# DS 2

le 18/12/2025 – Durée : 2h

- L'usage de la calculatrice n'est pas autorisé.
- La clarté et la précision des raisonnements interviendront pour une grande part dans la notation. La présentation (en particulier les indentations) est également essentielle.
- Le résultat d'une question peut être admis afin de traiter une question suivante ; une fonction peut être utilisée dans la suite d'un exercice même si elle n'a pas été écrite.
- Le barème tiendra compte de la longueur du devoir.

## Exercice 1

On dispose d'une base de données composée de quatre tables (**annexe 1**) :
– La table `Continents` constituée des champs suivants :
  - *nom :* nom du continent (chaîne de caractères) ;
  - *surface :* surface du continent en kilomètres carrés (entier).

– La table `Pays` constituée des champs suivants :
  - *nom :* nom du pays (chaîne de caractères) ;
  - *code_pays :* identifiant unique du pays (chaîne de caractères) ;
  - *capitale :* capitale administrative du pays (chaîne de caractères) ;
  - *population :* nombre d'habitants du pays (entier).

– La table `Inclusion` constituée des champs suivants :
  - *code_pays :* identifiant unique du pays (chaîne de caractères) ;
  - *continent :* nom du continent auquel appartient le pays (chaîne de caractères).

– La table `Frontieres` constituée des champs suivants :
  - *code_pays1 :* identifiant unique du premier pays (chaîne de caractères) ;
  - *code_pays2 :* identifiant unique du second pays (chaîne de caractères) ;
  - *longueur :* longueur en kilomètres de la frontière entre *pays1* et *pays2* (nombre flottant strictement positif).

On a toujours *code_pays1<code_pays2* pour l'ordre lexicographique, ce qui assure que chaque frontière n'apparaît qu'une fois dans la table `Frontieres`.

**6/8**

Annexe 1

Table `Continents`

| nom | surface |
|---|---|
| 'Asie' | 44579000 |
| 'Europe' | 9938000 |
| ... | ... |

Table `Pays`

| nom | code_pays | capitale | population |
|---|---|---|---|
| 'Albanie' | 'AL' | 'Tirana' | 3088385 |
| 'Algerie' | 'DZ' | 'Alger' | 44487616 |
| ... | ... | ... | ... |

Table `Inclusion`

| code_pays | continent |
|---|---|
| 'AL' | 'Europe' |
| 'DZ' | 'Afrique' |
| ... | ... |

Table `Frontieres`

| code_pays1 | code_pays2 | longueur |
|---|---|---|
| 'AL' | 'GR' | 282.8 |
| 'AL' | 'MK' | 151.5 |
| ... | ... | ... |

1. Écrire une requête SQL permettant de récupérer le code du pays nommé 'France'.
2. Écrire une requête SQL permettant d'obtenir les noms des pays appartenant au continent 'Europe'.

Dans les deux questions suivantes, on admet que le code du pays nommé 'France' est 'F' et on peut utiliser librement cette information.

3. Écrire une requête SQL permettant de récupérer la longueur totale de la frontière du pays nommé 'France'.
4. Écrire une requête SQL permettant de récupérer les noms des pays frontaliers du pays nommé 'France'.
5. Écrire une requête SQL permettant de récupérer les noms des 10 pays les plus peuplés, par ordre décroissant.
6. Écrire une requête SQL permettant de récupérer le nom du troisième pays le plus peuplé.
7. Écrire une requête SQL permettant de récupérer les noms des continents avec pour chacun sa population. On supposera dans cette question que chaque pays appartient à un seul continent.

# Exercice 2

## I. Présentation

On considère le problème suivant. Une *grille* est constituée de $n$ lignes de $p$ disques, chacun d'entre eux portant un poids, ainsi que d'un point de départ (D) et d'un point d'arrivée (A). On appelle *chemin* une suite de disques telle qu'à un disque succède un autre situé dans la colonne suivante soit sur la même ligne, soit sur la ligne au-dessus, soit sur la ligne au-dessous (avec les restrictions évidentes si un disque est situé sur la première ou la dernière ligne et avec la particularité que D mène à tous les disques de la première colonne et que tous les disques de la dernière colonne mènent à A). Sur le graphe ci-dessous, un passage possible entre deux disques est représenté par une flèche (appelée aussi arc). Le *score* d'un chemin est la somme des poids des disques visités. Un *parcours* de la grille est un chemin partant de D et terminant en A. Le but du problème est la recherche, dans cette grille, d'un parcours de score maximal.

