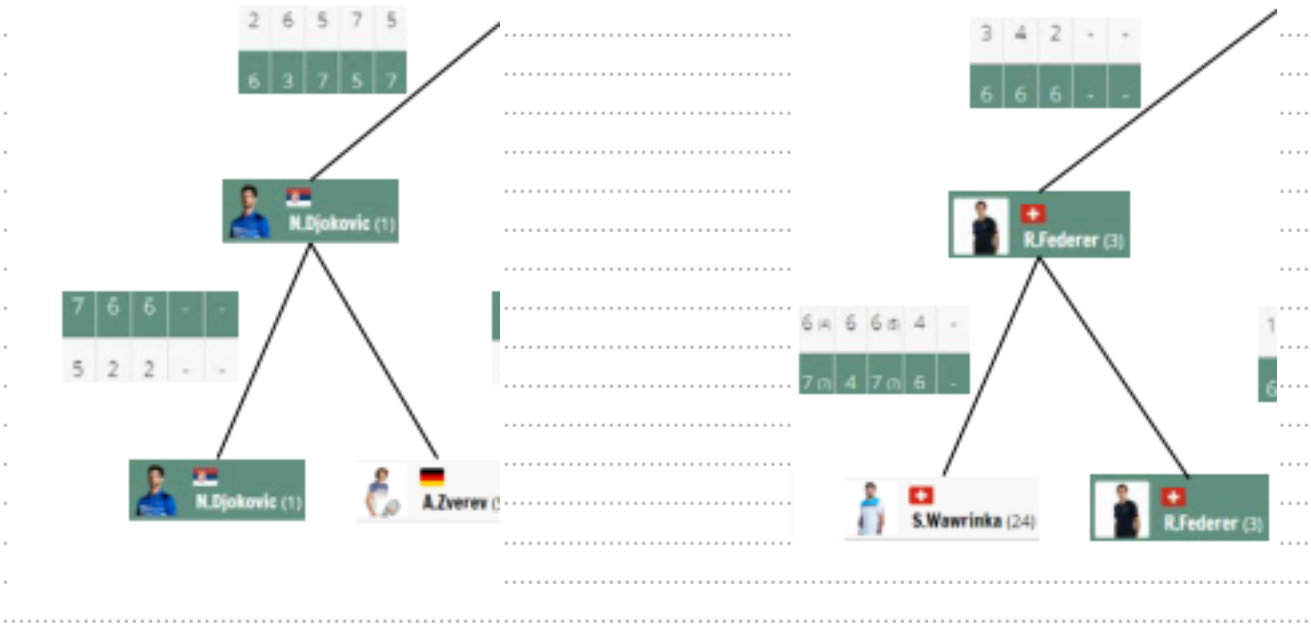
8 : N.Djokovic(1), A.Zverev(5), D.Thiem(4),K.Khachanov(10), S.Wawrinka(24), R.Federer(3), K.Nishikori(7), R.Nadal(2)

d. Quels sont les nœuds de profondeur 1 ? Quels sont leur fils droit ? Comment les appelle-ton lorsqu'ils ont atteint ce stade de la compétition ?

D.Thiem(4), R.Nadal(2) leurs fils droit sont D.Thiem(4), R.Nadal(2)
Ce sont les finalistes quand ils sont au noeuds de profondeur 1

e. Combien d'arbres sont issus du fils gauche de profondeur 1 ? Quels sont-ils ?

Il y a 2 arbres



4 Les graphes

4.1 Exemples et définitions

Exemple

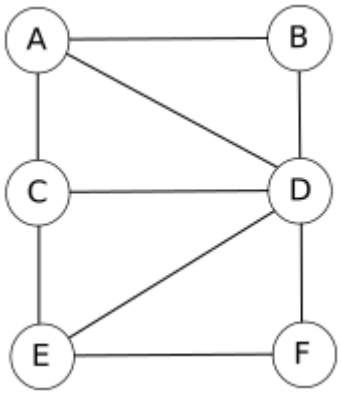
Imaginons un réseau social possédant 6 abonnés (A, B, C, D, E et F) où :

- A est ami avec B, C et D
- B est ami avec A et D
- C est ami avec A, E et D
- D est ami avec tous les autres abonnés
- E est ami avec C, D et F
- F est ami avec E et D

La description de ce réseau social, malgré son faible nombre d'abonnés, est déjà quelque peu rébarbative, alors imaginez cette même description avec un réseau social comportant des millions d'abonnés !

Il existe un moyen plus "visuel" pour représenter ce réseau social : on peut représenter chaque abonné par un cercle (avec le nom de l'abonné situé dans le cercle) et chaque relation "X est ami avec Y" par un segment de droite reliant X et Y ("X est ami avec Y" et "Y est ami avec X" étant représenté par le même segment de droite).

Voici ce que cela donne avec le réseau social décrit ci-dessus :



Auteur : David Roche

Ce genre de figure s'appelle un graphe. Les graphes sont des objets mathématiques très utilisés. Notamment en informatique où l'on s'en sert comme SDD. Les cercles sont appelés des sommets et les segments de droites qui relient 2 sommets des arêtes.

Définition

Nous avons étudié les arbres dans le chapitre précédent. Les arbres sont un cas particulier de graphe. Dans un graphe, les sommets peuvent être reliés à autant de nœuds que nécessaire et il peut comporter des boucles: en partant de certains nœud on est capable de revenir sur le nœud de départ. Les arêtes peuvent avoir une valeur numérique, un poids, on parle alors d'arbre « valué ». Selon la nature du graphe, les arêtes peuvent avoir un sens et des poids différents selon le sens, on parlera alors d'arcs.

