$\mathcal{M}$	$\mathcal{P}$	$\mathcal{S}$	$\mathcal{I}$	2

LYCEE CHARLEMAGNE
2020/2021

MARDI 17 NOVEMBRE

I.P.T.

Avec le cours de maths, vous connaissez l'addition et la multiplication matricielle. Du moins j'espère.
Mais on va étudier ici l'addition industrielle et la multiplication industrielle, appelée aussi algèbre des sandwiches.
Explication : on étudie le temps nécessaire à l'exécution de taches en parallèle et en série.
Pourquoi sandwich ? Vous êtes déjà allé chez Subway ? Le sandwiche est fabriqué suivant des méthodes industrielles « à la chaîne ».
L'employé doit effectuer des taches successives pour garnir votre sandwich (prendre le pain, le préchauffer, l'ouvrir, mettre des ingrédients, une sauce, emballer...). Pour ces tâches, les temps s'additionnent. Mais il y a aussi des tâches effectuées en parallèle (un autre employé remplit les bacs de tranches de concombres, de rondelles de cornichons, de demi olives, apporte les stocks de canettes...). Si il tarde, le premier employé est retardé. Si il va plus vite que le premier employé, c'est le temps de réalisation du premier qui entre en jeu. Cette fois, c'est le maximum des temps qui intervient (un employé doit attendre si l'autre met plus de temps que lui).

Extrait d'un article de Pour La Science, repris pour un cours de l'ENSTA Bretagne

Nous avons tous appris à l'école que l'on n'additionne pas des oranges et des bananes. Ne pourrions-nous ajouter que des quantités de même nature, exprimées dans une même unité ? L'expression « un mètre plus deux kilogrammes » serai-telle vide de sens ? Non. Il existe des situations de la vie courante où nous transgressons ces règles sans y penser et où nous utilisons une autre façon de compter.

Un fabricant d'automobiles n'a aucun mal à « additionner » un moteur, une carrosserie et quatre roues : cela donne une voiture. Pour l'employé d'une boulangerie qui fabrique des sandwichs jambon-fromage, un morceau de baguette « plus » une tranche de jambon « plus » une lamelle de fromage font un sandwich. Dans cette situation, l'addition de trois quantités de natures différentes donne un résultat de nature distincte. Selon cette opération, « 1 + 1 + 1 = 1 ». De surcroît, si l'employé garnit le sandwich de deux tranches de jambon au lieu d'une, le résultat reste un sandwich unique. Autrement dit, « 2 + 1 + 1 = 1 ».

L'opération revient à prendre le minimum des quantités qu'il faut associer : le nombre de sandwichs préparés est égal au minimum des quantités de pain, de jambon et de fromage disponibles. Cette étrange façon de compter correspond à une structure mathématique nommée algèbre min-plus.

Ce n'est pas une curiosité mathématique : elle permet, avec l'algèbre associée max-plus, de décrire la plupart des systèmes qui nécessitent une synchronisation de ressources, tels les ateliers de production industriels, les correspondances d'un réseau ferroviaire ou la gestion des feux de circulation. Après avoir présenté ces systèmes dits à événements discrets et leur représentation graphique, nous détaillerons les propriétés des algèbres min-plus et max-plus, puis nous analyserons ces systèmes à l'aide de ces outils mathématiques.
Dans l'étude des systèmes dynamiques classiques, on examine l'évolution d'un état du système en fonction du temps, qui peut s'écouler de façon continue ou progresser par à-coup, mais de façon autonome. Or certains systèmes dynamiques fonctionnent différemment.

Imaginons un voyageur qui désire aller en train dans une ville voisine. Qu'il se rende à la gare une demi-heure ou cinq minutes en avance par rapport à l'horaire de départ ne changera pas son heure d'arrivée : il doit patienter jusqu'à l'entrée en gare du train. Durant le trajet, rien ne se passe jusqu'à ce qu'il descende et prenne une correspondance. Les seuls événements significatifs de cette situation sont les rencontres du voyageur avec le train, puis avec la correspondance. L'état de ce type de système n'évolue que lors de ces instants ponctuels définis par leur dynamique même, indépendamment de l'écoulement du temps. On les nomme systèmes à événements discrets. Dans ces systèmes, les actions ne peuvent être accomplies que lorsqu'un certain nombre de conditions (autorisations ou ressources matérielles) sont réunies. L'enchaînement des processus dans nombre de systèmes réels – réseaux de transport, chaînes de production, automates – repose sur cette synchronisation : un voyage nécessite la conjonction d'un voyageur sur le quai et d'un train arrivant en gare ; la préparation d'un sandwich réclame simultanément du pain, du jambon et du fromage ; l'assemblage d'une automobile n'est possible que si une machine et des pièces sont disponibles en même temps, etc.