Une grille est représenté dans la figure 1, avec en grisé un parcours possible dans cette grille. Ce parcours grisé de la figure 1 a donc un score de 16.

Nous allons envisager plusieurs stratégies pour résoudre ce problème. Dans un premier temps, nous mettons en place les structures de données permettant de le traiter.

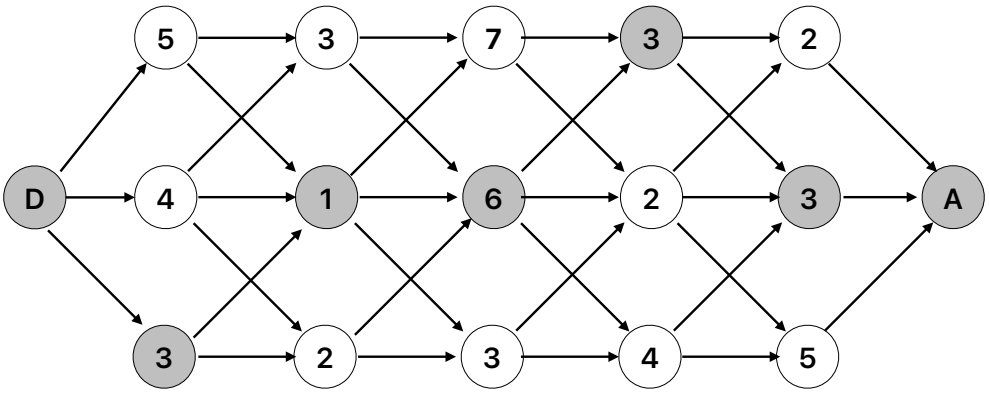


Figure 1 - Une grille avec un parcours

## II. Représentation des données du problème

Remarquons que pour représenter une grille on n'a pas besoin de représenter le point de départ et le point d'arrivée. On représente donc une grille par une liste de listes `Lgrille`, la liste `Lgrille[i]` étant la liste des nombres portés par les disques de la ligne `i` (la ligne du haut porte le numéro `0`). Par exemple, la grille de la figure 1 est représentée par la liste `[[5,3,7,3,2],[4,1,6,2,3],[3,2,3,4,5]]`.
On obtient ainsi un système de repérage dans une grille : un disque est repéré par sa ligne (on utilisera également le terme niveau) et par sa position sur la ligne (on utilisera également le terme colonne) ; on parlera du disque `(i,j)` pour parler du disque situé sur la ligne `i` et la colonne `j` ; le poids porté par le disque `(i,j)` est ainsi `Lgrille[i][j]`. Par exemple, dans la grille de la figure 1, le disque `(1,2)` porte le nombre 6 et le disque `(1,3)` porte le nombre 2.
Un parcours dans la grille est représenté par une liste `lParcours` de $p$ entiers : la liste des niveaux (lignes) des disques empruntés par le parcours à chaque colonne (on ne représentera pas D et A). Par exemple, le parcours grisé sur la figure 2 est représenté par la liste `[2,1,1,0,1]`.
Remarquons que si l'on avait voulu représenter un chemin quelconque, il aurait fallu une suite de couples (ou bien donner également la colonne de départ).

1. Dessiner la grille représentée par la liste de listes `[[3,4,5],[1,2,7],[4,9,4],[2,1,7]]` (y compris les disque de départ et d'arrivée).
   Griser (ou entourer les disques en une couleur différente) dans cette grille le parcours représenté par `[2,3,3]`.
   Calculer le score de ce chemin.

2. Écrire une fonction `scoreParcours(Lgrille ,lparcours)` qui prend en paramètre une grille `Lgrille` et un parcours `lparcours` (on ne demande pas de vérifier que ce parcours est valable) et qui renvoie le score de ce parcours.

3. On pourrait (en théorie) résoudre ce problème en calculant le score de chaque parcours et en déterminant le score maximal.
   En notant toujours $n$ le nombre de niveaux de la grille et $p$ son nombre de colonnes, déterminer une minoration du nombre de ces parcours (on pourra minorer le nombre d'arcs sortant d'un disque par 2).
   Quel type de complexité cette méthode exhaustive donnerait-elle ?

## III. Stratégie gloutonne

On va tout d'abord résoudre le problème de score maximal par une stratégie gloutonne. À chaque étape, on choisit le disque suivant qui porte le poids de plus élevé. Plus précisément, on détermine tout d'abord le disque de la première colonne portant le poids maximal (car tous ces disques sont accessibles à partir de $D$), puis à chaque étape on détermine le disque suivant accessible portant le poids maximal.