Plus formellement, on dira qu'un graphe G est un couple G = (V, E) avec V un ensemble de sommets et E un ensemble d'arêtes.

Un exemple de graphe est le plan d'une ville dont les sommets sont les carrefours et les arcs sont les rues (avec leur sens de circulation).

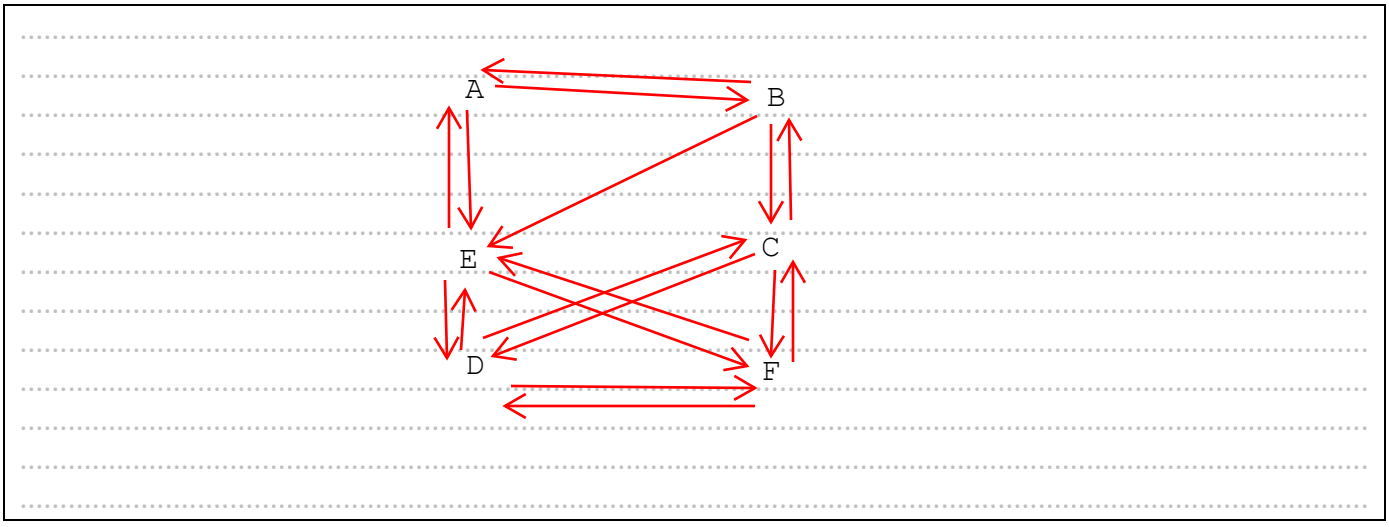
Un autre exemple est le plan d'un métro dont les sommets sont les stations et les arcs sont les liens entre stations directement reliées. Il y a bien d'autres exemples

Les graphes interviennent dès qu'une relation binaire (les arcs) existe dans un ensemble d'éléments (les sommets).

A Faire Vous-même I

Construisez un graphe de réseau social à partir des informations suivantes :

- A est ami avec B et E
- B est ami avec A, C et E
- C est ami avec B, F et D
- D est ami avec C, F et E
- E est ami avec A, D et F
- F est ami avec C, D et E



D'autres utilisations possibles : les logiciels de cartographie

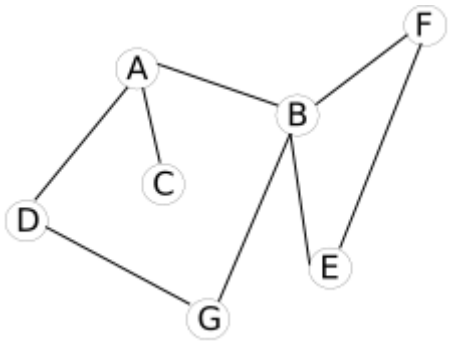
Ces logiciels de cartographie permettent, connaissant votre position grâce à un récepteur GPS, d'indiquer la route à suivre pour se rendre à un endroit B. Comment modéliser l'ensemble des lieux et des routes ? Simplement à l'aide d'un graphe ! Chaque lieu est un sommet et les routes qui relient les lieux entre eux sont des arêtes.

Soit les lieux suivants : A, B, C, D, E, F et G.

Les différents lieux sont reliés par les routes suivantes :

- il existe une route entre A et C
- il existe une route entre A et B
- il existe une route entre A et D
- il existe une route entre B et F
- il existe une route entre B et E
- il existe une route entre B et G
- il existe une route entre D et G
- il existe une route entre E et F

Ici aussi, la représentation sous forme de graphe s'impose :



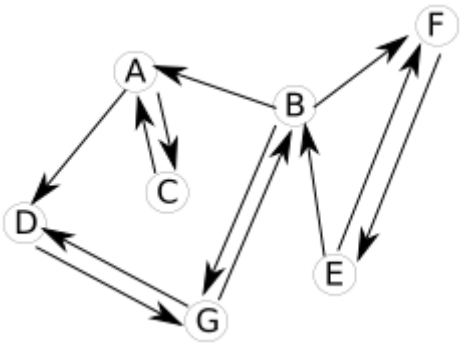
Auteur : David Roche

Problème : avec cette représentation du réseau routier sous forme de graphe, il est impossible de tenir compte des routes en sens unique (par exemple il est possible d'aller de A vers D mais pas de D vers A)

