NSI 1ère - Algorithmique - 4 - tableaux

QK

Tableaux

Tableaux

- Un tableau (*array* en anglais) est une SDD qui contient un ensemble d'éléments auquel on accède avec un numéro d'indice.
- Le temps d'accès à un élément par son indice est *constant*
- Les éléments sont contigus dans l'espace mémoire. Avec l'indice on sait à combien de cases mémoire se trouve l'élément en partant du début du tableau
- Souvent désignés par une majuscule : T est un tableau, $T[i]$ est son élément d'indice i

Avantages / Inconvénients

- **Avantages** : accès direct au ième élément
- **Inconvénients** : les opérations d'insertion et de suppression sont impossibles.
  - Il faut créer un nouveau tableau, de taille plus grande ou plus petite (selon l'opération). Il faut alors copier tous les éléments du tableau original dans le nouveau tableau. Cela fait beaucoup d'opérations.

Tableaux en Python : des listes !

Dans Python les tableaux sont des objets *listes*.

On peut les construire de plusieurs manières :

- `[]` est une liste vide
- `l = [""abc", "def", "ghi""]` et on accède avec `l[1] "def"`
- Par compréhension : `[2 * x + 1 for x in range(4)] [1, 3, 5, 7]`

Listes par compréhension, suite

On peut itérer dans une chaîne de caractères donc facilement la découper

```
>>> lettres = "abcdefg"
>>> [ i for i in lettres ]
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

- Il y a bien d'autres façons de faire.

La grande différence entre les tableaux et les listes ?

Les listes sont **mutables** :

```
>>> l = ['abc', 'def', 'ghi']
>>> l[1]
'def'
>>> l[1] = 'xyz' # modifier un élément
>>> l
['abc', 'xyz', 'ghi']
>>> l.insert(2, "mno") # insérer
>>> l
['abc', 'xyz', 'mno', 'ghi']
>>> l.remove('abc') # supprimer l'élément 'abc'
>>> l
['xyz', 'mno', 'ghi']
```

Affecter : le même objet !

Dans Python quand on affecte un objet à un autre, ils sont identiques !

```
>>> l = ['abc', 'def', 'ghi']
>>> m = l # m et l : les mêmes objets
>>> m[1] = "xyz" ; l[0] = "pqr" # l et m modifiés
>>> l, m
(['pqr', 'xyz', 'ghi'], ['pqr', 'xyz', 'ghi'])
```

Copier : une copie de l'objet

On contourne cette difficulté avec un "slice"

```
>>> l = ['abc', 'def', 'ghi']
>>> l[1:] # une copie de l à partir de l'élément 1
>>> n = l[:] # une COPIE complète de l
>>> l[0] = "pqr" # l est modifée, pas n
>>> l, n
(['pqr', 'def', 'ghi'], ['abc', 'xyz', 'ghi'])
```

Retour sur les tableaux

- On met ce qu'on veut dans un tableau

  $T = [1.44,\ 3.14,\ 2.72]$

- Pas forcement des objets de même nature

  $T = [\text{"}abc\text{"},\ 3.14,\ True]$

Élément $\neq$ indice

Ne pas confondre l'élément et l'indice

$T = [1.44,\ 3.14,\ 2.72]$

3.14 est l'élément, son indice est 1

$T[1]$ est 3.14

Tableaux multidimensionnels

Tableau bidimensionnel

Un **tableau bidimensionnel** ou **matrice** est un tableau qui contient des tableaux.

Tableaux de tableaux...

- Par exemple :

$$T \leftarrow [\ [a, b, c], [d, e, f], [m, n, o]\ ]$$

l/c	0	1	2
0	a	b	c
1	d	e	f
2	m	n	o

- On dit que T est une matrice à 3 lignes et 3 colonnes

Accéder à un élément

- Comment accéder à f ?

$$T \leftarrow [\ [a, b, c], [d, e, f], [m, n, o]\ ]$$

l/c	0	1	2
0	a	b	c
1	d	e	f
2	m	n	o

- f est l'élément $T[1][2]$

  ligne 1, colonne 2

Python : listes de listes

0	1
2	3

Pour présenter facilement on s'aligne au même niveau

```
T = [[0, 1],
     [2, 3]] # [[0, 1], [2, 3]]
T[1][0] # 2
```

Python : listes de listes par compréhension

Une méthode efficace :

```
T = [
     [j for j in range(3*i, 3*i+3)]
        for i in range(3)
     ]
```

Qu'obtient-on dans `T` ?

Python : listes de listes - solution

```
>>> T = [[j for j in range(3*i, 3*i+3)]
... for i in range(3)]

>>> T
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

l/c	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

Itérer dans une matrice.

pour itérer dans une matrice il faut 2 boucles *imbriquées*

```
Pour i allant de 1 à n {
  Pour j allant de 1 à n {
    faire... T[i][j] ...
  }
}
```

Calcul d'une moyenne des éléments d'un tableau.

Le tableau T comporte n lignes et p colonnes.

On calcule la moyenne habituelle donnée par la formule par

$$\frac{1}{n \times p} \sum_{i=0}^{n-1} \sum_{j=0}^{p-1} T[i][j]$$

- $\sum$ repésente la somme
- $\sum_{i=0}^{n-1}$: pour i allant de 0 à $n-1$ ajouter...
- Les deux $\sum$ sont l'une dans l'autre.
- On divise la somme des termes par $n \times p$

Exemple en Python.

Créer une fonction qui prenne une matrice en entrée et renvoie la moyenne de ses valeurs en sortie

Solution "naturelle"

```
def moyenne(T):
  s = 0; n = len(T); p = len(T[0])
  for i in range(n):
    for j in range(p):
      s += T[i][j]
  return s / (n * p)
```

Solution avec des outils de python

```
def moyenne(T):
  s = 0; k = len(T)*len(T[0])
  for ligne in T:
    for x in ligne:
      s += x
  return s / k
```

Solutions encore plus radicale...

```
def moyenne(T):
    s = 0; k = len(T) * len(T[0])
    for ligne in T:
        s += sum(ligne)
    return s/k
```

Python Tutor Et la plus courte à laquelle j'ai pensé :

```
def moyenne(T):
    return sum([sum(row) for row in T])\
      / (len(T) * len(T[0]))
```

Il y a encore plus court...

```
def moyenne(T):
  return sum(map(sum, T)) / (len(T) * len(T[0]))


def moyenne(T):
  return sum(sum(T,[])) / (len(T) * len(T[0]))
```

Exercices

Une étape de 2048

On considère une liste de nombres (exemple : `[2, 0, 4, 8]`)

Programmer une fonction `zeroADroite(liste)` qui renvoie une liste de même taille mais avec tous les 0 qu'elle contenait déplacés à droite.

Exemples :

```
>>> zeroADroite([2, 0, 4, 8])
[2, 4, 8, 0]
>>> zeroADroite([0, 0, 4, 0])
[4, 0, 0, 0]
>>> zeroADroite([2, 0, 4, 8])
[2, 4, 8, 0]
>>> zeroADroite([4, 0, 0, 16])
[4, 16, 0, 0]
```