On crée donc une structure dans la quelle on remplace l'addition par le maximum
la multiplication par la somme.
On a donc un presque-anneau $(\mathbb{R}^+, Max, +)$. Si on était en cours de maths, je vous demanderais de vérifier $(\mathbb{R}^+, Max)$ interne
associative $Max(a,\ Max(b,c)) = Max(Max(a,b),\ c)$
commutative
avec un neutre : $Max(a,0) = a$
mais hélas sans symétriques $Max(a,\ ?) = 0$
$(\mathbb{R}^+, Max, +)$ interne
associative $(a+b)+c = a+(b+c)$
commutative
distributive sur Max
avec un neutre (le même que Max !)
Arrêtons nous sur « $+$ distributive sur Max » pour nous habituer à passer d'une formule à l'autre :

$a.(b+c)$	$=$	$a.b + a.c$
$a + Max(b,\ c)$	$=$	$Max(a+b,\ a+c)$

Vous pouvez vous entrainer à écrire des choses comme les identités remarquables ou la formule du binôme

:

$(a+b)^2 = a^2 + b^2 + 2.a.b$	$2.Max(a,b) = Max(2.a,\ 2.b,\ (a+b))$
$(a+b)^3 = a^3 + 3.a^2.b + 3.a.b^2 + b^3$	?

Oui, quand on passe à l'anneau $(\mathbb{R}^+, Max, +)$, le terme $2.a.b$ (à lire $a.b + a.b$) doit se lire $Max(a+b,\ a+b)$!)

Et il existe aussi l'algèbre $\mathbb{R}^+, Min, +)$.

On peut même ensuite commencer à faire du calcul matriciel, qui consiste à remplacer l'addition classique par le Max (terme à terme) et la formule de multiplication par... oui, tiens, par quoi ?

Addition classique	
$\begin{pmatrix} a & b & c \\ a' & b' & c' \\ a'' & b'' & c'' \end{pmatrix} + \begin{pmatrix} \alpha & \beta & \gamma \\ \alpha' & \beta' & \gamma' \\ \alpha'' & \beta'' & \gamma'' \end{pmatrix}$	$= \begin{pmatrix} a+\alpha & b+\beta & c+\gamma \\ a'+\alpha' & b'+\beta' & c'+\gamma' \\ a''+\alpha'' & b''+\beta'' & c''+\gamma'' \end{pmatrix}$
$\begin{pmatrix} a_1^1 & \dots & a_1^n \\ \vdots & & \vdots \\ a_n^1 & \dots & a_n^n \end{pmatrix} + \begin{pmatrix} b_1^1 & \dots & b_1^n \\ \vdots & & \vdots \\ b_n^1 & \dots & b_n^n \end{pmatrix}$	$= \begin{pmatrix} a_1^1+b_1^1 & \dots & a_1^n+b_1^n \\ \vdots & & \vdots \\ a_n^1+b_n^1 & \dots & a_n^n+b_n^n \end{pmatrix}$
A de terme général a_i^k B de terme général b_i^k	$S = A + B$ a pour terme général s_i^k avec $c_i^k = a_i^k + b_i^k$
Addition industrielle	
$\begin{pmatrix} a_1^1 & \dots & a_1^n \\ \vdots & & \vdots \\ a_n^1 & \dots & a_n^n \end{pmatrix} \oplus \begin{pmatrix} b_1^1 & \dots & b_1^n \\ \vdots & & \vdots \\ b_n^1 & \dots & b_n^n \end{pmatrix}$	$= \begin{pmatrix} Max(a_1^1,\ b_1^1) & \dots & Max(a_1^n,\ b_1^n) \\ \vdots & & \vdots \\ Max(a_n^1,\ b_n^1) & \dots & Max(a_n^n,\ b_n^n) \end{pmatrix}$
A de terme général a_i^k B de terme général b_i^k	$C = A \oplus B$ a pour terme général c_i^k avec $c_i^k = Max(a_i^k,\ b_i^k)$

On rappelle la notation mathématique pour le terme général d'une matrice : $a^{colonne}_{ligne}$ (et c'est facile a_i^kavec i comme lIgne et k comme Kolonne).