Voici donc de nouvelles contraintes :

- il existe une route entre A et C (double sens)
- il existe une route entre A et B (sens unique B->A)
- il existe une route entre A et D (sens unique A->D)
- il existe une route entre B et F (sens unique B->F)
- il existe une route entre B et E (sens unique E->B)
- il existe une route entre B et G (double sens)
- il existe une route entre D et G (double sens)
- il existe une route entre E et F (double)

Pour tenir compte de ces nouvelles contraintes, on utilisera un graphe orienté :

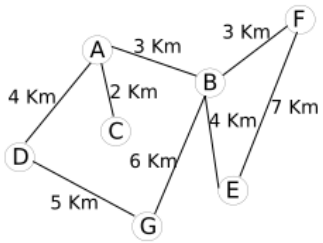


Auteur : David Roche

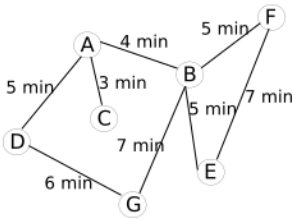
Un graphe orienté G = (S, A) possède un ensemble S de sommets et un ensemble A d'arcs. Un arc « a » relie un sommet s à un sommet « t »; « a » est l'origine de l'arc, « t » est son extrémité.

Il peut être intéressant d'associer aux arrêtes ou aux arcs des valeurs, on parle alors de graphes pondérés. Si nous revenons à notre "graphe cartographie", il est possible d'associer à chaque arête :

- la distance en Km entre les 2 lieux, graphe A :



- la durée du trajet entre 2 points, graphe B :



En fonction du choix fait par le conducteur (trajet le plus court "en distance" ou trajet le plus court "en temps"), l'algorithme permettant de déterminer le "chemin le plus court entre 2 points" travaillera sur le graphe A ou sur le graphe B. À noter que ce dernier peut évoluer au cours du temps en fonction du trafic routier : une application comme « Waze » utilise les données en provenance des utilisateurs de l'application afin de mettre à jour en temps réel leur "graphe pondéré (minutes) cartographie".

- Une chaine est une suite d'arêtes (ou d'arcs) consécutives dans un graphe, un peu comme si on se promenait sur le graphe. On la désigne par les lettres des sommets qu'elle comporte.
- Un cycle est une chaine qui commence et se termine au même sommet.

4.2 Implémentation

Il existe deux méthodes permettant d'implémenter un graphe : les matrices d'adjacences et les listes d'adjacences.

4.2.1 Les matrices d'adjacences

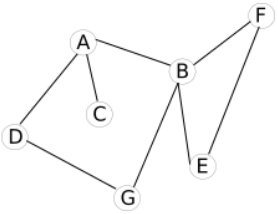
Une matrice n × m est un tableau de nombres à n lignes et m colonnes :

$$M = \begin{pmatrix} 2 & 3 & 2 & 10 \\ 5 & 0 & 8 & 8 \\ 9 & 2 & 7 & 12 \\ 4 & 11 & 14 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

n et m sont les dimensions de la matrice.

La matrice M ci-dessus est constituée de 5 lignes et 4 colonnes.
Une matrice carrée d'ordre n est une matrice de dimension n × n, autrement dit une matrice à n lignes et n colonnes.
Les matrices d'adjacences sont des matrices carrées.

Reprenons l'exemple du "graphe cartographie" :

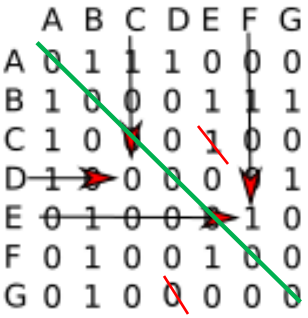


La matrice d'adjacence de ce graphe est :

0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	0	0	0	0	0
1	0	0	0	0	0	1
0	1	0	0	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0

Comment cette matrice a-t-elle été construite ?
A chaque ligne correspond un sommet du graphe et qu'à chaque colonne correspond aussi un sommet du graphe. À chaque intersection ligne i-colonne j (ligne i correspond au sommet i et colonne j correspond au sommet j), on place un 1 s'il existe une arête entre le sommet i et le sommet j, et un zéro (0) s'il n'existe pas d'arête entre le sommet i et le sommet j.

- Par conséquent, la première diagonale de la matrice est remplie de zéros.
- Il existe une arête entre le sommet E et le sommet F, nous avons donc placé un 1 à l'intersection de la ligne E et de la colonne F (il en est de même à l'intersection de la ligne F et de la colonne E)
- Il n'existe pas d'arête entre le sommet D et le sommet C, nous avons donc placé un 0 à l'intersection de la ligne D et de la colonne C (il en est de même à l'intersection de la ligne C et de la colonne D)

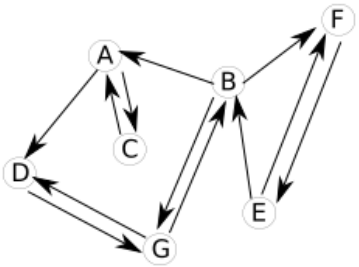


A Faire Vous-même II

Malheureusement, des erreurs ont été commises lors de la réalisation de la matrice. Vérifiez-la et corrigez les erreurs.

```
  ABCDEFG
A 0111000
B 1000111
C 1000000
D 1000001
E 0100010
F 0100100
G 0101000
```

Il est aussi possible d'établir une matrice d'adjacence pour un graphe orienté. Le principe reste le même : si le sommet i (ligne) est lié au sommet j (colonne), nous avons un 1 à l'intersection (0 dans le cas contraire).



