? À rendre mercredi 5 novembre 2025

Devoir Maison n°3 : Tableaux, Glouton, SQL

- Le DM devra être rendu en **manuscrit**. Je vous encourage à y réfléchir dessus avec un crayon et un papier, sans ordinateur.
- Les algorithmes devront être commentés.
- Pour toute question, n'hésitez pas à me contacter par mail : nvalade.pcsi@gmail.com

Exercice n°1 Révisions sur les tableaux

Partie I Quelques rappels

— Création d'un tableau de 5 lignes et 10 colonnes contenant des 0 :

```
# 1ère solution :
T = []
for i in range(5) :
    ligne = []
    for j in range(10) : # chaque ligne a 10 éléments
        ligne.append(0) # remplissage d'une ligne
    T.append(ligne)
# 2ème solution (par compréhension:
T = [ [ 0 for j in range(10) ] for i in range(5) ]
```

— L'élément de ligne `i` et de colonne `j` du tableau précédent s'obtient par `T[i][j]`. En effet `T[i]` permet de se placer dans l'élément de rang `i` de `T` qui est une liste. Puis on récupère l'élément de rang `j` de cette liste.

— Soit un tableau `T`, on veut créer un tableau `T2` qui contient les éléments de `T` élevés au carré.

```
T2 = []
for i in range(len(T)):
    ligne = [] # liste
    for j in range(len(T[0])):
        ligne.append(T[i][j]**2)
    T2.append(ligne)
```

Partie II Applications

Q1. ♥ Écrire une fonction `zeros(n:int,m:int)->list[list]` qui crée un tableau rempli de zéros de `n` lignes et `m` colonnes.

Q2. ♥ Écrire une fonction `dim(T:list[list])->(int,int)` qui prend en argument un tableau `T` et qui renvoie le couple d'entiers `(h,w)` où `h` est la hauteur et `w` la largeur du tableau `T`.

Q3. ♥ Écrire une fonction `copie(T:list[list])->list[list]` qui prend en argument un tableau `T` et qui renvoie une copie du tableau `T` totalement indépendant de `T`.
Il est interdit d'utiliser le module `copy` et les fonctions `copy` ou `deepcopy`, par contre, vous êtes invités à utiliser les deux fonctions écrites précédemment.

Q4. Écrire une fonction `produit(n:int,m:int)->list[list]` qui, après créé un tableau rempli de zéros (*on utilisera une fonction précédente !*), renvoie un tableau pour lequel chaque case est remplie du produit (rang de la ligne)×(rang de la colonne).

Q5. ♪♪ Écrire une fonction `moyenne(T:list[list])->list[list]` qui renvoie un nouveau tableau dans lequel la valeur contenue dans une case correspond à la moyenne des quatre cases à côté d'elle (gauche, droite, au-dessus, en-dessous). Les cases sur les quatre bords du tableau ne sont pas modifiées.

Q6. ♪♪♪ Écrire une fonction `reduc_moitie(T:list[list])->list[list]` qui renvoie un nouveau tableau correspondant à `T` dont toutes les colonnes de rangs impaires ont été enlevées.

Exercice n°2 Base de données

Afin d'acheter votre équipement nécessaire pour effectuer la randonnée, vous consultez la base de données des magasins de sport.

`Ville` (`code_postal`, `nom`)
`Magasin` (`code_mag`, `nom`, `adresse`, `code_postal`)
`Vend` (`idvente`,`idmag`,`idart`)
`Article` (`code_art`, `nom`, `prix`, `rayon`).

`code_postal` de la table `Magasin` est une clé étrangère en relation avec la clé primaire `code_postal` de la table `ville`.

`idmag` est une clé étrangère de la table `Vend` en relation avec la clé primaire `code_mag` de la table `Magasin`
`idart` est une clé étrangère de la table `Vend` en relation avec la clé primaire `code_art` de la table `Article`

Q1. Le nom de la ville peut-il servir de clé primaire ? Pourquoi ?

Q2. Écrire la requête permettant d'obtenir la liste des articles du rayon « Randonnée » et à moins de 2 € classés par ordre de prix décroissant.

Q3. Écrire la requête permettant d'obtenir le nombre d'articles distincts du rayon Randonnée.

Q4. Écrire la requête permettant d'obtenir les troisième, quatrième et cinquième articles les moins chers du rayon « Randonnée » .

