NOTIONS FONDAMENTALES – 5 - TABLEAUX

RESUME DE COURS - EXERCICES

Version sept 2018

7 - Les tableaux

1. Tableau à une dimension

tab=[] // création d'un tableau vide

tab = [5, 1, 8, 3, 6] // création d'un tableau avec 5 éléments, 5 entiers en l'occurrence.

tab	5	1	8	3	6
	tab[0]	tab[1]	tab[2]	tab[3]	tab[4]

```
lire (tab[0])
tab[0] = tab[0]*2
afficher tab[0]
```

Manipuler tous les éléments d'un tableau : la boucle for, len(tab) ou tab.len

```
pour i de 0 à tab.len -1
    Afficher(tab[i])
fin pour
```

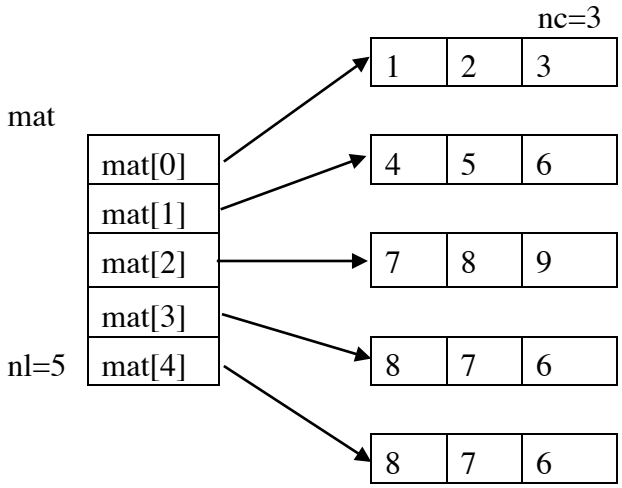
Tableau à taille fixe et tableau à taille variable

Insertion à la fin : tab.apped(valeur), suppression à la fin : valeur =tab.pop()

2. Tableau à 2 dimensions : matrice – mat de taille nl, nc

mat nc=3

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



mat=[] // création d'une matrice vide

mat = [[1,2,3], [4,5,6], [7,8,9], [8,7,6], [5,4,3]] // création d'une matrice de 5 lignes avec un tableau de 3 éléments par ligne.

```
lire ( mat[0][0] )
tab[0] = mat[0][0]*2
afficher( mat[0][0] )
```

Manipuler tous les éléments d'une matrice: 2 boucle for imbriquées

On peut écrire **len(mat)** ou **mat.len** pour récupérer les éléments d'un tableau.

```
pour i de 0 à mat.len -1
    pour j de 0 à len( mat([i] )-1
        Afficher(mat[i][j])
    Finpour
    Afficher (passage à la ligne)
fin pour
```

Recherche, recherche dichotomique dans un tableau trié

Tri à bulles, Tri par extraction, Tri par insertion

Python de base

Tableau

```
# création d'un tableau de 5 éléments
tab=[5, 1, 8, 3, 6]
print(tab)
for i in range(0, len(tab)):
    print(tab[i])

#il faut un saut de ligne à la fin de bloc car on n'est pas dans une fonction
tab.append(10)
print(tab)
valeur=tab.pop()
print(tab)
```

Matrice

```
# création d'une matrice de 5 lignes avec un tableau de 3 éléments par ligne
mat=[[1,2,3], [4,5,6], [7,8,9], [8,7,6], [5,4,3]]

def printMat(mat): # function d'affichage d'une matrice
    nc=len(mat)
    for i in range(0, nc):
        nl=len(mat[i])
        for j in range(0, nl):
            print("%3d" %(mat[i][j]), end='')
            if(j!=nl-1): print(', ', end='')
        print();
```

```
printMat(mat)
mat[0].append(999)
printMat(mat)
mat[1].pop()
printMat(mat)
```

EXERCICES – SERIE 5 – TABLEAUX

Méthode de base d'analyse algorithmique

La méthode de base pour écrire un algorithme suit les 4 étapes suivantes :

1. Comprendre le problème : bien lire le sujet et bien comprendre ce qu'il y a à faire.
2. Lister ce dont on a besoin pour résoudre le problème (les données) et ce qu'on va produire (les résultats) : préciser les Entrées et les Sorties.
3. Trouver un principe de résolution : se donner les grandes lignes, en français, de la méthode de résolution. Eventuellement, se donner des procédures ou des fonctions (des actions générales).
4. Ecrire l'algorithme en détail.

Complexité

Pour tous les exercices, on donnera la complexité temporelle théorique de l'algorithme proposé.

1 - Tableaux à une dimension - base

Exercice 1 : somme des éléments d'un tableau

Ecrire une fonction qui calcule la somme des éléments d'un tableau.

Exercice 2 : valeur la plus grande d'un tableau (max)

Ecrire une fonction qui détermine la valeur la plus grande d'un tableau. On renverra le maximum puis la position du maximum dans le tableau.