Cela nous donne la matrice suivante :

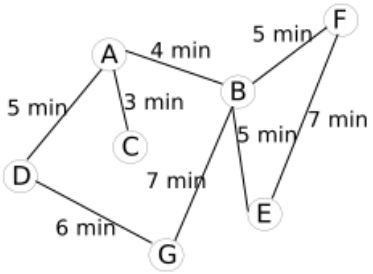
0	0	1	1	0	0	0
1	0	0	0	0	1	1
1	0	0	0	0	0	0
0	0	0	0	0	1	0
0	1	0	0	0	0	1
0	0	0	0	1	0	0
0	1	0	1	0	0	0

A Faire Vous-même III

Décidément, le réalisateur n'est pas attentif. Cette fois, il a interverti deux colonnes. Vérifiez la matrice et corrigez les erreurs. Vous prendrez soin de nommer les lignes et les colonnes afin de rétablir la matrice correspondant au graphe orienté ci-dessus.

	ABCDEFG
A	0011000
B	1000011
C	1000000
D	0000001
E	0100010
F	0000100
G	0101000

Il est également possible d'utiliser une matrice d'adjacence pour implémenter un graphe pondéré : on remplace les 1 (un) par les valeurs liées à chaque arc.



Cela nous donne la matrice suivante :

	A	B	C	D	E	F	G
A	0	4	3	5	0	0	0
B	4	0	0	0	5	5	7
C	3	0	0	0	0	0	0
D	5	0	0	0	0	0	6
E	0	5	0	0	0	7	5
F	0	7	0	0	7	0	0
G	0	7	0	6	0	0	0

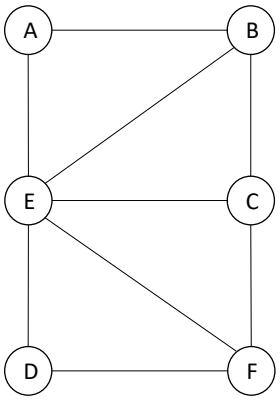
A Faire Vous-même IV

Vérifiez la matrice et corrigez les erreurs. Vous prendrez soin de nommer les lignes et les colonnes afin de rétablir la matrice correspondant au graphe pondéré ci-dessus.

```
   ABCDEFG
A 0435000
B 4000557
C 3000000
D 5000006
E 0500070
F 0500700
G 0706000
```

A Faire Vous-même V

Etablissez la matrice d'adjacence du réseau social G5 ci-contre :




```
  ABCDEF
A 010010
B 101010
C 010011
D 000011
E 111101
F 001110
```

En python, il est relativement aisé de manipuler des matrices d'adjacence. En effet, pour ce faire, nous utilisons des tableaux. En classe de première, nous avons appris à utiliser des tableaux avec des listes bidimensionnelles.

Rappel : les listes bidimensionnelles (listes de listes)

Il s'agit de tableaux à deux entrées. L'allocation prend la forme suivante :

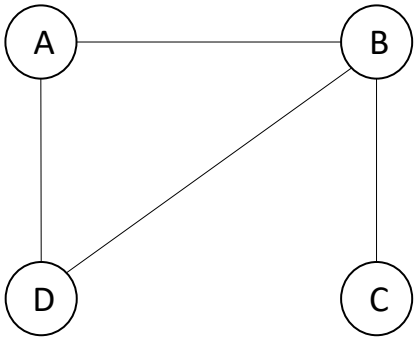
[[e for j in range(0,e1)] for i in range(0,e2)],

pour allouer une zone mémoire qui accueille une liste de i lignes et de j colonnes.

De même que pour les listes à une dimension, l'allocation et l'affectation peuvent tenir dans une seule instruction (compréhension de liste).

A Faire Vous-même VI

Soit le graphe G6 suivant :



a. Etablir sa matrice d'adjacence m6

```
  ABCD
A 0101
B 1011
C 0100
D 1100
```

b. Le programme suivant initialise la matrice « m6 » avec des « zéros » (0) :

```
# Libellé : matrice_init
# initialisation matrice i lignes j colonnes m[i][j]
# Cas particulier de la matrice carrée i = j

# initialisation d'une matrice carrée (4,4) avec la valeur 0
m6 = [[0 for j in range(0,4)] for i in range(0,4)]

# impression écran
print(m6)
```

Ecrivez ce programme dans un éditeur, sauvegardez-le avec le libellé « matrice_init.py » puis exécutez le. L'impression « écran » de « m6 » est celle d'une liste de listes.
Afin d'obtenir une impression écran de type matrice :

```
|  0  ||  0  ||  0  ||  0  |
|  0  ||  0  ||  0  ||  0  |
|  0  ||  0  ||  0  ||  0  |
|  0  ||  0  ||  0  ||  0  |
```