Q5. Écrire la requête permettant d'obtenir la liste des magasins où on peut acheter un « Sac à dos 50 L ».

Q6. Écrire la requête permettant d'obtenir le prix moyen des articles par rayon du magasin de code 53.

Q7. ♪ ♪ Écrire la requête permettant d'obtenir la liste des noms des villes ayant au moins trois magasins de sport.

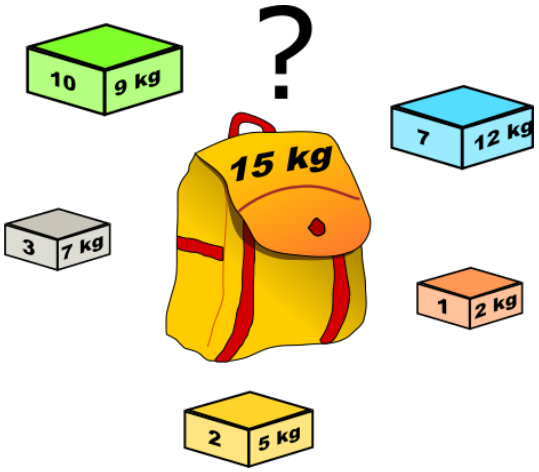
Q8. ♪ ♪ Écrire la requête permettant d'obtenir la liste des rayons coûtant en moyenne moins de 20 €.

Q9. ♪ ♪ ♪ Écrire la requête permettant d'obtenir la liste des articles de prix supérieurs à celui du « Sac à dos 50 L ».

Q10. ♪ ♪ ♪ Écrire la requête permettant d'obtenir le pourcentage d'articles dont le prix est supérieur au prix moyen de l'ensemble des articles.

Exercice n°3 Préparation d'un sac à dos

« Étant donné plusieurs objets possédant chacun une masse et une valeur (qui mesure son importance) et étant donné une masse maximale pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser la masse maximale autorisée pour le sac ? »
On considère les objets, numérotés $i \in [\![0, n-1]\!]$, de valeurs v_i et de masses m_i.
Les valeurs et masses sont stockées dans une liste de liste des `objets=[[mi,vi]]` :
`objets=[[9,10],[5,2],[7,3],[2,1],[12,7]]`



Partie I La méthode brutale

La première idée qui vient est de tester toutes les possibilités de placement des objets dans le sac, à chaque fois on calcule la masse et la valeur emportée. Puis de sélectionner la solution optimale, c'est-à-dire qui vérifie la masse maximale autorisée pour le sac tout en maximisant la valeur emportée.

Q1. Combien de combinaisons sont-elles à tester ? Quelle serait la complexité temporelle de l'algorithme ? Quel peut être l'intérêt d'une telle façon de procéder ? Quel est son inconvénient ?

Une méthode approchée a pour but de trouver une solution avec un bon compromis entre la qualité de la solution et le temps de calcul.
Pour le problème du sac à dos, nous allons utiliser l'**algorithme glouton** en procédant de cette façon :

— trier tous les objets par ordre décroissant de la valeur ;

— sélectionner les objets un à un dans l'ordre décroissant de leur valeur et ajouter l'objet sélectionné dans le sac si la masse maximale reste respectée.

Partie II Trier les objets

Pour mettre en œuvre l'algorithme glouton, il est nécessaire de trier les objets par ordre de valeur (importance) décroissant. Pour cela, nous proposons deux algorithmes de tris.

Partie II.a) Un premier tri : le tri bulles

On adapte ici l'algorithme du tri bulles au tri par valeurs décroissantes.
La liste que l'on doit trier est une liste de listes `objets=[[mi,vi]]` que l'on souhaite trier par ordre décroissant des valeurs des objets.
Le tri à bulles consiste à faire monter les objets (c'est-à-dire les sous listes `[mi,vi]`) de valeur `vi` *la plus faible* en permutant successivement les objets de la liste : on parcourt la liste, et à chaque fois que l'objet de gauche est de valeur `vi` strictement *inférieure* à celle de l'objet de droite, on les permute. À la fin de ce parcours, l'objet de plus *petite* valeur `vi` de la liste est en dernière position.
On recommence le parcours de la liste pour trier les $n-1$ premiers éléments, puis les $n-2$ premiers éléments, etc.
Le nom « tri à bulles » vient du fait que les éléments les plus grands (ici, plus *petits*) remontent plus vite, comme les bulles dans l'eau.

