? À rendre mercredi 6 novembre 2024

Devoir Maison n°3

- Ce DM traite de divers thèmes
  - langage SQL,
  - tris (cf 1re année),
  - algorithme glouton (cf 1re année),
  - programmation dynamique.

- Le DM devra être rendu en **manuscrit**. Je vous encourage à y réfléchir dessus avec un crayon et un papier, sans ordinateur.
- Les fonctions des questions Q11, Q14 et Q21 sont à traiter **sur le document réponse** page 4.
- Pour toute question, n'hésitez pas à me contacter par mail : nvalade.pcsi@gmail.com ou par cahier-de-prepa.

I Base de données

On considère la base de données des jeux olympiques et paralympiques de Paris 2024 constituée des tables suivantes. On donne quelques exemples du contenu de chaque table pour illustrer leur constitution.

id	nom	prénom	ddn	pays	oly	para
1	Marchand	Léon	20020517	France	1	0
2	Aubert	Aurélie	19970609	France	0	1

Table 1 – Table `athletes`

id	sport	nom	sexe	jeux
1	athlétisme	finale 100 m	femme	oly
2	natation	finale 100 m nage libre	femme	para
3	triathlon	équipe	mixte	oly

Table 3 – Table `epreuves`

id	ida	ide
1	2	150

Table 2 – Table `participation`
ida est une clé étrangère en lien avec la clé primaire de la table `athletes`, et ide est une clé étrangère en lien avec la clé primaire de la table `epreuves`

id	ide	or	argent	bronze
1	45	1	50	46
20	150	2	60	70

Table 4 – Table `medailles`
`ide` est une clé étrangère en lien avec la clé primaire de la table `epreuves`. `or`, `argent` et `bronze` sont des clés étrangères en lien avec la clé primaire de la table `athletes` et indique qui a obtenu la médaille.

Q1. Écrire la requête en langage SQL qui donne les noms et prénoms des athlètes nés à partir de 2000.

Q2. Écrire la requête en langage SQL qui donne les noms et prénoms des athlètes ayant participé à la fois aux jeux olympiques et aux jeux paralympiques.

Q3. Écrire la requête en langage SQL qui donne le nombre d'athlètes envoyé par chaque pays aux jeux olympiques.

Q4. Écrire la requête en langage SQL qui donne le nom du pays ayant envoyé le plus d'athlètes aux jeux paralympiques, et le nom du pays ayant envoyé le moins d'athlètes aux jeux olympiques.

Q5. Écrire la requête en langage SQL qui donne le nombre d'épreuves organisées en "natation".

Q6. Écrire la requête en langage SQL qui donne le nombre d'athlètes paralympiques (on ne devra compter chaque athlète qu'une fois) ayant participé à des épreuves de triathlon.

Q7. Écrire la requête en langage SQL qui donne les épreuves auxquelles ont participé Léon Marchand.

Q8. Écrire la requête en langage SQL qui donne le nom et prénom de l'athlète paralympique ayant obtenu la médaille d'or à l'épreuve féminine "finale individuelle BC1" dans le sport "boccia".

Q9. Écrire la requête en langage SQL qui donne la liste des pays ayant envoyé 4 athlètes ou moins aux jeux olympiques et paralympiques réunis.

II Programme de rêve

Vous vous rendez pour une journée aux jeux olympiques ou paralympiques (oui... c'est un peu tard). Vous souhaitez vous faire votre programme de rêve.

L'épreuve d'indice i est définie par l'instant de début (d_i), l'instant de fin (f_i) et son intérêt (à qui vous lui avez attribué une valeur v_i) : $[d_i, f_i, v_i]$.

Ainsi la liste des n épreuves ayant lieu la journée où vous y êtes est notée $E = [[d_0, f_0, v_0], ..., [d_{n-1}, f_{n-1}, v_{n-1}]]$.

Vous souhaitez assister à un maximum d'épreuves tout en optimisant les intérêts que vous y portez. On cherche à déterminer le sous-ensemble S de E tel que $\sum_{i \in S} v_i$ est maximal.