Modifiez le programme « matrice_init.py » et sauvegardez-le avec le libellé « matrice_init_affichage.py »

c. Initialisation de la matrice « m6 ». Modifiez le programme « matrice_init_affichage.py » et sauvegardez-le avec le libellé « matrice_init_affichage_m6.py » afin d'obtenir le résultat suivant :

```
   |  A  ||  B  ||  C  ||  D  |
A  |  0  ||  0  ||  0  ||  0  |
B  |  0  ||  0  ||  0  ||  0  |
C  |  0  ||  0  ||  0  ||  0  |
D  |  0  ||  0  ||  0  ||  0  |
```

d. Modifiez le programme « matrice_init_affichage_m6.py » et sauvegardez-le avec le libellé « m6_adj_graphe_G6_aff_1.py » afin d'obtenir le résultat suivant :

```
   |  A  ||  B  ||  C  ||  D  |
A  |  0  ||  1  ||  0  ||  1  |
B  |  1  ||  0  ||  1  ||  1  |
C  |  0  ||  1  ||  0  ||  0  |
D  |  1  ||  1  ||  0  ||  0  |
```

e. Afin de vérifier plus aisément la conformité d'une matrice d'adjacence avec le graphe dont elle est issue, il est plus commode de pouvoir identifier une valeur à l'aide de ses coordonnées dans la matrice. Par exemple, dans notre matrice d'adjacence m6, l'élément situé au croisement de la 2ème ligne et de la 4ème colonne a pour coordonnées (2, 4) et pour valeur « 1 ». Nous pouvons l'identifier avec l'expression « m6(2, 4) »

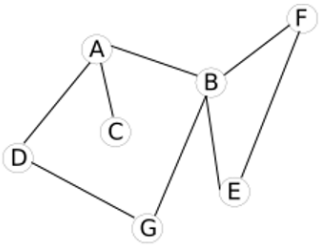
Modifiez le programme « m6_adj_graphe_G6_aff_1.py » et sauvegardez-le avec le libellé « m6_adj_graphe_G6_aff_2.py » afin d'obtenir le résultat suivant :

```
  |          A          ||          B          ||          C          ||          D          |
A | m6( 1 , 1 )=  0  || m6( 1 , 2 )=  1  || m6( 1 , 3 )=  0  || m6( 1 , 4 )=  1  |
B | m6( 2 , 1 )=  1  || m6( 2 , 2 )=  0  || m6( 2 , 3 )=  1  || m6( 2 , 4 )=  1  |
C | m6( 3 , 1 )=  0  || m6( 3 , 2 )=  1  || m6( 3 , 3 )=  0  || m6( 3 , 4 )=  0  |
D | m6( 4 , 1 )=  1  || m6( 4 , 2 )=  1  || m6( 4 , 3 )=  0  || m6( 4 , 4 )=  0  |
```

4.2.2 Les listes d'adjacence

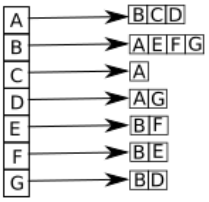
Afin d'implémenter un graphe, on commence par définir la liste des sommets du graphe. À chaque élément de cette liste, on associe une autre liste qui contient les sommets liés à cet élément :

Reprenons l'exemple du "graphe cartographie" du « afvm I » :



Auteur : David Roche

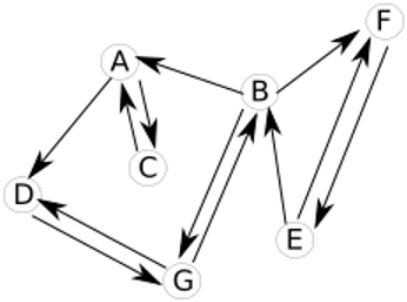
Voici la liste d'adjacence de ce graphe :



Pour les graphes orientés, il est nécessaire de définir 2 listes : la liste des successeurs et la liste des prédécesseurs. Soit un arc allant d'un sommet A vers un sommet B (flèche de A vers B). On dira que B est un successeur de A et que A est un prédécesseur de B.

A Faire Vous-même VII

Reprenons l'exemple du "graphe orienté cartographie" du « afvm I » :



Auteur : David Roche

En vous aidant des indications précédentes, établissez :

- la liste d'adjacence des successeurs,
- la liste d'adjacence des prédécesseurs,

du graphe orienté cartographie ci-dessus

```
A -> CD
B -> AFG
C -> A
D -> G
E -> BF
F -> E
```

```
A <- BC
B <- EG
C <- A
D <- AG
E <- F
F <- BE
```

5 Les dictionnaires

5.1 Qu'est-ce que c'est ?

Un dictionnaire (aussi appelé table d'association) est un type de donnée associant à un ensemble de clefs, un ensemble correspondant de valeurs. Chaque clef est associée à une seule valeur (au plus).

Du point de vue du programmeur, le dictionnaire peut être vu comme une généralisation du tableau : alors que le tableau traditionnel associe des entiers consécutifs à des valeurs, le tableau associatif associe des clefs d'un type arbitraire à des valeurs d'un autre type. *(Wikipédia)*

5.2 A quoi ça sert ?

Le dictionnaire sert essentiellement à optimiser certaines opérations sur les données.

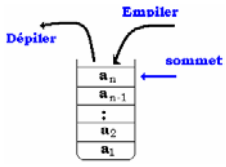
Concrètement, les dictionnaires sont des structures de données dont les temps d'accès, d'insertion, d'effacement et le test d'appartenance sont indépendants du nombre d'éléments.

En python, les dictionnaires sont des objets mutables que l'on peut modifier en place avec une excellente efficacité mémoire.

A retenir : Un dictionnaire est une collection de couples « 'clé' : valeur ». La clé peut être n'importe quel objet immuable.