Exercice 3 : recherche d'un élément dans un tableau non trié

Écrire une procédure (ou une fonction) qui recherche une valeur dans un tableau non trié sans doublon.

2 - Tableaux à une dimension - avancé

Exercice 1 : recherche d'un élément dans un tableau trié

Écrire une procédure (ou une fonction) qui recherche une valeur dans un tableau trié sans doublon. On utilisera la méthode de recherche dichotomique.

Exercice 2 : inversion d'un tableau

Ecrire une fonction qui inverse un tableau : le premier élément est permuté avec le dernier, le deuxième avec l'avant dernier, etc.

Exercice 3 : tri à bulles

Ecrire l'algorithme de tri d'un tableau par la méthode du tri à bulles.

Le principe est d'inverser les couples de valeurs successives du tableau en fonction de l'ordre du tri, en parcourant le tableau d'un bout à l'autre et en répétant l'opération autant de fois que nécessaire.

Exemple avec le tableau : 6 10 5 8 12 4

On teste 6 et 10 et on ne fait rien

On teste 10 et 5 et on inverse : 6 5 10 8 12 4

On teste 10 et 8 et on inverse : 6 5 8 10 12 4

On teste 10 et 12 et on ne fait rien

On teste 12 et 4 et on inverse : 6 5 8 10 4 12

Le plus grand (12) est tout en bas.

Et on recommence :

On teste 6 et 5 et on inverse : 5 6 10 8 12 4

On teste 6 et 10 et on ne fait rien

Etc.

Quelle est la complexité de l'algorithme ?

Exercice 4 : décalage ver le bas

Ecrire une fonction qui décale les éléments d'un tableau vers le bas (le 5ème passe en 4ème par exemple et le premier passe en dernier, le Nième.

Exercice 5 : décalage vers le haut

Ecrire une fonction qui décale les éléments d'un tableau vers le haut (le 4ème passe en 5ème par exemple et le le dernier, le Nième, passe en premier.

Exercice 6 : tri par extraction

Ecrire une fonction qui tri un tableau en utilisant la méthode de tri par extraction. Le principe de ce tri est d'aller chercher le plus petit élément du tableau pour le mettre en premier, puis de recommencer l'opération en partant du second élément du tableau, puis du troisième, etc.

Exercice 7 : suppression d'un élément

Écrire une procédure (ou une fonction) qui supprime une valeur dans un tableau trié sans doublon.

Exercice 8 : ajout d'un élément

Écrire une procédure (ou une fonction) qui ajoute une valeur dans un tableau trié sans doublon.

Exercice 9 : suppression des doublons

Ecrire une fonction qui élimine les doublons d'un tableau d'entiers. On considérera d'abord le tableau comme étant trié puis on traitera le cas général.

Matrices

Exercice 1

Ecrire une fonction d'affichage des éléments d'une matrice. Faites-en sorte que l'affichage soit lisible.

Exercice 2

Ecrire une fonction qui calcule la moyenne des sommes des lignes et la moyenne des sommes des colonnes d'une matrice.

Exercice 3

Ecrire une fonction qui calcule la moyenne par ligne et par colonne dans une matrice.

Exercice 4

Ecrire une fonction qui multiplie deux matrices.

Avec A matrice n lignes, m colonnes. B matrice m lignes, p colonnes. C matrice n lignes, p colonnes.

$C_{i,j}=somme(k=0, m)\ A_{i,k} * B_{k,j}$.

Exercice 5

Ecrire une procédure qui permet de trier une matrice selon une colonne donnée (utiliser la méthode du tri à bulles).

Exercice 6

Ecrire une procédure qui permet de trier une matrice selon une ligne donnée (utiliser la méthode du tri à bulles).

Problèmes

Exercice 1 : Tours de Hanoi

➢ *Description du jeu*

Le jeu est constitué d'une plaquette de bois où sont plantées trois tiges : les tours.

Sur ces tiges sont enfilées des disques de diamètres tous différents.

Les seules règles du jeu sont que l'on ne peut déplacer qu'un seul disque à la fois, et qu'il est interdit de poser un disque sur un disque plus petit.

Au début tous les disques sont sur la tige de gauche, et à la fin sur celle de droite.

➢ *Méthode de résolution itérative*

Déplacer le 1 à gauche (« faire le tour » et revenir sur la tour de droite si on est déjà complètement à gauche).

Déplacer celui des deux qui n'est pas 1 et qu'on peut déplacer.

Répéter les deux opérations jusqu'au déplacement complet de la tour.

Subtilité : Si le nombre total de disques est impair, il faut déplacer le 1 à gauche. Mais si le nombre total de disques est pair, il faut déplacer le 1 à droite !

➢ *Premier objectif*

Ecrire un programme qui affiche les déplacements dans des tours de Hanoï.

A chaque coup, on affichera le numéro du coup pour arriver au nombre de coups joués pour arriver au but.