Multiplication classique	
$\begin{pmatrix} a & b \\ a' & b' \end{pmatrix} + \begin{pmatrix} \alpha & \beta \\ \alpha' & \beta' \end{pmatrix}$	$= \begin{pmatrix} a.\alpha + b.\alpha' & a.\beta + b.\beta' \\ a'.\alpha + b'.\alpha' & a'.\beta + b'.\beta' \end{pmatrix}$
$\begin{pmatrix} a_1^1 & \dots & a_1^n \\ \vdots & & \vdots \\ a_n^1 & \dots & a_n^n \end{pmatrix} + \begin{pmatrix} b_1^1 & \dots & b_1^n \\ \vdots & & \vdots \\ b_n^1 & \dots & b_n^n \end{pmatrix}$	$= \begin{pmatrix} a_1^1.b_1^1 + \dots + a_1^n.b_n^1 & \dots & a_1^1.b_1^n + \dots + a_1^n.b_n^n \\ \vdots & & \vdots \\ a_n^1.b_1^1 + \dots + a_n^n.b_n^1 & \dots & a_n^1.b_1^n + \dots + a_n^n.b_n^n \end{pmatrix}$
A de terme général a_i^k B de terme général b_i^k	$P = A \times B$ a pour terme général s_i^k avec $c_i^k = a_i^1.b_1^k + a_i^2.b_2^k + \dots + a_i^n.b_n^k$ $c_i^k = \sum_{j=1}^{n} a_i^j.b_j^k$
Multiplication industrielle	
$\begin{pmatrix} a_1^1 & \dots & a_1^n \\ \vdots & & \vdots \\ a_n^1 & \dots & a_n^n \end{pmatrix} \oplus \begin{pmatrix} b_1^1 & \dots & b_1^n \\ \vdots & & \vdots \\ b_n^1 & \dots & b_n^n \end{pmatrix}$	$= \begin{pmatrix} Max(a_1^1,\ b_1^1) & \dots & Max(a_1^n,\ b_1^n) \\ \vdots & & \vdots \\ Max(a_n^1,\ b_n^1) & \dots & Max(a_n^n,\ b_n^n) \end{pmatrix}$
A de terme général a_i^k B de terme général b_i^k	$C = A \oplus B$ a pour terme général c_i^k avec $c_i^k = Max(a_i^1 + b_1^k,\ a_i^2 + b_2^k,\ \dots,\ a_i^n + b_n^k)$ $c_i^k = Max(a_i^k + b_i^k \mid j \leqslant n)$

On notera que dans les formules $c_i^k = a_i^1.b_1^k + a_i^2.b_2^k + \dots + a_i^n.b_n^k)$ et

$$c_i^k = Max(a_i^1 + b_1^k,\ a_i^2 + b_2^k,\ \dots,\ a_i^n + b_n^k)$$

qui calcule le terme de ligne i et colonne k de la matrice « produit », tous les a_i^j viennent de la ligne i de la matrice A

tous les b_j^k viennent de la colonne k de la matrice B

Si j'ai le temps et le courage, je vous fais un joli dessin en couleur.

Il est temps de passer à la mise en place de l'addition et de la multiplication.
Pour l'addition et la multiplication usuelle, il existe des modules tout prêts sous `numpy` que je devrai vous enseigner pour les concours.
Mais on est en informatique que diable !
On n'utilise pas des boites noires sans réfléchir, sans savoir les reconstruire soi même.
Alors on y va « brut », avec des listes et des listes de listes.

Une matrice est un tableau, c'est donc une liste de listes.
La matrice $\begin{pmatrix} a & b & c \\ a' & b' & c' \\ a'' & b'' & c'' \end{pmatrix}$ c'est trois lignes $\begin{pmatrix} a & b & c \end{pmatrix}$, $\begin{pmatrix} a' & b' & c' \end{pmatrix}$ et $\begin{pmatrix} a'' & b'' & c'' \end{pmatrix}$.
C'est donc une liste de trois lignes `[[a,b,c], [a',b',c'], [a",b",c"]]`.
Et $\begin{pmatrix} a & b & c & d \\ a' & b' & c' & d' \end{pmatrix}$ c'est `[[a,b,c,d], [a',b',c',d']]`.
Quant à `[[a,b,c], [a',b',c'], [a",b"]]`
ou `[[a,b,c], [a',b',c',d'], [a",b",c"]]`
ou `[[a,b,c], a',b',c', [a",b",c"]]`,
ce ne sont pas des matrices.

Pour créer une matrice, des crochets.