4. Compléter la fonction `scoreMaxGlouton(L)` qui prend en entrée une grille `L` (on a raccourci `Lgrille` en `L` pour simplifier l'écriture) et qui renvoie le score obtenu par cette stratégie dans cette grille ainsi que le parcours emprunté pour obtenir ce score.

```
def scoreMaxGlouton(L):
    n , p = len(L) , len(L[0])
    score , lparcours = 0 , []
    # On détermine la ligne contenant le poids max
        # dans la première colonne
    iMax , poidsMax = 0 , L[0][0]
    for i in range(..., ...):
        if ....                :
            iMax , poidsMax = ..., ...

    score  = ...
    lparcours.append(...  )
    # On parcourt ensuite la grille de gauche à droite
    # Si à l'étape j-1 on se trouve au niveau i,
    # à l'étape j on pourra se trouver au niveau i, i-1 ou i+1
    # (si ces niveaux sont valables)
    for j in range(1 ,p ):
        iMaxSuivant , poidsMax = iMax , L[iMax][j]
        if iMax - 1 >= 0 and L[iMax - 1][j] > poidsMax:
            iMaxSuivant , poidsMax = ...    , ...
        if ...                                      :
            ...
        iMax = iMaxSuivant
        score = ...
        lparcours.append(...   )
    return score , lparcours
```

5. Quel score obtient-on par cette stratégie dans l'exemple de la figure 1 ?
   Obtient-on le score maximal ?

6. Quel est, en fonction du nombre de niveaux $n$ et du nombre de colonnes $p$ de la grille, la complexité de cette méthode ?

## IV. Algorithme récursif naïf

On peut résoudre ce problème de manière récursive.
Ceci est basé sur la remarque suivante : si l'on connaît le score maximal obtenu en partant des trois (dans le cas général) disques accessibles à partir d'un disque `(i,j)`, il suffit, pour obtenir le score maximal à partir ce disque `(i,j)` de prendre le maximum de ces trois scores et de lui ajouter le poids du disque `(i,j)`.
On voit dans cette description que l'on doit généraliser le problème et chercher à déterminer le score maximal que l'on peut obtenir en partant d'un disque quelconque de la grille (et non plus uniquement des parcours) et en allant jusqu'au disque $A$.

Rappelons qu'un disque est repéré par son niveau $i$ ($0 \leq i < n$) et sa colonne $j$ ($0 \leq j < p$). On notera $M_{i,j}$ le score maximal que l'on peut obtenir en partant du disque situé en position $j$ sur la ligne $i$ et en allant jusqu'au disque $A$. On adopte également la notation suivante : le poids porté par le disque situé en position $j$ sur la ligne $i$ est noté $L_{i,j}$ (dans la liste de listes `L`, avec les conventions adoptées précédemment, il s'agit de `L[i][j]`).
Notre objectif final est donc de déterminer les valeurs de $M_{i,0}$ ($0 \leq i < n$), puis le maximum parmi ces valeurs.
Remarquons que si $j = p - 1$, alors $M_{i,j}$ est égal à $L_{i,j}$.

On illustre le principe du calcul récursif sur la figure suivante :

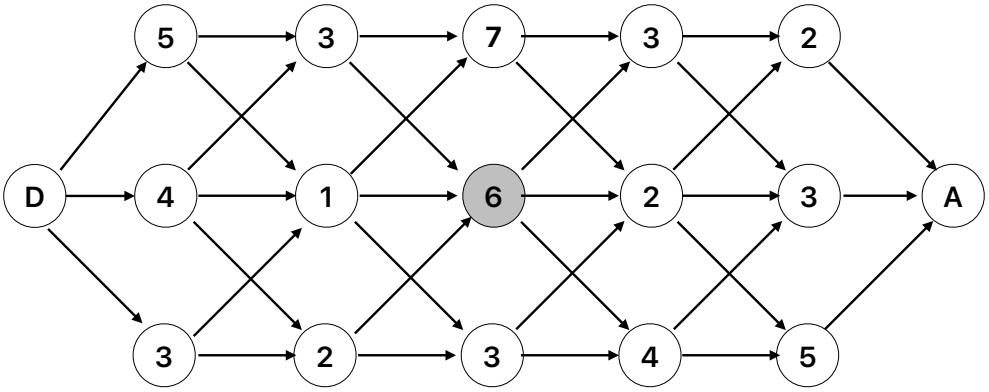


Figure 2

Le disque grisé est en position $(1, 2)$. Pour connaître le score maximal à partir de ce disque, il faut regarder les trois disques auxquels on peut accéder à partir de celui-ci. On observe que $M_{0,3} = 6$, $M_{1,3} = 7$ et $M_{2,3} = 9$. Le score maximal en partant de $(1, 2)$ sera donc obtenu en choisissant pour disque suivant le disque $(2, 3)$ ; le score obtenu est alors $M_{1,2} = L_{1,2} + M_{2,3} = 6 + 9 = 15$.