5.3 Exemples

L'annuaire téléphonique peut être vu comme un dictionnaire, dans lequel on associe des noms (les clefs) à des numéros de téléphone (les valeurs).
Un autre exemple est celui du dictionnaire (d'une langue), dans lequel on associe des mots (les clefs) à des définitions (les valeurs).
Enfin, afin de faire le lien avec la séquence « Bases de données », une table de base de données peut également être vue comme un dictionnaire dans lequel on associe des ensembles d'attributs (les clefs) à des enregistrements (les valeurs). Une base de données entière peut être vue comme un ensemble de tableaux associatifs liés par les contraintes que constituent les règles mathématiques (algèbre relationnel) d'Edgar Codd.



5.4 Implémentation

En python, la SDD « Dictionnaire » est utilisable au travers du type buit-in « dict ».

```
>>> help(dict)
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
```

Les méthodes accessibles sont :

```
clear(...)
    D.clear() -> None.  Remove all items from D.

copy(...)
    D.copy() -> a shallow copy of D

fromkeys(iterable, value=None, /) from builtins.type
    Returns a new dict with keys from iterable and values equal to value.

get(...)
    D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.

items(...)
    D.items() -> a set-like object providing a view on D's items

keys(...)
    D.keys() -> a set-like object providing a view on D's keys

pop(...)
    D.pop(k[,d]) -> v, remove specified key and return the corresponding value.
    If key is not found, d is returned if given, otherwise KeyError is raised

popitem(...)
    D.popitem() -> (k, v), remove and return some (key, value) pair as a
    2-tuple; but raise KeyError if D is empty.

setdefault(...)
    D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D

update(...)
    D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does:  for k in E: D[k] = E[k]
    If E is present and lacks a .keys() method, then does:  for k, v in E: D[k] = v
    In either case, this is followed by: for k in F:  D[k] = F[k]

values(...)
    D.values() -> an object providing a view on D's values
```

5.4.1 Création, mise à jour et effacement

Nous allons maintenant manipuler les dictionnaires au travers de quelques opérations élémentaires.

a. Création

Pour créer un dictionnaire, on utilise la notation accolades :

```
>>> age = {'agnès':23,'alexis':17,'gabrielle':37}
>>> type(age)
<class 'dict'>
>>>
```

Ici, nous avons créé un dictionnaire qui possède trois clés 'agnès', 'alexis' et 'gabrielle', et trois valeurs 23, 17 et 37. Vous pouvez voir les dictionnaires comme une collection de couples clé - valeur. Votre dictionnaire va stocker cette collection qui n'est pas ordonnée. En effet, il n'y a pas d'ordre dans un dictionnaire

b. Accès aux éléments

Vous pouvez accéder aux différents éléments en nommant la clé :

```
>>> age['agnès']
23
```

c. Construction d'un dictionnaire à partir d'une liste

Il existe une autre manière de construire un dictionnaire, en partant d'une liste :

```
>>> age = [('agnès',23,),('alexis',17,),('gabrielle',37,)]
>>> type(age)
<class 'list'>
>>> Age = dict(age)
>>> Age
{'agnès': 23, 'alexis': 17, 'gabrielle': 37}
```

La fonction built-in « dict() » crée un dictionnaire à partir de ces couples (clé – valeur) où vous aurez: ' agnès ' qui correspond à la valeur 23, 'alexis' à la valeur 17 et 'gabrielle' à la valeur 37.

L'accès aux valeurs se fait de la même manière :

```
>>> Age['gabrielle']
37
```

d. Modification

Testons le caractère « mutable » d'un dictionnaire en changeant la valeur d'une clé. Par exemple, donnons la majorité à 'alexis':

```
>>> Age['alexis'] = 18
>>> Age
{'agnès': 23, 'alexis': 18, 'gabrielle': 37}
>>>
```

e. Effacement

Vous pouvez également effacer un couple clé – valeur en utilisant l'instruction « del » :

```
>>> del Age['agnès']
>>> Age
{'alexis': 18, 'gabrielle': 37}
>>>
```

5.4.2 Méthodes « built-in Python »

Les dictionnaires possèdent des opérations qui sont très proches des séquences lorsque cette opération a un sens. Par exemple, combien est-ce que j'ai de clés dans mon dictionnaire ? Combien de couples clé - valeur ? Pour le savoir, nous utilisons la fonction built-in « len » :

```
>>> len(Age)
2
>>>
```

Il en va de même pour le test d'appartenance sur un dictionnaire :

```
>>> 'gabrielle' in Age
True
>>> 'agnès' in Age
False
```

a. Accès aux clés, aux valeurs, ou aux couples (clé – valeur)

En python un couple (clé – valeur) se nomme un « item » d'un dictionnaire. L'accès aux clés se fait avec la méthode keys() :

```
>>> clefs = Age.keys()
>>> clefs
dict_keys(['alexis', 'gabrielle'])
>>> type(clefs)
<class 'dict_keys'>
>>>
```

Puis nous accédons aux valeurs avec « values() » qui retourne un objet qui contient les valeurs :

```
>>> Age.values()
dict_values([18, 37])
```

Enfin nous accédons aux couples (clé - valeur) qu'on appelle les items avec la méthode sur le dictionnaire items() :

```
>>> Age.items()
dict_items([('alexis', 18), ('gabrielle', 37)])
```

Ces méthodes keys(), values() et items() retournent un objet qu'on appelle une « vue ». Qu'est-ce qu'une vue en Python ? C'est un objet sur lequel on peut itérer, (avec une boucle for), faire un test d'appartenance (avec in). La vue est mise à jour en même temps que le dictionnaire. Reprenons notre variable « clefs »:

```
>>> clefs = Age.keys()
>>> clefs
dict_keys(['alexis', 'gabrielle'])
>>> Age['agnès'] = 20
>>> clefs
dict_keys(['alexis', 'gabrielle', 'agnès'])
```

Nous constatons que la vue « clefs » a automatiquement été mise à jour avec cette nouvelle clé 'agnès'.