Vous ne pouvez bien évidemment assister qu'à une épreuve à la fois, et vous n'assistez qu'à des épreuves entières. L'intersection entre les intervalles de deux épreuves auxquelles vous assistez est l'ensemble vide ou limité à un nombre ($d_i = f_j$).

II.1 Questions préliminaires

Q10. Écrire une fonction d'en-tête `valeur(E:list[list])->int` qui prend en argument une liste `E` du format précédent (représentant un ensemble d'épreuves) et qui renvoie la valeur totale de l'ensemble des épreuves.

II.2 Algorithme glouton

On envisage un algorithme glouton. Pour cela, on trie les épreuves par instant de fin croissant et on applique l'algorithme glouton suivant :

- on choisit l'épreuve se terminant le plus tard,
- puis on choisit l'épreuve se terminant au plus tard parmi celles qui sont compatibles avec l'épreuve choisie précédemment,
- et ainsi de suite.

On suppose qu'il n'y a pas deux épreuves qui finissent à la même heure.

II.2.a) Tri rapide

On souhaite trier les épreuves par **ordre d'instant de fin croissant**. On souhaite écrire l'algorithme de tri rapide appliqué à une liste de listes `E` de trois éléments, que l'on classe par ordre croissant de l'élément de rang 2 de chaque sous-liste :

- choisir le premier élément de la liste, que l'on notera `p`, appelé pivot,
- on partitionne la liste en deux sous listes :
  - une liste `E1` contenant les épreuves d'heure de fin strictement inférieure à l'heure de fin de `p`,
  - une liste `E2` contenant les épreuves d'heure de fin supérieure à celle de `p`. On trie récursivement chacune des deux listes et on rassemble le tout.

Par ex. `tri_rapide([[0,3,2],[3,1,4],[5,5,1],[2,2,3]])` renvoie `[[3,1,4],[2,2,3],[0,3,2],[5,5,1]]`

Q11. Compléter la la fonction d'en-tête `tri_rapide(E:list[list])->list[list]` qui prend en argument une liste `E` que l'on souhaite trier par ordre croissant du deuxième élément de chaque sous-liste.

Pour la suite, on suppose que la liste `E` est triée par ordre d'instant de fin croissant.

II.2.b) Glouton

On donne l'exemple de la liste d'épreuves suivantes :
`epreuves=[[9,10,3],[9,13,2],[11,14,1],[11,15,3],[17,21,3],[19,22,2]]`.

Q12. Mettre en œuvre l'algorithme glouton à la main pour déterminer le programme optimal des épreuves à aller voir. Quelle est la valeur totale de ce programme ? Est-ce que la solution trouvée avec l'algorithme glouton est optimale ? Peut-on trouver une meilleure solution ?

Q13. Écrire une fonction d'en-tête `compatible(E:list[list],k:int)->int` qui détermine l'indice d'épreuve (parmi la liste `E`) compatible avec l'épreuve d'indice `k` et qui se termine au plus près de cette épreuve `k`.
La fonction renvoie -1 si aucune épreuve n'est compatible avec l'épreuve k.

Q14. Compléter la fonction d'en-tête `glouton(E:list[list])->list[int]` qui prend en argument une liste `E` des épreuves sous le format défini précédemment, et renvoie la liste `Eavoir` des épreuves à aller voir en utilisant l'algorithme glouton décrit précédemment.
On pourra avantageusement utiliser la fonction `compatible` écrite précédemment.
Pour la liste donnée en exemple, `glouton(epreuves)` renvoie `[[19,22,2],[11,15,3],[9,10,3]]`

Q15. Écrire les instructions nécessaires (on n'attend pas une fonction, mais un appel aux fonctions écrites précédemment) pour obtenir la somme des valeurs du programme élaboré grâce à l'algorithme glouton.

III Programmation dynamique

On cherche toujours à déterminer le sous-ensemble D de E tel que $\sum_{i\in D} v_i$ est maximal.