La remarque au début de ce paragraphe peut alors être reformulée de manière plus précise par les équations suivantes :
Pour $0 \leq i < n$, $M_{i,p-1} = L_{i,p-1}$.
Pour $0 \leq j < p - 1$

$$M_{i,j} = \begin{cases} L_{i,j} + \max\{M_{i,j+1}, M_{i+1,j+1}\} & \text{si } i = 0 \\ L_{i,j} + \max\{M_{i-1,j+1}, M_{i,j+1}, M_{i+1,j+1}\} & \text{si } 0 < i < n-1 \\ L_{i,j} + \max\{M_{i-1,j+1}, M_{i,j+1}\} & \text{si } i = n-1 \end{cases}$$

7. Compléter la fonction récursive `scoreMaxRecPart(L,i,j)` qui prend en paramètre une grille `L`, un niveau `i` et une colonne `j` (tous deux supposés valables) et qui renvoie, en utilisant la stratégie récursive décrite précédemment, le score maximal que l'on peut obtenir en partant du disque repéré par $(i, j)$ et en allant jusqu'au disque $A$.
   On demande que cette fonction ne manipule que des entiers (pas de création de liste). En pratique, lors des appels récursifs, seules les valeurs des paramètres `i` et `j` changeront, pas celle de `L`.
   Remarquons :

   - que cette fonction ne traite pas le choix du score maximal obtenu en partant de $D$ ;
   - que cette fonction doit renvoyer uniquement le score, et pas le chemin pour l'obtenir.

```
def scoreMaxRecPart(L,i,j):
    n , p = len(L) , len(L[0])
    if ...           : # cas de base
        return  ...
    else:
        # on recherche le score max
        # parmi les trois (ou 2) successeurs de (i,j) en effectuant
        # un appel récursif à cette fonction pour chaque successeur
        scoreMax = scoreMaxRecPart(L,i,j+1)
        if i-1 >= 0: # pour écarter le cas i=0
            score = scoreMaxRecPart(L , ... , ... )
            if score > scoreMax:
                scoreMax = ...
        if  ...   :
            score = ...
            if ...           :
                scoreMax = ...
        return ...
```

8. Déduire de la fonction précédente une fonction `scoreMaxRec(L)` permettant de calculer le score maximum d'un parcours, c'est à dire à partir du disque $D$.

9. Quelle est, en fonction de $n$ et $p$, la complexité de la fonction `scoreMaxRec(L)` ?
On donnera une minoration, que l'on justifiera brièvement.

## V. Résolution par mémoïzation

La méthode récursive naïve utilisée dans le paragraphe précédent donne une complexité mauvaise. Pour remédier à cela, nous allons recourir à une méthode de mémoïzation, qui reprend la méthode récursive en évitant les appels multiples à la fonction avec les mêmes paramètres.
Pour cela, on stocke les résultats obtenus dans une grille de scores (représentée par une liste de listes `Lscore` de mêmes dimensions que la liste de listes représentant la grille) destinée à contenir $M_{i,j}$ et position $(i, j)$, et on ne lance un appel récursif pour calculer le score maximal à partir d'un disque $(i, j)$ qu'après avoir vérifié que ce disque ne contient pas déjà le score voulu.

10. Représenter la grille des scores que l'on doit obtenir après remplissage par cette méthode à partir de la grille représentée sur la figure 1.

11. Écrire une fonction `grilleZeros(Lgrille)` qui construit et renvoie une liste de listes aux mêmes dimensions que la grille représentée par `Lgrille` et contenant des 0.

