Devoir ITC n°2

Autour des matrices

Le sujet suivant se compose de 3 parties indépendantes. Vous pouvez traiter les différentes parties dans l'ordre que vous souhaitez.

Lorsqu'une fonction est demandée par l'énoncé, même si vous ne proposez pas de code pour la définir, **vous pouvez supposer qu'elle est définie et vous en servir dans les questions suivantes**.

Dans tout le sujet, les fonctions sont données avec leur signature sous forme d'annotations : par exemple, `fonction(x:int, y:float) -> list` signifie que la fonction prend deux paramètres x (de type entier) et y (de type flottant) et renvoie une liste. Dans le sujet, nous définissons le type `Mat` comme étant une matrice, c'est-à-dire une liste de listes comme défini au-dessus. **Il n'est pas demandé d'écrire les annotations dans la copie**.

Les points suivants seront pris en compte à la notation :

- vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction ; 1 point pourra être attribué en bonus ou en malus selon la lisibilité de la copie ;
- de même, il est demandée de **tirer un trait entre deux questions** afin de les délimiter clairement ;
- la syntaxe sera évaluée de façon bienveillante, des erreurs mineures ne seront pas sanctionnées ; cependant, une syntaxe Python complètement fantaisiste fera perdre des points de présentation ;
- pour chaque fonction, un point sera attribué si celle-ci a une approche naturelle pour répondre au problème ;
- il est fortement recommandé de délimiter les niveaux d'indentation avec des traits verticaux ;
- si vous êtes amené à repérer ce qui vous semble être une erreur d'énoncé, vous le signalerez sur votre copie et devrez poursuivre votre composition en expliquant les raisons des initiatives que vous avez prises ;
- il est essentiel de respecter le temps imparti : les stylos seront levés à la fin de la composition ;
- le sujet ne sera **pas** rendu avec la copie.

Sur ce, bon courage ☺

Le sujet suivant s'intéresse aux opérations sur les matrices ; on rappelle qu'au sens mathématique, une matrice carrée de taille n peut être vue comme un tableau de nombres : par exemple, on propose ci-dessous une matrice A de taille $n = 4$ à valeurs dans $\mathbb{Z}$ et une représentation d'une matrice M de taille n quelconque par ses éléments $M_{i,j}$, i étant le numéro de ligne et j celui de colonne de la matrice :

$$A = \begin{pmatrix} 7 & 2 & 8 & 1 \\ 1 & -1 & -3 & 0 \\ 2 & -2 & 2 & 9 \\ 4 & 7 & 5 & -7 \end{pmatrix} \quad ; \quad M = (M_{i,j}) = \begin{pmatrix} M_{0,0} & M_{0,1} & \cdots & M_{0,n-1} \\ M_{1,0} & M_{0,1} & \cdots & M_{0,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ M_{n-1,0} & M_{n-1,1} & \cdots & M_{n-1,n-1} \end{pmatrix}$$

On appelle **matrice nulle** la matrice 0_n ne contenant que des zéros et **matrice identité** la matrice I_n contenant des 1 sur la diagonale et des zéros partout ailleurs ; par exemple pour $n = 3$:

$$0_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{et} \quad I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Dans un programme informatique en Python, les matrices sont représentées comme des listes de listes de nombres. En effet, si `m` est une liste de n listes, chacune étant une liste de taille n contenant des nombres, alors :

- `m[i]` pour $i \in [\![0; n-1]\!]$ est une liste de taille n représentant la i^{e} ligne de la matrice ;
- `m[i][j]` pour $(i, j) \in [\![0; n-1]\!]^2$ est le j^{e} élément de la liste `m[i]` , donc le j^{e} élément de la i^{e} ligne de `m` ; finalement, `m[i][j]` est bien l'élément de la matrice `m` en ligne i, colonne j.

I. Manipulations élémentaires sur une matrice

1 – Écrire une fonction `matrice_nulle(n: int) -> Mat` qui prend en paramètre un entier n et renvoie la matrice nulle de taille $n \times n$.

2 – Écrire une fonction `matrice_identité(n: int) -> Mat` qui prend en paramètre un entier n et renvoie la matrice identité de taille $n \times n$. On

3 – Écrire une fonction `copie_matrice(m: Mat) -> Mat` qui prend en paramètre une matrice `m` et renvoie une nouvelle matrice, indépendante, contenant les mêmes éléments que `m` .

On appelle **trace** de la matrice la somme des éléments sur sa diagonale, soit

$$\mathrm{tr}(M) = \sum_{i=0}^{n-1} M_{i,i}$$

Par exemple, la trace de la matrice identité vaut n, et celle de la matrice A donnée en exemple au début du sujet est 1.

4 – Écrire une fonction `trace(m: Mat) -> int` qui prend en paramètre une matrice `mat` et renvoie la trace de la matrice.

La somme de deux matrices carrées de taille n est une matrice carrée de taille n dont les éléments sont les sommes des deux matrices d'entrées :

$$S_{i,j} = M_{i,j} + N_{i,j}$$

5 – Pourquoi l'instruction `s = m1 + m2` ne fait pas la somme de deux matrices au sens ci-dessus en Python ? Illustrer le problème sur le cas

$$m_1 = \begin{pmatrix} 2 & 3 \\ 1 & 1 \end{pmatrix} \quad \text{et} \quad m_2 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

6 – Écrire une fonction `add(m1: Mat, m2:Mat) -> Mat` qui renvoie la somme de deux matrices `m1` et `m2` . Quelle est sa complexité ?