Et pour l'afficher ?
Si vous avez tapé `A=[[1, 2, 3], [4, 0, 1], [2, 0, 2]]` (en pensant à $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 0 & 1 \\ 2 & 0 & 2 \end{pmatrix}$ et que vous validez `print(A)`, c'est peu lisible.
`[[1, 2, 3], [4, 0, 1], [2, 0, 2]]`

Il faudrait afficher chaque ligne l'une après l'autre.
Et les trois lignes de `A`, c'est `A[0]`, `A[1]` et `A[2]`.
D'où deux syntaxes possibles

		affichage

`def Affiche(A):`
`....for k in range(len(A)):`
`........print(A[k])`
	`for ligne in A:`
	`....print(ligne)`
		[1, 2, 3]
		[4, 0, 1]
		[2, 0, 2]

Vérifiez en définissant la procédure `Affiche` puis en lui soumettant une matrice.

La première est « basique pour un élève formé par un prof de maths et/ou un prof de physique ».
La seconde est « basique pour un élève formé par des informaticiens » (mais pas forcément comprise par certains correcteurs un peu bas de plafond aux concours, si si).

On note au passage que `len(A)` donne le nombre de lignes de `A`.
Et pour le nombre de colonnes : il suffit de compter la longueur d'une ligne. C'est donc...

Comment accéder à un élément d'une matrice ?
Il suffit de connaitre son indice `i` de ligne,
de lire la ligne `i` : `A[i]`
et sur cette ligne, de lire l'élément d'indice `k`.
On fait donc appel à `A[i][k]` [1].

1. double crochet surprenant la première fois, usuel ensuite, et je vous devine prenant l'habitude de taper tout de suite des `A[ ][ ]` que vous revenez ensuite compléter avec les indices

Exercices : par exemple, en taille 2, que représente `A[0][0]*A[1][1]-A[0][1]*A[1][0]` ?

Exercice : en mathématiques, un objet usuel et pratique est la trace d'une matrice, somme des éléments de sa diagonale principale $a_1^1 + \ldots + a_n^n$.
Écrivez une procédure (qui ne servira pas ensuite) qui prend en entrée une matrice (supposée carrée, mais pas forcément de taille 2) et qui retourne sa trace.

Attention, l'habitude mathématique fait aller les indices de 1 *à* n*, celle de* **`Python`** *les fait aller de* **`0`** *à* **`n`** *(exclu), ou plutôt « dans* **`range(n)`** *».*

Et pour créer vous même une matrice. Il y a certes l'option « je tape des crochets et des termes ». Gentil.
Mais si je vous dis que je veux la matrice de taille `5` sur `5` dont le terme général est `0`. Que faites vous ? Pas grave, vous tapez...
Non. Une ligne de 5 zéros, c'est `[0]*5`
ou même `[0 for k in range(5)]`
(je préfère la seconde par sécurité)
Et cinq lignes de 5 zéros, c'est `[[0]*5 for i in range(5)]`
ou même `[[0 for k in range(5] for i in range(5)]`
Ne tentez pas **`[[0]*5]*5`***, c'est la porte ouverte à des erreurs que je vous expliquerai en présentiel.*
Plus généralement, pour créer une matrice de taille **`n`** sur **`n`**, le « constructeur » est du type `[[.... for k in range(n)] for i in range(n)]` dont vous aurez très vite l'habitude.

Qui de
```
[[1 for k in range(6)] for i in range(5)]
[[1 for k in range(5)] for i in range(6)]
[[1 for k in range(5)] for k in range(6)]
[1 for k in range(6) for i in range(5)]
```
va donner la matrice rectangulaire $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

Qui va donner une erreur ?
Écrivez une procédure qui prend en entrée n et crée la matrice de taille n sur n de la table de multiplication $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{pmatrix}$ en indexation de matheux.
Je veux créer une matrice de taille n sur n à coefficients pris aléatoirement entre `0` et `R-1`.
Que pensez vous de

```
from random import *
```

```
def Alea(n, R) :
....terme = randrange(R)
....M=[[terme for k in range(n)] for i in
range(n)]
....return(M)
from random import *
```

```
def Alea(n, R) :
....M=[[randrange(R) for k in range(n)] for
i in range(n)]
....return(M)
```

```
def Alea(n, R) :
....M=[ ]
....for i in range(n) :
........Ligne = [ ]
........for k in range(n) :
............Ligne.append(randrange(R))
........M.append(Ligne)
....return(M)
```

```
def Alea(n, R) :
....M=[ ]
....Ligne = [ ]
....for i in range(n) :
........for k in range(n) :
............Ligne.append(randrange(R))
........M.append(Ligne)
....return(M)
```

```
def Alea(n, R) :
....M=[ ]
....for i in range(n) :
........Ligne = [ ]
........for k in range(n) :
............Ligne.append(randrange(R))
....M.append(Ligne)
....return(M)
```

En tout cas, gardez celle qui vous plait, pour pouvoir créer des matrices aléatoires.

