M1102: Introduction à l'algorithmique et à la programmation

Feuille de TD n°9

Implémentation des structures de données complexes

INSTRUCTIONS: à la fin du TD, vous devez déposer sous Celene une archive zip contenant les fichiers Python correspondant à ce TD (fournis avec la feuille). Vous pourrez déposer une version améliorée de votre travail jusqu'à la **VEILLE** du TD suivant.

L'évaluation de la période 2 sera basée sur ces rendus.

Objectifs

L'objectif de cette feuille de TD est de travailler sur la notion de structures de données complexes qui sont amenées à être utilisées dans différents contextes comme la structure de matrice par exemple. Ce travail va préparer à l'introduction de la programmation orientée objet en Python.

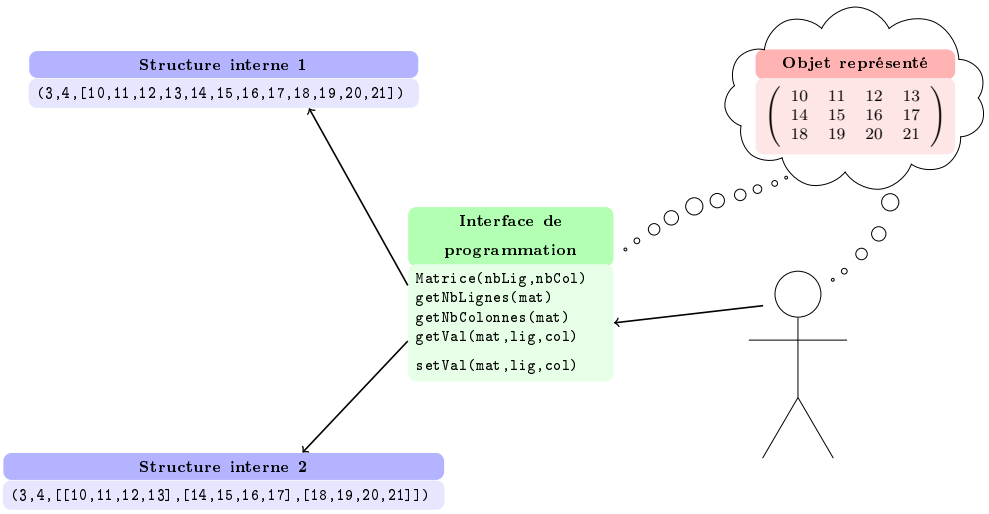

Figure 1 – Les différentes visions d'une structure de données

Afin de faciliter l'utilisation d'une telle structure, il est très important de bien séparer la représentation interne de la structure de données d'un côté et la manière d'accéder aux informations contenues dans cette structure de l'autre côté (interface de programmation ou API).
Une première représentation des matrices peut être un triplet `(nbLig,nbCol,listeVal)` : ce triplet est la représentation interne des matrices. De l'autre côté, on a une interface de programmation permettant de manipuler et de consulter des matrices. Cette API est composées des fonctions suivantes : `Matrice(nbLig,nbCol)`, `getNbLignes(mat)`, `getNbColonnes(mat)`, `getVal(mat,lig,col)`, `setVal(mat,lig,col)`.
Ces fonctions n'induisent pas la représentation interne de la matrice mais permettent de faire les opérations élémentaires sur une matrice. En effet, on pourrait décider de représenter les

matrices par un triplet `(nbLig,nbCol,listeDeLignes)` où `listeDeLignes` est une liste dont les éléments sont des listes contenant les valeurs d'une ligne et d'utiliser la même interface de programmation juste en ré-implémentant les cinq fonctions de cette API. De cette manière, on peut écrire des fonctions utilisant des matrices indépendantes de l'implémentation qui en est faite.
La figure 1 décrit d'un coté la manière dont un utilisateur se représente la structure de données matrice et l'interface de programmation qu'il utilise ; de l'autre coté, les deux représentations internes des matrices citées précédemment.

Exercice 1 *Une première implémentation de la structure de matrice*

La structure de *matrices* est une structure de données très utilisée en mathématiques mais aussi en informatique ! Ce nom de *matrice* peut effrayer certains informaticiens débutants mais en fait elle peut prendre des noms moins inquiétants comme grille ou labyrinthe, lorsqu'il s'agit de représenter un plateau de jeu (comme pour le Puissance 4 ou un labyrinthe dans lequel évolue des personnages comme le PacMan ou le Bomberman). Beaucoup des exercices que vous aurez à faire vous seront utiles à l'implémentation du projet de fin de semestre.
Dans le fichier `matrice.py` vous avez l'implémentation des fonctionnalités de base de manipulation des matrices 2D. Cette représentation utilise un triplet `(nbLig,nbCol,listeVal)` où les deux premiers composants sont des entiers indiquant respectivement le nombre de lignes et le nombre de colonnes de la matrice et le dernier est une liste de `nbLig` $\times$ `nbCol` éléments contenant les valeurs contenues dans la matrice. Ces éléments sont rangés comme illustré ci-dessous.

$$\begin{pmatrix} 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 \end{pmatrix}$$

sera représentée par le triplet `(3,4,listeVal)` où `listeVal` est la liste suivante.

indices	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
valeurs	10	11	12	13	14	15	16	17	18	19	20	21