Testons maintenant l'appartenance avec l'opérateur « in » :

```
>>> 'alexis' in clefs
True
>>> 'léo' in clefs
False
>>>
```

b. Parcours d'un dictionnaire

Une façon commode de le parcourir est d'itérer sur le dictionnaire en utilisant une boucle et la notation de « tuple unpacking » :

```
>>> for (clefs, valeurs,) in Age.items():
        print(f"couple ({clefs},{valeurs})")
        #ici on formate la sortie écran avec une f-string
couple (alexis,18)
couple (gabrielle,37)
couple (agnès,20)
>>>
```

Enfin, notez que l'itérateur « naturel » (sans préciser une vue) sur un dictionnaire est un itérateur directement sur les clés :

```
>>> for clé in Age:
        print(clé)
alexis
gabrielle
agnès
>>>
```

A retenir :

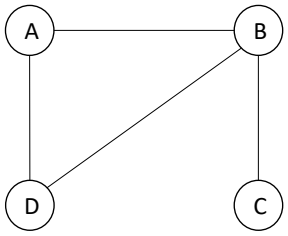
- Le type dictionnaire permet l'accès, l'insertion, la modification, et le test d'appartenance, indépendamment du nombre d'éléments.
- Le dictionnaire est une structure de données extrêmement souple, qui vous permet par exemple d'implémenter sans aucun effort un agenda ou un annuaire.

5.4.3 Exemple

Au cours du chapitre précédent, nous avons vu qu'on pouvait décrire un graphe en utilisant une liste d'adjacence. En python, il est possible de manipuler une liste d'adjacence en utilisant un dictionnaire.

Dans ce cas, il semble opportun d'associer des sommets (les clefs) à des ensembles de sommets (les valeurs).

Reprenons le graphe de réseau social G6 du « afvm VI » :



Nous avions établi les relations suivantes :

- A est ami avec B et D
- B est ami avec A, C et D
- C est ami avec B
- D est ami avec A et B

Par conséquent, nous en déduisons, L6, la liste d'adjacence de G6 :

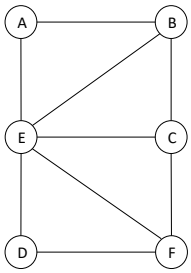
A ⟶ B, D
B ⟶ A, C, D
C ⟶ B
D ⟶ A, B

Le dictionnaire L6 correspondant est donc :

```
>>> L6
{'A': ('B', 'D'), 'B': ('A', 'C', 'D'), 'C': ('B',), 'D': ('A', 'B')}
```

A Faire Vous-même VII

Soit le graphe de réseau social G5 du « afvm V » :



a. Etablir liste5, la liste d'adjacence de G5

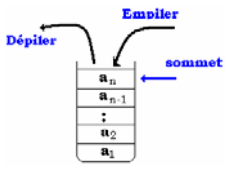
```
A -> BE
B -> ACE
C -> BEF
D -> EF
E -> ABCDF
F -> CDE
```

b. Initialiser L5, le dictionnaire du graphe G5

```
L5 = {'A':('B','E'),'B':('A','C','E'),'C':('B','E','F'),
      'D':('E','F'),'E':('A','B','C','D','F'),'F':('C','D','E')}
```

c. A l'aide d'une instruction Python, vérifiez la liste des amis de « E »

```
>>> L5['E']
('A', 'B', 'C', 'D', 'F')
```



Les Structures De Données

Cours & Activité Pratique

A Faire Vous-même VIII

Soit Ta, la table d'association qui associe à un mot sa définition :

1	Liste	Tableau à deux dimensions dans le jargon des Bdd	A
2	Graphe	structure de données basée sur le principe « Premier entré, premier sorti »	B
3	File	un couple (V, E) avec V un ensemble de sommets et E un ensemble d'arêtes.	C
4	Arbre	Mot ou groupes de mots utilisés pour décrire un contenu.	D
5	Serveur	Suite finie de règles et d'opérations élémentaires qui permet de résoudre une classe de problèmes	E
6	Table	structure de données permettant de regrouper des données de manière à pouvoir y accéder librement	F
7	Algorithme	Ordinateur dont les informations peuvent être consultées à distance par d'autres ordinateurs.	G
8	Balise	structure de données constituée de sommets reliés par des arêtes	H
9	Dictionnaire	une collection de couples « 'clé' : valeur »	I

a. Modifier Ta afin d'associer à chaque mot, une définition correcte

1F	.....
2C	.....
3B	.....
4H	.....
5G	.....
6A	.....
7E	.....
8D	.....
9I	.....