Q2. Compléter le document réponse, la fonction itérative `tri_bulle(L:list[list])->list[list]` qui trie une liste `L` de listes de deux éléments dans l'ordre décroissant du deuxième élément, selon l'algorithme ci-dessus.
Par exemple pour l'application de la fonction à la liste des `objets`, on obtient :

```
>>> objets=[[9,10],[5,2],[7,3],[2,1],[12,7]]
>>> tri_bulle(objets)
[[9,10],[12,7],[7,3],[5,2],[2,1]]
```

Q3. Évaluer en justifiant la complexité temporelle de cette fonction. Commenter.

Partie II.b) Un deuxième tri : le tri rapide

On rappelle que le tri rapide est un tri récursif basé sur le principe de « diviser pour mieux régner », dans lequel on divise un problème initial en deux sous-problèmes portant sur moins d'éléments.

On choisit un élément, que l'on notera `p`, appelé pivot, le premier élément de la liste, on partitionne la liste en trois sous listes : une liste contenant les éléments strictement inférieurs à `p`, une liste contenant les éléments égaux au pivot `p`, et une liste contenant les éléments strictement supérieurs à `p`. On trie récursivement chacune des deux listes et on rassemble tout.

On adapte ici l'algorithme du tri rapide au cas de la liste de listes de deux éléments précédente que l'on souhaite trier par ordre décroissant du deuxième élément de chaque sous-liste.

Par exemple pour l'application de la fonction à la liste des `objets`, on obtient :

```
>>> objets=[[9,10],[5,2],[7,3],[2,1],[12,7]]
>>> tri_rapide(objets)
[[9,10],[12,7],[7,3],[5,2],[2,1]]
```

Q4. Compléter, **sur le document réponse**, la fonction `tri_rapide` qui trie la liste de liste `L` de deux éléments selon l'algorithme du tri rapide, par ordre décroissant du deuxième élément.

Le meilleur des cas correspond au cas où la séparation se fait en deux parties égales. On peut alors montrer (avec pas mal de calculs !) que le tri est de complexité quasi-linéaire $O(n\log(n))$.

Partie III Algorithme Glouton : remplissage (optimal ?) du sac

Q5. Mettre en œuvre, à la main, l'algorithme glouton pour l'exemple considéré en expliquant chaque étape.

Q6. Compléter la fonction `glouton`, sur **le document réponse**, qui permet de déterminer la composition optimale (plutôt sensée être) du sac à dos selon l'algorithme glouton.

Q7. L'algorithme glouton donne-t-il la solution optimale dans l'exemple étudié ici ?

Qu'en est-il pour des objets de masses 8 kg, 12 kg, 6 kg, 1 kg, 5 kg et de valeurs (importances) respectives 8, 10, 6, 1, 4 ? Commenter.

DOCUMENT RÉPONSE À RENDRE

NOM : _____ Prénom : _____

Q2

```
def tri_bulle(L):
    n=len(L)
    for j in range(.....):
        for i in range(.........):
            if ........................................
                ........................................
    return L
```

Q4

```
def tri_rapide(L):
    if len(L)<=1 : # cas de base
        return .....
    p=L[0] #choix du pivot, le premier élément de L
    Linf,Lp,Lsup=[],[p],[]# Linf (Lsup) liste des objets de L de valeurs strictement inférieures (supérieures) à la valeur du pivot
    for  ..................... : #parcours des objets de la liste L
        if ..................... :  # comparaison de l'élément de rang 1 de la sous-liste à celui du pivot
            ......................
        elif ..................... :
            ......................
        else :
            ......................
    return  ..................... + Lp + ..................... # appels récursifs
```

Q6

```
def glouton(objets,Mmax):
    # Etape 1 : tri décroissant selon l'algorithme du tri rapide
    objets_tries= .....
    # Etape 2 : fabrication du sac
    sac=[] # liste des objets à emporter
    masse_tot=0
    valeur_tot=0
    # parcours des objets par ordre de valeur décroissant, ajout de l'objet si la masse totale est inférieure à la masse maximale
    i=0 # parcours de objets_tries
    while .......... and ............. :
        if masse_tot + .......  > ..... :
            i =  .....
        else:
            sac ..............
            masse_tot= ............
            valeur_tot= ..............
    return ..............
```