Ainsi l'élément 0,0 de la matrice se trouve à l'indice 0 de `listeVal` est l'élément `i,j` à l'indice 4*i+j (4 étant le nombre de colonnes).
Les fonctions présentes dans `matrice.py` permettent de manipuler une matrice 2D sans se soucier de cette représentation. Vous pouvez tester la conformité des fonctions que vous avez implémentées au fur et à mesure du TD en tapant la commande

```
./tests_exo1.py -v
```

- `Matrice(nbLignes,nbColonnes,valeurParDefaut=0)` créer une nouvelle matrice en mettant la valeur par défaut dans chacune des cases de celle-ci (`valeurParDefaut=0` signifie que si l'utilisateur de la fonction ne donne pas de 3ème paramètre celui-ci sera replacé par la valeur 0).
- `getNbLignes(matrice)` permet de connaître le nombre de lignes de la matrice.
- `getNbColonnes(matrice)` permet de connaître le nombre de colonnes de la matrice.
- `getVal(matrice,lig,col)` permet de connaître la valeur de l'élément $a_{lig,col}$ de la matrice
- `setVal(matrice,lig,col,val)` permet de mettre la valeur `val` dans l'élément $a_{lig,col}$ de la matrice.
- `afficheMatrice(matrice,tailleCellule=4)` affiche une matrice dont les cellules auront une largeur de `tailleCellule`

Voici la liste des fonctions à implémenter sur les matrices.

1. `isNulle(matrice)` qui indique si la matrice ne contient que des valeurs nulles ou non.
2. `isCarre(matrice)` qui indique si une matrice est carrée ou non

3. `moyenne(matrice)` qui calcule la moyenne des valeurs d'une matrice.

   A la fin de cet exercice il est normal que le test de l'addition ne fonctionne pas car il n'est pas encore implémenté (voir exercice 2).

Exercice 2 *Nouvelle représentation des matrices*

Constant Denlechangeman, votre chef de projet, a décidé que l'implémentation des matrices de l'exercice précédent ne lui convenait pas. *Il faut revenir à l'essentiel* vous indique-t-il. Il veut une représentation sous la forme d'une liste de lignes sans autres fioritures. $\begin{pmatrix} 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 \end{pmatrix}$ est représentée par `[[10,11,12,13],[14,15,16,17],[18,19,20,21]]`.

1. Sauvegarder le début de votre fichier `matrice.py` (jusqu'à l'endroit indiqué) dans le fichier `matriceAPI1.py` et remplacez cette partie par l'instruction `from matriceAPI1 import *`. Cette ligne indique que vous importez toutes les définitions données dans le fichier `matriceAPI1.py`. Vérifiez que vos fonctions de `matrice.py` marchent toujours avec le cas de tests `tests_exo1.py`
2. Réécrivez les fonctions `Matrice(nbLig,nbCol)`, `getNbLignes(mat)`, `getNbColonnes(mat)`, `getVal(mat,lig,col)`, `setVal(mat,lig,col)` pour cette nouvelle représentation en utilisant le fichier Python `matriceAPI2.py`.
3. Vérifiez que les fonctions que vous aviez écrites dans le fichier `matrice.py` marchent toujours en remplace l'instruction `from matriceAPI1 import *` par `from matriceAPI2 import *`. Si ce n'est pas le cas modifiez vos fonctions pour qu'elles marchent avec les deux implémentation des matrices. Le cas de tests pour la deuxième implémentation des matrices est `tests_exo2.py`.

   Attention ! `tests_exo2.py` ne fonctionne qu'avec `matriceAPI2` et `tests_exo1.py` ne fonctionne qu'avec `matriceAPI1` puisque les cas de tests dépendent de la représentation interne des matrice.
4. Constant avait écrit une super fonction, dont le code se trouve figure 4, bien optimisée pour l'addition des matrices qui ne marche plus du tout avec la nouvelle représentation. Il vous demande de la réécrire de manière à ce qu'elle fonctionne à la fois dans l'ancienne représentation et dans la nouvelle.

FIGURE 2 – Fonction d'addition de deux matrices

```
1 def additionMat(mat1,mat2):
2     #cas des matrices pas de même taille
3     #ça n'a pas l'air de bien marcher!!!
4     if len(mat1[2])!=len(mat2[2]):
5         return None
6     # les matrices ont la même taille
7     return (mat1[0],mat1[1],[mat1[2][i]+mat2[2][i]
8                              for i in range(len(mat1[2]))])
```

Exercice 3 *D'autres fonctions sur les matrices (Exercices supplémentaires)*

Complétez le fichier `matrice.py` pour y intégrer les fonctions suivantes :

- `getLigne(matrice,lig)` qui retourne sous la forme d'une liste, la ligne n°`lig)` de la matrice.
- `getColonne(matrice,col)` qui retourne sous la forme d'une liste, la colonne n°`col` de la matrice.
- `getDiagonalePrincipale(matrice)` qui retourne sous la forme d'une liste la diagonale principale d'une matrice carrée.
- `getDiagonaleSecondaire(matrice)` qui retourne sous la forme d'une liste la diagonale secondaire d'une matrice carrée.
- `getBloc(matrice,lig,col,largeur,hauteur)` qui retourne la sous-matrice de la matrice commençant à l'élément $a_{lig,col}$ et qui a comme dimensions `largeur` `hauteur`.
- `transpose(matrice)` qui retourne la transposée d'une matrice
- `isTriangulaireInf(matrice)` qui indique si une matrice est triangulaire inférieure
- `isTriangulaireSup(matrice)` qui indique si une matrice est triangulaire supérieure