ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ
ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⴰⵏⴰⵎⵓⵔ
ⴰ ⵙⵓⴽⵙ ⴰⵎⴰⵡⴰⵍ ⴰ ⵙⵓⵏⵖⵓ

المملكة المغربية
وزارة التعليم العالي
والبحث العلمي والابتكار

ROYAUME DU MAROC
Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et de l'Innovation

المدرسة الوطنية العليا للمعادن بالرباط
ⵜⵉⵎⵙⵙⵓⵜⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵣⵎⵎⴰⵎⵜ ⵖ ⵔⵕⴱⴰⵟ
ECOLE NATIONALE SUPERIEURE DES MINES DE RABAT

وزارة الانتقال الطاقي
Ministère de la Transition Énergétique
والتنمية المستدامة
et du Développement Durable
Département de la Transition Energetique قطاع الانتقال الطاقي

MINES-RABAT

CNC 2022

Concours National Commun

d'Admission dans les Établissements de Formation d'Ingénieurs et Établissements Assimilés

Épreuve d'**INFORMATIQUE**

Filière : **MP**

Durée **2 heures**

Cette épreuve comporte 12 **pages au format A4**, en plus de la page de garde

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre

Remarques générales :

- ✓ Cette épreuve est composée d'un exercice et de trois parties tous indépendants ;
- ✓ Toutes les instructions et les fonctions demandées seront écrites en Python ;
- ✓ Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- ✓ Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈

Important : Le candidat doit impérativement commencer par traiter toutes les questions de l'exercice ci-dessous, et écrire les réponses dans les premières pages du cahier de réponses.

Exercice : (4 points)

On considère une liste X qui contient des nombres réels.

Q1- Écrire la fonction ***affiche (X)*** qui parcourt et affiche les éléments de la liste X.

Q2- Écrire la fonction ***somme (X)*** qui retourne la somme des éléments de de la liste X, sans utiliser la fonction prédéfinie sum().

Q3- Déterminer la complexité de la fonction ***somme (X)***, avec justification.

Q4- Écrire la fonction ***moyenne (X)*** qui retourne la valeur de la moyenne m de la liste X :

$$m = \frac{\sum x_i}{n}$$

Q5- Écrire la fonction ***compte (X)*** qui retourne le compte des éléments de la liste X, qui sont strictement supérieurs à la moyenne m de la liste X :

L'**écart type** de la liste X est une mesure de la dispersion des nombres autour de la moyenne des éléments de X. Il se note en général avec la lettre grecque σ :

$$\sigma = \sqrt{\frac{\sum(x_i - m)^2}{n}}$$

- x_i représente les éléments de X
- m est la moyenne des éléments de X
- n est la longueur de X.

Q6- Écrire la fonction ***sigma (X)*** qui retourne la valeur de la somme suivante :

$$\sum(x_i - m)^2$$

Q7- Écrire la fonction ***ecart_type (X)*** qui retourne la valeur de l'écart type de X.

Partie I : Base de données et langage SQL

Internationaux de tennis

On considère une base de données qui permet la gestion des joueurs de tennis, ayant gagné au moins un tournoi international de tennis simple. Cette base de données est composée de **3** tables : **'Joueur'**, **'Gagne'** et **'Tournoi'**.

Joueur	
id	*(entier)*
nomJr	*(texte)*
genre	*(texte)*
nationalité	*(texte)*

Gagne	
idJr	*(entier)*
codTr	*(entier)*
année	*(entier)*
prime	*(réel)*

Tournoi	
code	*(entier)*
nomTr	*(texte)*
catégorie	*(texte)*
pays	*(texte)*

a- Structure de la table 'Joueur'

La table **'Joueur'** contient des informations sur les joueurs ayant gagné au moins un tournois international de tennis simple : Le champ **id** est la clé primaire, ce champ contient un entier unique pour identifier chaque joueur. Le champ **nomJr** contient le nom complet de chaque joueur. Le champ **genre** contient **'F'** pour féminin, ou **'M'** pour masculin. Le champ **nationalité** contient le pays d'origine de chaque joueur.

