Chap. 3 Tableaux Thème 2

Matrices : tableaux de tableaux

Sommaire du chapitre 3

I Structures imbriquées

- **Les structures imbriquées**
  - On peut **imbriquer** des p-uplets, des tableaux et des dictionnaires. La seule règle est qu'une clé de dictionnaire doit être immuable.
  - On peut alors construire des tableaux de tableaux, des tableaux de p-uplets, des tableaux de dictionnaires, des p-uplets de tableaux, des p-uplets de dictionnaires, des dictionnaires ayant des tableaux ou des p-uplets comme valeurs, ...

- **Parcours d'une structure imbriquée**
  Le parcours d'une structure imbriquée nécessite plusieurs boucles.

II Création d'un tableau de tableaux

Un tableau peut lui même contenir des tableaux. Une **matrice** est ainsi un tableau contenant un nombre quelconque de tableaux de même longueur, dont les éléments sont tous de même type (généralement des nombres). Une matrice forme ainsi une sorte de « tableau » rectangulaire à deux dimensions.

Exemples

L'instruction `T = [[3, 4], [8, 1]]` crée le tableau de tableaux `T` représentant le tableau suivant

3	4
8	1

Remarques

Lorsque l'on souhaite attribuer à chaque colonne un nom particulier (ou des types différents), on utilisera des tables plutôt que des matrices, comme nous le verrons au chapitre 5.

La figure ci-dessous montre comment un tableau de tableau `m` représente une matrice : `m[y]` est un tableau contenant la $(y+1)$-ième ligne de la matrice, donc `(m[y])[x]`, que l'on écrit `m[y][x]`, accède à la case d'indixe `x` dans ce tableau. Ici, `m[1][0]` vaut `'a21'`, `m[0][3]` vaut `'a14'` et `m[2][3]` vaut `'zc'`.

Définition Python:

```
m = [['a11', 'a12', 'a13', 'a14'],
['a21', 'a22', 'a23', 'a24'],
[ 'z', 'za', 'zb', 'zc']]
# il n'est pas nécessaire de
# passer à la ligne
```

Représentation mathématique:

$$\begin{array}{c} \xrightarrow{\quad x \quad} \\ \left(\begin{array}{cccc} 'a11' & 'a12' & 'a13' & 'a14' \\ 'a21' & 'a22' & 'a23' & 'a24' \\ 'z' & 'za' & 'zb' & 'zc' \end{array}\right) \\ \downarrow y \end{array}$$

Un peu d'histoire

Les matrices ont été utilisées dès le $II^{ème}$ avant JC en Chine et dès le $XVI^{ème}$ en Europe pour résoudre des systèmes d'équations. Elles se sont imposées comme un outil incontournable en algèbre linéaire au cours du $XIX^{ème}$.

III Parcours d'un tableau de tableaux

En l'absence de tri préliminaire, une recherche dans une matrice imposera, comme pour les tableaux, de parcourir l'ensemble de la matrice. On va donc généralement écrire deux boucles imbriquées :

```
1 def recherche_matrice(v1, m) :
2     """ Renvoie True ssi v1 est dans la matrice """
3     for ligne in m :
4         for v2 in ligne : # on effectue ici une recherche_tableau
5             if v1 == v2 :
6                 return True
7     return False
```

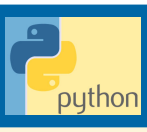
Remarques

Le code ci-dessus itère sur les tableaux. On peut bien sûr aussi parcourir la matrice par une boucle imbriquée sur les indices :

```
n = len(m)
p = len(m[0])
for i in range(n) :
    for j in range(p) :
        ...
```

Remarques

- On pourrait représenter une matrice `a` de taille $3 * 2$ (ou $n * m$) par un unique tableau `t` de longueur 6 (ou $n \times m$), en utilisant `t[n*i+j]` pour stocker la case `a[i][i]`. Mais un tableau de tableaux est plus pratique.
- Un inconvénient de l'implémentation des matrices par tableau de tableaux en Python est qu'on n'y contrôle ni la taille de chaque tableau, ni le type des données contenues : rien n'empêche l'utilisateur de modifier la matrice `m` ci-dessus par `m[1]=['aa']`, `m[2]=[8,0]` ; le résultat n'est plus une matrice.



Les matrices sont très utilisés en science, et sont donc déjà implémentées en Python dans une librairie dédiée, `NumPy`, que nous n'étudierons pas.