II. Déterminant d'une matrice

On peut calculer[1] le déterminant d'une matrice carrée de la façon suivante :

- si $n = 1$,

$$\det(M) = M_{0,0}$$

- si $n > 1$, on peut ramener le calcul du déterminant à celui de n déterminants de taille $n-1$ en développant selon la colonne j :

$$\det(M) = \sum_{i=0}^{n-1} (-1)^{i+j} M_{i,j} \det(A_{i,j})$$

où $A_{i,j}$ est la matrice de taille $n-1$ obtenue en retirant la ligne i et la colonne j de la matrice j.

7 – Écrire une fonction `extraire_sous_matrice(m: Mat, i: int, j: int)` qui extrait la matrice A_{ij} de taille $n-1$ et la renvoie.

8 – Écrire une fonction `déterminant(m: Mat) -> int` qui calcule le déterminant d'une matrice.

III. Produits de matrices

Par définition, le produit P de deux matrices M et N est obtenu en sommant les termes produits de la i^{e} ligne de M et la j^{e} colonne de N :

$$P_{i,j} = \sum_{k=0}^{n-1} M_{i,k} N_{k,j}$$

Par exemple,

$$\begin{pmatrix} 2 & 1 \\ 3 & -1 \end{pmatrix} \cdot \begin{pmatrix} 5 & 4 \\ -2 & -3 \end{pmatrix} = \begin{pmatrix} 2\times 5 + 1\times(-2) & 2\times 4 + 1\times(-3) \\ 3\times 5 + (-1)\times(-2) & 3\times 4 + (-1)\times(-3) \end{pmatrix} = \begin{pmatrix} 8 & 5 \\ 17 & 15 \end{pmatrix}$$

La puissance p d'une matrice M est définie comme la répétition de p multiplications de la matrice :

$$M^p = M \cdot M^{p-1} = \underbrace{M \cdot M \cdot \ldots \cdot M}_{p \text{ fois}}$$

9 – Écrire une fonction `prod(m1: Mat, m2: Mat) -> Mat` qui calcule ainsi le produit de deux matrices en appliquant la définition. La complexité doit être en $\mathcal{O}(n^3)$, et on justifiera succintement cette complexité.

10 – En utilisant la fonction `prod`, écrire une fonction `puiss(m: Mat, p: int) -> Mat` qui calcule la puissance p de la matrice m. La complexité attendue est $\mathcal{O}(n^3 p)$, sans justification.

Pour les matrices comme pour les nombres, on peut améliorer le temps de calcul par l'algorithme d'**exponentiation rapide** : au lieu de calculer p fois le produit, on calcule $y = x^{p//2}$ puis

$$x^p = \begin{cases} y \times y & \text{si } p \text{ est pair} \\ x \times y \times y & \text{si } p \text{ est impair} \end{cases}$$

11 – À quelle famille d'algorithmes appartient l'algorithme d'exponentiation rapide ?

12 – En utilisant l'algorithme d'exponentiation rapide, proposer une fonction **récursive** `puiss2(m: Mat, p: int) -> Mat` dont on donnera la complexité sans justification.

[1] La définition correcte sera dans votre cours de mathématiques

On s'intéresse à des améliorations possibles du calcul du produit lui-même ; dans un premier temps, on remarque que si on décompose les matrices A, B et le produit C en quatre sous-matrices de taille égale :

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad ; \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \quad ; \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

alors la formule pour calculer les sous-matrices de C est la même que pour une matrice de nombres 2×2 :

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

Ainsi, le calcul du produit $A \cdot B$ de matrices de taille n se ramène au calcul de 8 produits de matrices de taille $n/2$, ainsi que 4 sommes. **Dans toute la suite, on supposera que les tailles des matrices sont des puissances de 2** : $n = 2^k$. On suppose également que l'on dispose des fonctions

```
découpage(m: Mat) -> (Mat, Mat, Mat, Mat)
reconstruction(m11: Mat, m12: Mat, m21: Mat, m22: Mat) -> Mat
```

qui permettent respectivement de décomposer une matrice en ses 4 sous-matrices, et de prendre 4 sous-matrices pour recréer une grande matrice. On admet que ces fonctions peuvent avoir une complexité $\mathcal{O}(1)$.

13 – Écrire une fonction **récursive** `prod2(m1: Mat, m2: Mat) -> Mat` qui utilise cette décomposition pour calculer le produit de deux matrices de taille n.

14 – On note c_n le temps de calcul de `prod2` pour une matrice de taille n, et an le temps de calcul de la somme de deux matrices de taille n avec a une constante, montrer que c_n est de la forme

$$c_n = \alpha c_{n/2} + \beta an + b$$

avec b une constante, et α et β à préciser.

15 – En introduisant $w_k = c_{2^k}$, et en négligeant le temps de calcul βan et b devant $\alpha c_{n/2}$, montrer que $c_n = \mathcal{O}(n^3)$. A-t-on gagné une meilleure complexité par rapport à la fonction `prod` ?

Pour améliorer cette complexité, on utilise l'algorithme de Strassen : on décompose les matrices comme précédemment, mais on effectue le calcul des sous-matrices de C de la façon suivante :

$$\begin{cases} M_1 = (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \\ M_2 = (A_{2,1} + A_{2,2}) \cdot B_{1,1} \\ M_3 = A_{1,1} \cdot (B_{1,2} - B_{2,2}) \\ M_4 = A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\ M_5 = (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\ M_6 = (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \\ M_7 = (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \end{cases} \quad \text{et} \quad \begin{cases} C_{1,1} = M_1 + M_4 - M_5 + M_7 \\ C_{1,2} = M_3 + M_5 \\ C_{2,1} = M_2 + M_4 \\ C_{2,2} = M_1 - M_2 - M_3 + M_6 \end{cases}$$

16 – En adaptant le calcul de complexité précédent, calculer la complexité de l'algorithme de Strassen.

17 – En combinant les résultats précédents, quelle complexité peut-on espérer au mieux pour le calcul de la puissance p d'une matrice de taille n ?