Exemples :

id	nomJr	genre	nationalité
2154	Roger Federer	M	Suisse
3432	Serena Williams	F	États Unis
2691	Novak Djokovic	M	Serbie
2374	Rafael Nadal	M	Espagne
...	...	...	...

b- Structure de la table 'Tournoi'

La table **'Tournoi'** contient des informations sur les tournois internationaux de tennis simples. Le champ **code** est la clé primaire, ce champ contient un entier unique pour chaque tournoi. Les champs **nomTr** et **catégorie** contiennent le nom et la catégorie de chaque tournoi. Le champ **pays** contient le pays dans lequel se déroule chaque tournoi.

Exemples :

code	nomTr	catégorie	pays
23	Roland-Garros	Grand Chelem	France
174	Doha	ATP 250	Qatar
10	Open d'Australie	Grand Chelem	Australie
59	Sydney International	ATP 250	Australie
85	Rotterdam	ATP 500	Pays-Bas
96	Masters d'Indiana Wells	Masters 1000	États Unis
...	...	...	...

c- Structure de la table 'Gagne'

La table **'Gagne'** contient des informations sur les joueurs et les tournois gagnés par chaque joueur. Les champs **idJr** et **codTr** sont deux clés étrangères qui font respectivement référence aux clés primaires **id** et **code**. Le champ **année** contient l'année de chaque tournoi. Chaque tournoi est organisé une seule fois chaque année. Le champ **prime** contient le montant reçu par le joueur gagnant de chaque tournoi, et il est exprimé en million de dollars. La prime d'un tournoi peut changer au cours des années.

Exemples :

idJr	codTr	année	prime
2374	23	2019	2.30
2374	23	2020	1.60
2374	23	2021	1.40
3432	85	2018	2.50
2691	10	2020	4.12
2691	10	2021	2.75
...	...	...	...

Q.1 – Rédiger une requête SQL, qui effectue dans la table '**Tournoi**', la modification suivante :

Modifier les deux catégories '**WTA Premier Mandatory**' et '**WTA Premier 5**' par la catégorie '**WTA 1000**'

Q.2 – Rédiger en algèbre relationnelle, une requête qui permet d'obtenir les pays qui organisent des tournois dans les deux catégories **'Masters 1000'** et **'ATP 500'**.

Q.3 – Écrire la requête **Q.2** en langage SQL.

Q.4 – Rédiger une requête SQL, qui permet d'obtenir les différentes catégories et le compte des tournois dans chaque catégorie, triés dans l'ordre alphabétique des catégories.

Q.5 – Rédiger une requête SQL, qui permet d'obtenir les noms et genres des joueurs, la plus grande prime et la plus petite prime reçues par chaque joueur, tels que la somme totale des primes reçue par chaque joueur est supérieure à **75** millions de dollars, triés dans l'ordre décroissant de la moyenne des primes reçues par chaque joueur.

Q.6 – Les tournois de la catégorie **Grand-Chelem** sont 4 : **'Open d'Australie'**, **'Roland-Garros'**, **'Wimbledon'** et **'US Open'**.

Rédiger une requête SQL, qui permet d'obtenir les années, les noms, les genres et les nationalités des joueurs qui ont gagné les **4** tournois du **Grand-Chelem** la même année, triés dans l'ordre décroissant des années.

Q.7 – Rédiger une requête SQL, qui permet d'obtenir les codes des tournois et les primes de l'année **2020** et les primes de l'année **2021** de chaque tournoi.

Exemples :

Code tournoi	prim2020	prim2021
23	1.60	1.40
85	0.98	1.20
...	...	...

Partie II : Calcul numérique

Décomposition LU

Algorithme de 'DOOLITTLE'

Dans cette partie, on suppose que le module **numpy** est importé :

import numpy as np

En algèbre linéaire, la **décomposition LU** est une méthode de décomposition d'une matrice comme produit de deux matrices ***L*** et ***U***, telles que :

- ***L*** est une matrice triangulaire inférieure, formée de **1** sur sa diagonale,
- ***U*** est une matrice triangulaire supérieure.