Dans un premier temps on fera un affichage très simplifié.

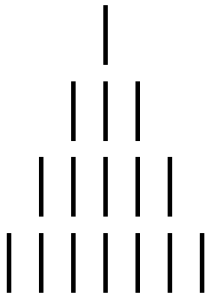
➢ *Méthode*

1) Bien comprendre le fonctionnement du jeu et le principe de résolution : faites-le « tourner » à la main. On peut tester le jeu en ligne, par exemple ici.
2) Quelles données envisagez-vous d'utiliser pour traiter le problème ?
3) Quel algorithme général pouvez-vous écrire qui traduise le principe de résolution et l'objectif, et qui utilise vos données ? L'algorithme général doit utiliser des fonctions
4) Une fois l'algorithme général conçu, il faut écrire le détail de chaque fonction, éventuellement en créant d'autres fonctions.

Exercice 2 : Jeu des allumettes ou jeu de Marienbad

- ***Description du jeu : test ici***

  On a 4 lignes d'allumettes avec 1, 3, 5 et 7 allumettes.



  Le but du jeu est de ne pas ramasser la dernière allumette : qui prend la dernière perd.

  Le jeu se joue à 2. A son tour, chaque joueur peut prendre autant d'allumettes qu'il veut mais sur une seule ligne.

- ***Comment gagner à tous les coups ?***

  D'abord, il ne faut pas commencer.

  Ensuite, il faut laisser le jeu dans une configuration particulière expliquée maintenant.

  Chaque ligne contient un nombre d'allumettes : 1, 3, 5 et 7 au départ. Il faut traduire ce nombre en binaire.

  Situation de départ :

Nombre d'allumettes	Nombre d'allumettes en binaire		
1	0	0	1
3	0	1	1
5	1	0	1
7	1	1	1
Total en base 10 de chaque colonne binaire :	2	2	4

  Le joueur 1 qui commence est dans une configuration telle que le total en base 10 de chaque colonne binaire est un nombre pair. C'est cette configuration qui fait que le joueur va perdre, si l'adversaire ne fait pas d'erreur.

  Quand le joueur 1 va joueur, quoi qu'il fasse, il laissera un ou plusieurs totaux impairs.

  Le joueur 2 devra jouer de telle sorte que chaque total soit à nouveau un nombre pair.

- ***Vérifier que vous avez compris comment gagner***

  Essayer de jouer en appliquant la méthode pour gagner à tous les coups pour vérifier que vous avez compris la méthode.

- ***Coder un programme qui permette de jouer contre l'ordinateur***

  Pour ça, vous devez d'abord réfléchir aux grandes lignes du programme : les grandes fonctions.

  Vous devez aussi réfléchir aux structures de données manipulées.

  Ensuite, pour chaque grande fonction identifier dans votre algorithme général, essayer d'écrire l'algorithme en le divisant en fonctions. C'est particulièrement vrai pour l'algorithme qui s'occupera de faire jouer l'ordinateur.

Exercice 3 : Bucket Sort

Principe du « bucket sort » :

Le « bucket sort » ordonne un tableau donné de N éléments. Il a besoin pour cela de tableaux auxiliaires. Le principe est de répartir les valeurs dans plusieurs tableaux puis de fusionner en commençant par les unités.

Voyons sur un exemple le fonctionnement du « bucket sort » :

- ***Soient les 11 nombres suivants à trier :***

112	97	156	253	744	514	266	634	800	1	333

- ***étape 1 : on range les nombres dans 10 sous tableaux (de T0 à T9) en fonction de leur unité :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
800	1	112	253	744		156	97		
			333	514		266			
				634					

Ensuite on remplace ces sous tableaux dans le tableau principal :

800	1	112	253	333	744	514	634	156	266	97

- ***étape 2 : ce tableau est à nouveau réparti mais cette fois-ci par rapport aux dizaines :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
800	112		333	744	253	266			97
1	514		634		156				

Ensuite on fusionne de nouveau et on obtient :

800	1	112	514	333	634	744	253	156	266	97

- ***Dernière étape : on répartit par rapport aux centaines :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
1	112	253	333		514	634	744	800	
97	156	266							

Ensuite on fusionne de nouveau et on obtient :

1	97	112	156	253	266	333	514	634	744	800

- ***Test des tris***

Créer un tableau de 2000 nombres < 1000.

Code python pour avoir un entier aléatoire entre 0 et 999

```
from random import *
randint(0, 999)
```

Mesurer le temps d'exécution avec un Bucket Sort et avec un tri à bulles.

Pour mesurer le temps d'exécution, deux méthodes :

Compter le nombres d'itérations dans les boucles

Mesurer le temps au début et à la fin (ça ne donne pas forcément, en première approche, une durée avec une unité connue, mais ça donne une base de comparaison).

```
import time
tmps1=time.clock()
# code
tmps2=time.clock()
print (tmps2-tmps1)
```