12. Précisons l'organisation de notre méthode. La fonction effectuant les appels récursifs sera une fonction auxiliaire `remplitLscore(L,Lscore,i,j)` qui ne renvoie rien mais remplit selon le principe décrit ci-dessus la case `(i,j)` de la liste de listes contenant les scores `Lscore`. On désigne toujours par `L` la liste de listes contenant les poids de la grille.
Compléter la fonction suivante afin qu'elle remplisse la case correspondant au disque `(i,j)` de `Lscore` suivant la méthode expliquée ci-dessus.
Remarque : on considèrera que tous les poids sont strictement positifs, de sorte que le fait que `Lscore[i][j]` est égal à 0 signale le fait que cette case n'a pas été traitée.

```
def remplitLscore(L,Lscore,i,j):
    n , p = ... , ...
    if ...          : # condition assurant que la case
                      #  n'a pas déjà été traitée
        if ...    : # cas de base
            Lscore[i][j] = ...
        else :
            remplitLscore(L,Lscore,i,j+1)
            scoreMax = Lscore[i][j+1]
            if i-1 >= 0:
                remplitLscore(...
                if     ...          > scoreMax:
                    scoreMax = ....
            if     ...    :
                ...
                if     ...                :
                    ...
            Lscore[i][j] = ...
```

13. Écrire une fonction `scoreMaxMemo(L)` qui utilise la fonction précédente afin d'obtenir et de renvoyer le score maximal que l'on peut obtenir avec la grille représentée par `L`.

## VI. Programmation dynamique

Dans cette partie on met en place une approche par programmation dynamique. Pour cela, on va mémoriser les résultats intermédiaires dans une nouvelle liste de listes (toujours notée `Lscore`), de mêmes dimensions que `L`, dont la case d'indice $(i, j)$ (position $j$ sur le niveau $i$) est destinée à contenir la valeur de $M_{i,j}$.
On rappelle que le principe de la programmation dynamique est de remplir la grille des scores maximaux «de bas en haut »(ici de la droite de la grille à sa gauche) en exploitant les équations établies au IV.

14. On va tout d'abord écrire une fonction `indMaxDisqueSuivant(Lscore,i,j)` qui renvoie le niveau `ind` du disque accessible à partir de $(i, j)$ en lequel `Lscore[ind][j+1]` est le plus grand. Cet indice sera donc choisi parmi `i`, `i-1`, `i+1` (sauf cas particuliers des première et dernière lignes).
Par exemple, en reprenant l'exemple de la figure 2, on a $M_{0,3} = 6$, $M_{1,3} = 7$ et $M_{2,3} = 9$. Le score maximal en partant de $(1, 2)$ sera donc obtenu en choisissant pour disque suivant le disque $(2, 3)$. La fonction précédente doit donc renvoyer 2.
On fera attention à traiter à part les cas $i = 0$ et $i = n - 1$.

15. Écrire une fonction `LscoreProgDyn(L)` qui crée, remplit (selon la méthode décrite au début de cette partie) et renvoie une liste de listes `Lscore` contenant, pour chaque couple valable $(i, j)$ de la liste `L` représentant la grille, le score maximal du chemin partant du disque $(i, j)$ et allant jusqu'au disque $A$.

16. Écrire une fonction `scoreMaxProgDyn(L)` qui utilise la fonction précédente afin d'obtenir et de renvoyer le score maximal que l'on peut obtenir pour un parcours de la grille représentée par `L`.

17. Quelle est, en fonction des dimensions de `L`, la complexité de cette fonction ?

En plus de déterminer le score maximal, on souhaite déterminer le parcours qui permet d'obtenir ce score.
Pour cela, on adapte la méthode précédente : en plus de construire une liste de listes `Lscore`, on va construire une liste de listes aux mêmes dimensions `Lsuiv` qui contient, , pour chaque couple valable $(i, j)$ de la liste `L` représentant la grille, le niveau du disque suivant le disque $(i, j)$ sur le chemin permettant d'atteindre ce score maximal.
Pour les disques de la colonne $p - 1$, le niveau du disque suivant sera fixé à `None`.
Pour fixer les idées, reprenons la grille de la figure 2. Dans `Lscore`, la case $(1, 2)$ (représentant le disque grisé) doit contenir le score 15). Dans `Lsuiv`, cette case devra contenir le niveau 2.

18. Représenter la grille `Lsuiv` correspondant à la grille représentée à la figure 1.

19. Détailler la reconstruction du parcours de score maximal pour la grille de la figure 1 à partir de la grille `Lsuiv` construite à la question précédente.

20. Écrire une fonction `LscoreLsuivProgDyn(L)` qui renvoie une couple constitué de `Lscore` et `Lsuiv`, obtenus à partir de la grille représentée par `L`.

21. À partir de la fonction précédente, écrire une fonction `ParcoursMaxiProgDyn(L)` qui renvoie le parcours de score maximal de la grille représentée par `L`.