Cette décomposition est utilisée pour résoudre des systèmes d'équations linéaires, pour calculer le déterminant d'une matrice, … .

Rappel : Soit ***M*** une matrice carrée de dimension **n**. La matrice ***M*** admet une décomposition **LU** unique si et seulement si les déterminants des sous-matrices de ***M***, d'ordre ***k*** tel que $k \in [\![1, n-1]\!]$, sont tous non nuls.

NB : Dans la suite de cette partie, on suppose que la matrice M admet une décomposition LU unique.

Algorithme de 'Doolittle'

L'algorithme de **Doolittle** est un algorithme qui décompose une matrice carrée, qui admet une décomposition LU unique, en deux matrices : la matrice ***L*** triangulaire inférieure avec des **1** dans sa diagonale, et la matrice **U** triangulaire supérieure :

Entrée : **M** matrice carrée d'ordre **n**, qui admet une décomposition LU unique.

Créer la matrice identité **L** d'ordre **n**

Créer la matrice carrée **U** d'ordre **n**, remplie par des **0**

Pour **i = 0** à **n-1** faire :

Pour **j = i** à **n-1** faire :

$$U_{i,j} \leftarrow M_{i,j} - \sum_{k=0}^{i-1}(L_{i,k} * U_{k,j})$$

Fin pour j

Pour **j = i+1** à **n-1** faire :

$$L_{j,i} \leftarrow (M_{j,i} - \sum_{k=0}^{i-1}(L_{j,k} * U_{k,i})) / U_{i,i}$$

Fin pour j

Fin pour i

Sortie : **L**, **U**

Q.1- Écrire la fonction ***matrice_zeros (n)*** qui reçoit en paramètre un entier strictement positif **n** et qui retourne la matrice d'ordre **n** (**n** lignes et **n** colonnes), remplie par des zéros.

Q.2- Écrire la fonction ***matrice_id (n)*** qui reçoit en paramètre un entier strictement positif **n** et qui retourne la matrice identité d'ordre **n** (**n** lignes et **n** colonnes).

Q.3- Écrire la fonction ***sigma (L, U, i, j)*** qui reçoit en paramètres deux matrices carrées **L** et **U** de même dimension, un entier positif **i** qui représente l'indice d'une ligne dans **L**, et un entier positif **j** qui représente l'indice d'une colonne dans **U**. La fonction retourne la valeur de la somme suivante :

$$\sum_{k=0}^{i-1} (L_{i,k} * U_{k,j})$$

Q.4- Écrire la fonction ***doolittle (M)*** qui reçoit en paramètre une matrice carrée **M** admettant une décomposition **LU**. En utilisant l'algorithme de Doolittle, la fonction effectue la décomposition **LU** de la matrice **M**, et retourne les matrices **L** et **U**.

Exemple :

On considère la matrice carrée **M** (qui admet une décomposition LU) suivante :

$$\begin{vmatrix} 1. & 2. & -1. & 4 \\ -2. & -1. & 1. & -1. \\ -1. & 1. & 2. & -3. \\ 3. & -5. & -2. & -3. \end{vmatrix}$$

La fonction **doolittle (M)** retourne les matrices **L** et **U** suivantes :

$$\begin{vmatrix} 1. & 0. & 0. & 0. \\ -2. & 1. & 0. & 0. \\ -1. & 1 & 1 & 0. \\ 3. & -3.67 & -1.33 & 1. \end{vmatrix} \qquad \begin{vmatrix} 1. & 2. & -1. & 4. \\ 0. & 3. & -1. & 7. \\ 0. & 0. & 2. & -6. \\ 0. & 0. & 0. & 2.67 \end{vmatrix}$$

Résolution d'un système d'équations linéaires par décomposition LU :

On considère le système d'équations linéaires : ***M * X = Y***, sachant que :

- **M** est une matrice carrée donnée, qui admet une décomposition LU ;
- ***Y*** est un vecteur donné ;
- ***X*** est le vecteur inconnu, qui représente la solution du système.