Et que font

```
def Attila(n, a=1) :
....M=[[a for k in range n] for i in
range(n)]
...return(M)
```

```
def Unite(n) :
....M=[[0 for k in range n] for i in
range(n)]
....for i in range(n) :
........M[i][i] = 1
...return(M)
```

Pouvez vous raccourcir la seconde

Passons à la procédure de somme industrielle.
On prend en entrée deux matrices (qu'on va supposer carrées de même format `len(A)`), et il faut calculer leur « somme industrielle).
Pour vous simplifier la vie, en `Python`, il existe une fonction directe `Max` qui retourne le maximum d'un t-uple : `Max(1, 5)` donne `5` et `Max(3, 6, 7, 4)` donne `6`. Alors qu'en maths, Max donne juste un élève de Spé qui est hyper heureux d'être en Prépa.
Vous pouvez faire tenir la réponse en une ligne. Ou plus suivant ce que vous aimez : concision ou lisibilité.

```
def Addition(A, B) :
....n = len(A)
....M = completez
```

Et maintenant, la multiplication...
Il nous faut un script qui prend en entrée deux matrices A et B (là encore supposées de même taille) et qui calcule leur produit industriel $A \otimes B$.
Il commencera par `def Produit(A, B) :`

```
....n = len(A)
```

et ensuite, vous pourrez soit construire la matrice ligne à ligne comme indiqué ci dessus,
soit construire une matrice nulle de bon format, puis en remplir les cases.
Vous aurez donc besoin de deux boucles `for` imbriquées

```
for i in range(n) :
....for k in range(n) :
............un calcul
............M[i][k] = resultat
```

Mais le calcul lui même aura besoin d'une boucle sur `j`.
Il y a donc trois boucles imbriquées.
C'est ce qui permet de dire que votre calcule sera de complexité de l'ordre de n^3 où n est le format des matrices.

Vérifiez si votre programme est correct, par exemple en utilisant

```
A = Alea(4, 1)
B = Alea(4, 1)
P = Produit(A, B)
Affiche(A)
Affiche(B)
Affiche(P)
```

Tiens, au fait, quelle différence pour les deux premières lignes avec `A, B = Alea(4, 1), Alea(4, 1)`

```
A=B=Alea(4,1)
```

Faites des tests pour savoir si il existe un élément neutre pour la multiplication industrielle.

Calculez les puissances successives de la matrice Attila [2].

Je vous propose, pour créer une matrice diagonale et calculer ses puissances :

2. au fait, pourquoi Attila ? parce qu'elle a des Huns partout

```
A = [[0 for k in range(5)] for i in range(5)]
for k in range(5) :
....A[k][k] = randrange(4)
....A[k][4-k] = randrange(4)
```
Tiens, sans l'afficher, quelle forme va avoir la matrice A ?

`P = Produit(A, A)`
`for k in range(6) :`
`....Affiche(P)`
`....P = Produit(A, A)`
	`P = Produit(A, A)`
	`for k in range(6) :`
	`....Affiche(P)`
	`....P = Produit(P, A)`
		`P = Produit(A, A)`
		`for k in range(6) :`
		`....Affiche(P)`
		`....P = Produit(P,P)`

`P = A`
`for k in range(6) :`
`....Affiche(P)`
`....P = Produit(P, A)`
	`for k in range(6) :`
	`....P = Produit(P, A)`
	`....Affiche(P)`
		`P = Produit(A, A)`
		`for k in range(6) :`
		`....P = Produit(P,A)`
		`Affiche(P)`

Lequel (lesquels) est correct ?

Si je dois calculer A^n, que dois je faire ? [3]

```
def Puissance(A, n) :
....un truc
....for k in range(n) : ou n-1
........calcul
....return(....)
```

```
def Puissance(A, n) :
....if n==1 :
........return(A)
....if n%2 == 0 :
........M = Puissance(A, n//2)
........return(Produit(M, M))
....else :
........P = Puissance(A,n-1)
........return(Produit(P,A))
```

Que pensez vous du script
Est il vrai qu'il calcule A^n par exemple pour $n = 5$
$n = 7$
$n = 11$
$n = 12$
$n = 1024$
$n = 2020$
Et pour chacun de ces entiers, combien de fois sollicite-t-il la procédure `Produit` ?

`https://www.ensta-bretagne.fr/jaulin/mastersds_sandwichs.pdf`

3. Évidemment, une fois de plus, c'est vous de compléter, on ne me sollicite pas en disant « monsieur, le compilateur Python me dit « calcul is not defined ».