b. Construire le dictionnaire « mon_dico » à partir de Ta. Vous utiliserez deux méthodes distinctes

```
mon_dico={"Liste":"Structure de donneés permettant de regrouper des donnéés de maniere à pouvoir y accéder librement",
          "Graphe":"Un couple (V,E) avec V un ensemble de sommets et E un ensemble d'arêtes",
          "File":"Structure de données basées sur le principe : Premiers entré, premier sorti",
          "Arbre":"Structure de données constituée de sommets reliés par des arêtes",
          "Serveur":"Ordinateur dont les informations peuvent être consultées à distance par d'autres ordinateurs",
          "Table":"Tableau à deux dimensions dans le jargon des Bdd",
          "Algorithme":"Suite finie de règles et d'operations élémentaires qui permet de résoudre une classe de problèmes",
          "Balise":"Mot ou groupe de mots utilisés pour décrire un contenu",
          "Dictionnaire":"Une collection de couple('clé':valeur)"}

liste_dico=[("Liste","Structure de donneés permettant de regrouper des donnéés de maniere à pouvoir y accéder librement"),
            ("Graphe","Un couple (V,E) avec V un ensemble de sommets et E un ensemble d'arêtes"),
            ("File","Structure de données basées sur le principe : Premiers entré, premier sorti"),
            ("Arbre","Structure de données constituée de sommets reliés par des arêtes"),
            ("Serveur","Ordinateur dont les informations peuvent être consultées à distance par d'autres ordinateurs"),
            ("Table","Tableau à deux dimensions dans le jargon des Bdd"),
            ("Algorithme","Suite finie de règles et d'operations élémentaires qui permet de résoudre une classe de problèmes"),
            ("Balise","Mot ou groupe de mots utilisés pour décrire un contenu"),
            ("Dictionnaire","Une collection de couple('clé':valeur)")]

mon_dico_2 = dict(liste_dico)
```

c. Après avoir vérifié que l'objet « mon_dico » était bien de type « dictionnaire », afficher la définition des mots « Graphe » et « File »

```
>>> type(mon_dico)
<class 'dict'>
>>> mon_dico['Graphe']
"Un couple (V,E) avec V un ensemble de sommets et E un ensemble d'arêtes"
>>> mon_dico['File']
'Structure de données basées sur le principe : Premiers entré, premier sorti'
>>>
```

d. En utilisant un test d'appartenance, vérifiez que « mon_dico » contient le mot « Pile ». Si ce n'est pas le cas, ajoutez le mot « Pile » avec sa définition à « mon-dico ».

```
>>> 'Pile' in mon_dico
False
>>> mon_dico['Pile'] = "Structure de données fondée sur le pricipe : Dernier arrivé, premier sorti"
```

e. Quelle est la taille de « mon_dico » ? Vérifiez-le au moyen d'une instruction Python

```
Il y a 10 couples clé,valeur
>>> len(mon_dico)
10
```

f. En créant une vue, afficher la liste des mots de « mon_dico » sur une ligne comme ceci :

Table File Graphe Balise Algorithme Liste Serveur Arbre Dictionnaire Pile

```
>>> cle_dico = mon_dico.keys()
>>> for cle in cle_dico:
        print(f"{cle}  ",end='')
```

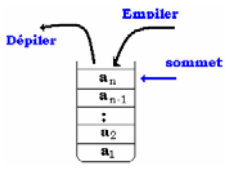
g. Enfin, afficher la liste des mots de « mon_dico » commençant par la lettre « A » accompagnés de leur définition comme ceci :

Algorithme : Suite finie de règles et d'opérations élémentaires qui permet de résoudre une classe de problèmes

Arbre : structure de données constituée de sommets reliés par des arêtes

```
>>> for cle in cle_dico:
        if cle[0] == 'A':
            print(f"{cle} : {mon dico[cle]}")
```

A Faire Vous-même VIII



Les Structures De Données

Cours & Activité Pratique

Nous allons tenter de répondre à la problématique d'un jeune entrepreneur qui se lance dans la vente en ligne d'articles de sport. Une de ses priorités est de gérer, au plus juste, son stock afin de pouvoir honorer ses commandes et limiter au maximum les frais de gestion de stock.

Ainsi, il expose sa problématique à un de ses amis. Celui-ci lui propose de gérer son stock d'article à l'aide d'une structure de données ayant les caractéristiques suivantes :

- Elle est constituée d'une collection d'enregistrements
- Chaque enregistrement possède la structure suivante :
  - Un code /* code article de type entier*/
  - Une dénomination /* nom du produit de type chaine de caractère*/
  - Un prix /* prix unitaire du produit de type réel */
  - Un stock /* stock disponible de type entier*/

Dans un premier temps, nous réaliserons l'initialisation et la saisie des enregistrements. Afin que l'exercice de saisie ne soit pas trop fastidieux, on se limitera à 10 enregistrements.

Ecrire un programme d'initialisation et de saisie de la structure de données. Dans une première approche, on pourra définir une liste de dictionnaires. Chaque dictionnaire stocke un enregistrement. Par exemple, pour trois articles de sport, le programme affichera le résultat suivant :

```
Veuillez entrer un code article :100
Veuillez entrer la dénomination de l'article :chaussures
Veuillez entrer le prix de l'article :59.99
Veuillez entrer le stock de l'article :3
Veuillez entrer un code article :210
Veuillez entrer la dénomination de l'article :raquette
Veuillez entrer le prix de l'article :39.59
Veuillez entrer le stock de l'article :6
Veuillez entrer un code article :650
Veuillez entrer la dénomination de l'article :VTT
Veuillez entrer le prix de l'article :899
Veuillez entrer le stock de l'article :2
```

Pour cette approche, la SDD possède la forme suivante :

```
[{'Code': 100, 'Dénomination': 'chaussures', 'Prix': 59.99, 'Stock': 3}
, {'Code': 210, 'Dénomination': 'raquette', 'Prix': 39.59, 'Stock': 6},
{'Code': 650, 'Dénomination': 'VTT', 'Prix': 899.0, 'Stock': 2}]
```

**** *Fin du Document Support* ****