Exemple :

$$\begin{vmatrix} 1. & 2. & -1. & 4 \\ -2. & -1. & 1. & -1. \\ -1. & 1. & 2. & -3. \\ 3. & -5. & -2. & -3. \end{vmatrix} * X = \begin{vmatrix} 1. \\ -1 \\ 0. \\ 2. \end{vmatrix}$$

En réalisant la décomposition **LU** de la matrice **M**, on obtient la matrice triangulaire inférieure **L** avec des **1** dans sa diagonale, et la matrice triangulaire supérieure **U**, telles que : ***M = L * U***.

Ainsi, résoudre le système initial ***M * X = Y,*** revient à résoudre du système ***L * U * X = Y***

Donc, pour résoudre ce dernier système, on doit résoudre successivement deux systèmes triangulaires :

i. On commence par résoudre le système triangulaire inférieur : ***L * Z = Y***, à fin de trouver le vecteur ***Z*** ;
ii. Ensuite, on résout le système triangulaire supérieur : ***U * X = Z***, à fin de trouver le vecteur ***X*** ;

NB : Dans la suite de cette partie, on ne doit pas utiliser la méthode prédéfinie solve().

Q.5- Écrire la fonction ***descente (L, Y)*** qui reçoit en paramètres la matrice triangulaire inférieure **L**, et le vecteur **Y**. La fonction retourne le vecteur ***Z***, solution du système triangulaire inférieur : ***L * Z = Y***.

Pour calculer les éléments du vecteur **Z**, on peut utiliser la formule suivante :

$$Z_i = Y_i - \sum_{j=0}^{i-1} (L_{i,j} * Z_j)$$

Exemple :

La fonction ***descente (L, Y)*** retourne le vecteur ***Z*** suivant : **[1. 1. 0. 2.67]**

Q.6- Écrire la fonction ***remonte (U, Z)*** qui reçoit en paramètres la matrice triangulaire supérieure **U**, et le vecteur **Z**. La fonction retourne le vecteur ***X***, solution du système triangulaire supérieur : ***U * X = Z***

Pour calculer les éléments du vecteur **X**, on peut utiliser la formule suivante :

$$X_i = (Z_i - \sum_{j=n-1}^{i+1} (U_{i,j} * X_j))/U_{i,i}$$

Exemple :

La fonction ***remonte (U, Z)*** retourne le vecteur ***X*** suivant : **[2. -1. 3. 1.]**

Partie III : Problème

Plus Longue Sous Séquence Commune

Une **séquence** est un ensemble d'éléments non vide, fini et ordonné. Pour représenter une séquence, on utilisera une liste. Si une séquence contient n éléments, alors ses éléments sont indicés de 0 à $n-1$.

Généralement, on représentera une séquence de n éléments par la liste $X = [\, x_0, x_1, x_2, \ldots, x_{n-2}, x_{n-1} \,]$.

Pour un entier $k \in [\![1; n]\!]$, on notera X_k pour désigner la séquence des k premiers éléments de la séquence X. Ainsi, la séquence X_k sera représentée par la liste $[\, x_0, x_1, x_2, \ldots, x_{k-1} \,]$

Dans la suite cette partie, on suppose que les éléments des séquences sont tous des entiers positifs.

Exemple :

La liste X = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]** représente une séquence de **10** éléments.

- ✓ La séquence X_1 est représentée par la liste **[6]**
- ✓ La séquence X_4 est représentée par la liste **[6, 23, 20, 18]**
- ✓ La séquence X_7 est représentée par la liste **[6, 23, 20, 18, 6, 54, 3]**
- ✓ ...
- ✓ La séquence X_{10} est représentée par la liste **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

1- Séquence triée

Q.1- Écrire la fonction de ***tri_sequence (S)*** qui reçoit en paramètre une séquence S, et qui trie les éléments de S dans l'ordre croissant.

Exemple :

On considère la séquence S = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

Après l'appel de la fonction **tri_sequence (S),** on obtient la séquence : **[3, 6, 6, 6, 13, 18, 18, 20, 23, 54]**

2- Sous-séquence d'une séquence

On considère deux séquences X et Y. On dit que X est une sous-séquence de Y, si X est composée des éléments qui apparaissent dans Y, dans le même ordre. Autrement dit, la sous-séquence X est obtenue en supprimant, dans Y, un certain nombre d'éléments (éventuellement aucun élément).

Formellement, étant données deux séquences $X = [\, x_0, x_1, x_2, \ldots, x_{n-1} \,]$ et $Y = [\, y_0, y_1, y_2, \ldots, y_{p-1} \,]$. On dit que X est une sous-séquence de Y, s'il existe une suite croissante d'indices $(i_1, i_2, \ldots, i_k)$ de Y, telle que pour tout $j = 1, 2, \ldots, k$, on a : $X_j = Y_{i_j}$

Exemple :

On considère la séquence X = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

- ✓ **[6, 20, 18, 13]** est une sous-séquence de X
- ✓ **[23, 18, 6, 54, 6, 18]** est une sous-séquence de X
- ✓ **[15, 18, 6, 20]** n'est pas une sous-séquence de X

Q.2- Écrire la fonction ***sous_sequence (X, Y)*** qui reçoit en paramètres deux séquences X et Y, et qui retourne **True**, si X est une sous-séquence de Y, sinon, la fonction retourne **False**.

Plus longue sous-séquence commune :

Le problème de la **plus longue sous-séquence commune** (**PLSSC**) à deux séquences X et Y, consiste à trouver une sous-séquence de X et de Y, et qu'elle soit de longueur maximale.

3- Taille de la plus longue sous-séquence commune

Étant données deux séquences $X = [\, x_0, x_1, x_2, \dots, x_{n-1} \,]$ et $Y = [\, y_0, y_1, y_2, \dots, y_{p-1} \,]$. Pour calculer la taille de la PLSSC à X et Y, il est possible de ramener ce problème à un calcul entre deux séquences de taille inférieure grâce à la relation de récurrence suivante :

si $x_{n-1} = y_{p-1}$ alors,

La taille de la PLSSC à X et $Y = 1 +$ taille de la PLSSC à X_{n-1} et Y_{p-1}

si $x_{n-1} \neq y_{p-1}$ alors,

La taille de la PLSSC à X et $Y = \max$ (taille de la PLSSC à X_{n-1} et Y_p , taille de la PLSSC à X_n et Y_{p-1})

Q.3 - Écrire la fonction ***taille_PLSSC (X, Y)*** qui reçoit en paramètres deux séquences X et Y, et qui retourne la taille de la PLSSC à X et Y, en utilisant la relation de récurrence citée ci-dessus.

Exemples :

On considère la séquence X **= [6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

- ✓ La fonction **taille_PLSSC** (X, [20, **23,** 10, **18, 6,** 25, **3,** 1, **6,** 8, 2]) retourne **5**
- ✓ La fonction **taille_PLSSC** (X, **[23, 54, 6, 18]**) retourne **4**
- ✓ La fonction **taille_PLSSC** (X, [28, 35, 12, 8, 17]) retourne **0**

4- Recherche d'une PLSSC

Une approche naïve est possible. On peut par exemple construire toutes les sous-séquences de X, puis rechercher parmi ces sous-séquences, celles qui sont aussi sous-séquences de Y, et de longueur maximale.

Mais, si la séquence X est composée de **n** éléments, alors par dénombrement, le nombre de sous-séquences de X est 2^n. Les essayer toutes pour trouver la plus longue qui soit une sous-séquence de Y a donc une complexité exponentielle, ce qui n'est pas souhaitable en pratique.

Le problème de la recherche d'une PLSSC entre deux séquences, peut être résolu en utilisant la programmation dynamique.

La programmation dynamique est un paradigme algorithmique qui résout un problème complexe en le divisant en sous-problèmes, et stocke les résultats des sous-problèmes pour éviter de recalculer à nouveau les mêmes résultats.

Étant données deux séquences $X = [\, x_0, x_1, x_2, \dots, x_{n-1} \,]$ de taille ***n*** et $Y = [\, y_0, y_1, y_2, \dots, y_{p-1} \,]$ de taille ***p***, on commence par créer une matrice ***M*** de ***n+1*** lignes et ***p+1*** colonnes.

Dans cette matrice ***M***, chaque case $M_{i,j}$ est destinée à contenir la longueur de la plus longue sous séquence commune entre X_i et Y_j.

On peut alors calculer de proche en proche la valeur $M_{i,j}$ pour chaque couple d'indices **i** et **j**, en utilisant l'algorithme **(1)** suivante :

$$(1)\quad \begin{cases} si\ \ i=0\ \ ou\ \ j=0\ \ \ alors\ \ \ M_{i,j}=0 \\ si\ \ i>0\ \ et\ \ j>0\ \ et\ \ x_{i-1}=y_{j-1}\ \ \ alors\ \ \ M_{i,j}=1+M_{i-1,j-1} \\ si\ \ i>0\ \ et\ \ j>0\ \ et\ \ x_{i-1}\neq y_{j-1}\ \ \ alors\ \ \ M_{i,j}=max(M_{i,j-1},M_{i-1,j}) \end{cases}$$

Q.4.a- Écrire la fonction ***matrice_nulle (n, p)*** qui reçoit en paramètres deux entiers ***n*** et ***p*** strictement positifs. La fonction retourne une liste de listes ***M***, qui représente la matrice composée de ***n+1*** lignes et ***p+1*** colonnes, remplie par des zéros.

Exemple :

La fonction **matrice_nulle (4, 7)** retourne la matrice ***M*** (5 lignes et 8 colonnes) suivante :

```
[ [ 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0 ] ]
```

Q.4.b- Écrire la fonction ***matrice_tailles*** (X, Y) qui reçoit en paramètres deux séquences X et Y. En utilisant l'algorithme **(1)** précédent, la fonction retourne la matrice ***M***, qui contient les longueurs des sous séquences communes à X et Y.

Exemple :

X = **[20, 23, 75, 18, 54, 6, 3, 18]** et Y = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

La fonction **matrice_tailles** (X, Y) retourne la matrice suivante :

		6	23	20	18	6	54	3	13	6	18
	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	1	1	1	1	1	1	1
23	0	0	1	1	1	1	1	1	1	1	1
75	0	0	1	1	1	1	1	1	1	1	1
18	0	0	1	1	2	2	2	2	2	2	2
54	0	0	1	1	2	2	3	3	3	3	3
6	0	1	1	1	2	3	3	3	3	4	4
3	0	1	1	1	2	3	3	4	4	4	4
18	0	1	1	1	2	3	3	4	4	4	5

Q.4.c- Écrire la fonction ***taille_plssc*** (X, Y) qui reçoit en paramètres deux séquences X et Y, et qui retourne la taille de la PLSSC à X et Y, en utilisant l'algorithme **(1)** précédent.

Q4.d- Déterminer la complexité de la fonction ***taille_plssc*** (X, Y), avec justification.

5- Construction d'une PLSSC

L'algorithme **(1)** précédent permet de calculer de proche en proche les cases de la matrice ***M***, et cette matrice contient les longueurs des sous séquences communes aux deux séquences principales.

Réciproquement, pour reconstituer une PLSSC, il suffit de construire un chemin dans cette matrice ***M***. Pour cela, on effectuera l'algorithme **(2)** suivant :

a. On commence par créer une séquence vide ***S*** ;

b. Ensuite, dans la matrice ***M***, on effectue un parcours depuis la case inférieure droite (dernière ligne, et dernière colonne) de ***M***, pour rejoindre la ligne **0** ou la colonne **0** de ***M***, suivant la règle de parcours suivante :

Depuis une case $M_{i,j}$:

b1. Si $x_{i-1} = y_{j-1}$ alors, on ajoute l'élément x_{i-1} (ou y_{j-1}) au début de ***S***, et on passe à la case $M_{i-1,j-1}$;

b2. Si $x_{i-1} \neq y_{j-1}$ et $M_{i,j-1} = M_{i-1,j}$ alors, on passe à la case $M_{i-1,j}$ **ou bien** à la case $M_{i,j-1}$;

b3. Si $x_{i-1} \neq y_{j-1}$ et $M_{i,j-1} > M_{i-1,j}$ alors, on passe à la case $M_{i,j-1}$;

b4. Si $x_{i-1} \neq y_{j-1}$ et $M_{i,j-1} < M_{i-1,j}$ alors, on passe à la case $M_{i-1,j}$.

Exemple :

		6	23	20	18	6	54	3	13	6	18
	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	1	1	1	1	1	1	1
23	0	0	1	1	1	1	1	1	1	1	1
75	0	0	1	1	1	1	1	1	1	1	1
18	0	0	1	1	2	2	2	2	2	2	2
54	0	0	1	1	2	2	3	3	3	3	3
6	0	1	1	1	2	3	3	3	3	4	4
3	0	1	1	1	2	3	3	4	4	4	4
18	0	1	1	1	2	3	3	4	4	4	5

case départ

Depuis la case de départ, et en suivant le chemin coloré dans la matrice ***M***, on déduit une PLSSC :

S = [20, 18, 54, 6, 18]

Q.5- Écrire la fonction ***plssc*** (X, Y) qui reçoit en paramètres deux séquences X et Y, et qui retourne une PLSSC à X et Y.

Exemple :

X = **[20, 23, 75, 18, 54, 6, 3, 18]** et Y = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

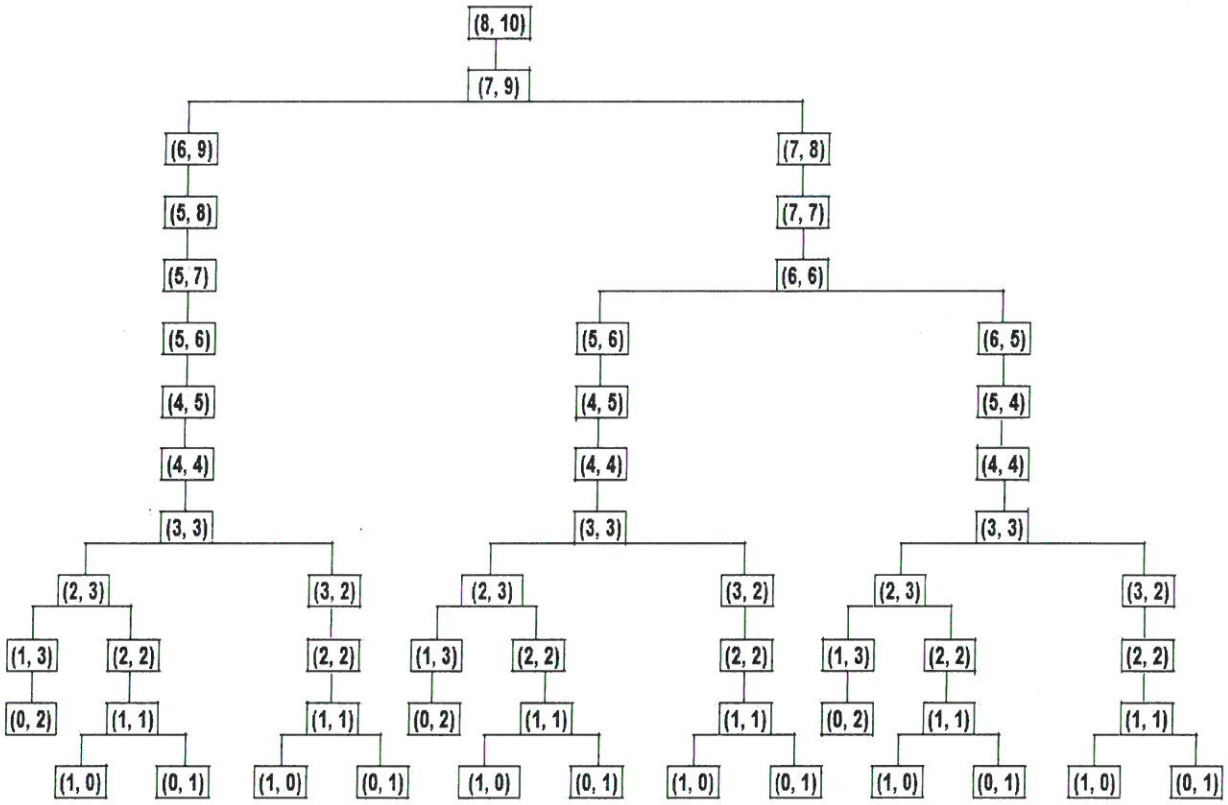
La fonction **plssc** (X, Y) retourne la séquence : **S = [20, 18, 54, 6, 18]**

6- Recherche de toutes les PLSSC

La matrice ***M*** contient les longueurs des sous séquences communes. À partir de la matrice ***M***, on peut trouver toutes les PLSSC. Pour cela, on doit construire tous les chemins possibles dans la matrice ***M***, depuis la case inférieure droite de ***M***, pour rejoindre la ligne ***0*** ou la colonne ***0*** de ***M***. Ainsi, dans l'étape **b3** de l'algorithme (2) précédent, on doit effectuer la modification suivante :

b3. Si $x_{i-1} \neq y_{j-1}$ et $M_{i,j-1} = M_{i-1,j}$ alors, on passe à la case $M_{i-1,j}$ **et** à la case $M_{i,j-1}$;

Par exemple, l'arbre suivant, montre tous les chemins qu'on peut construire, dans la matrice ***M*** de l'exemple précédent :



NB :

- ✓ Chaque élément de l'arbre est un tuple qui contient de la ligne et la colonne d'une case dans la matrice ***M*** ;
- ✓ Dans l'arbre, plusieurs chemins peuvent aboutir à une même PLSSC.

Q.6- Écrire la fonction ***lcs (X, Y)*** qui reçoit en paramètres deux séquences X et Y, et qui retourne une liste qui contient toutes les PLSSC à X et Y, sans doublons.

Exemple :

X = **[20, 23, 75, 18, 54, 6, 3, 18]** et Y = **[6, 23, 20, 18, 6, 54, 3, 13, 6, 18]**

La fonction **lcs (X, Y)** retourne la liste de toutes les PLSSC à X et Y :

[[20, 18, 54, 6, 18] , [23, 18, 54, 6, 18] , [20, 18, 54, 3, 18] , [23, 18, 54, 3, 18] , [20, 18, 6, 3, 18] , [23, 18, 6, 3, 18]]

7- Plus longue sous-séquence croissante

La recherche d'une **plus longue sous-séquence croissante** dans une séquence, est un problème classique en algorithmique. Ce problème est lié à celui de la recherche d'une PLSSC à deux séquences.

Q.7- Écrire la fonction ***plss_croissantes (S)*** qui reçoit en paramètre une séquence S et qui retourne toutes les plus longues sous-séquences croissantes dans S.

Exemple :

S = **[12, 10, 9, 34, 25, 28, 30, 36, 47, 42]**

La fonction **plss_croissantes** (S) retourne la liste de toutes les plus longues sous-séquences croissantes dans S :

[[12, 25, 28, 30, 36, 47] , [10, 25, 28, 30, 36, 47] , [9, 25, 28, 30, 36, 47] , [12, 25, 28, 30, 36, 42] , [10, 25, 28, 30, 36, 42] , [9, 25, 28, 30, 36, 42]]

********** FIN **********