On suppose qu'on a une liste de n épreuves du type `E=[[d0,f0,v0],...]` triée par heure de fin croissante.

On note $S(E,k)$ la somme maximale des valeurs des épreuves qu'on peut aller voir, quand on considère les épreuves jusqu'au rang k inclus : $E_0 ... E_k$.

Q16. Pour quelle valeur de k obtient-on la solution à notre problème ?

Q17. Quelle est la valeur de $S(E,-1)$?

On cherche une relation de récurrence entre $S(E,k)$ (=la valeur totale en considérant les épreuves jusqu'au rang k) et $S(E,k-1)$ (=la valeur totale en considérant les épreuves jusqu'au rang $k-1$).

Pour comparer ces deux valeurs, il faut se demander si aller voir l'épreuve k est plus intéressant que de ne pas aller le voir.

Q18. (a) Exprimer $S(E,k)$ en fonction de $S(E,k-1)$ si vous n'allez pas voir l'épreuve k.

(b) Exprimer $S(E,k)$ en fonction de $S(E,.....)$ et v_k si vous allez voir l'épreuve k. On pourra utiliser la fonction `compatible(E,k)` qui renvoie le rang de l'épreuve compatible avec l'épreuve `k` et qui se termine au plus près de cette épreuve `k`.

(c) Comment exprimer $S(E,k)$ en fonction des deux cas précédents ?

Q19. Recopier et compléter la relation de récurrence suivante :

$$S(E,k) = \begin{cases} .... & \text{si } k=-1 \\ \max\Big(..... + S(E,.....), ...\Big) & \text{sinon} \end{cases}$$

Q20. En quoi est-ce un problème à sous-structure optimale ? Pourquoi est-il intéressant d'utiliser la programmation dynamique ?

Q21. En traduisant la relation de récurrence (Q19), compléter la fonction : `sol_mem(E:list[list])->list`.
On définit le dictionnaire `dico` qui stocke les solutions successives, ses clés sont les valeurs de k, et les valeurs associées $S(E,k)$, c'est-à-dire la valeur totale maximale des épreuves auxquelles vous pouvez assister en considérant les épreuves $E_0 ... E_k$.
`dico` est une variable locale pour `sol_mem`, et une variable globale pour la fonction auxiliaire `S(E:list[list],k:int)` qui renvoie la valeur maximale quand on considère les épreuves jusqu'à celle de rang k.

DOCUMENT RÉPONSE À RENDRE

NOM : _____ Prénom : _____

Q11

```
def tri_rapide(E):
    """
    E: liste des épreuves [[d,f,v]...]
    renvoi : liste des épreuves triée par ordre de fin croissant
    """
    if len(E)<=1: # cas de base
        return ......
    p=E[0] # pivot
    E1,E2=[],[] # E1 contient  les épreuves terminant avant p  ; E2 celles terminant après p
    for e in E : # parcourt de la liste des épreuves
        if ..... : # on compare les heures de fin de l'épreuve e avec l'heure de fin du pivot
            ..... # ajout de e à ....
        else:
            ..... # ajout de e à ....
    return ..... + ..... # appels récursifs et concaténation
```

Q14

```
def glouton(E):
    """
    E : liste des épreuves
    Eavoir : liste des épreuves à aller voir pour maximiser la valeur des épreuves vus
    """
    Eavoir=[] # liste des épreuves à voir
    i=len(E)-1 # parcours de épreuves à partir de celui qui termine le plus tard
    ..... # ajout de l'épreuve finissant le plus tard
    i = ..... # on cherche l'épreuve compatible
    while i>=0: # s'il y a une épreuve compatible
        ..... # ajout de l'épreuve finissant le plus tard
        i = ..... # on cherche l'épreuve compatible
    return Eavoir
```

Q21

```
def sol_mem(E):
    """
    Renvoie la valeur maximale des épreuves pouvant être vues parmi les épreuves E
    """
    dico={-1:0} # dictionnaire pour mémoïser, pour k=-1 (aucune épreuve), alors valeur totale =0
    def S(E,k):
        """ renvoie la valeur totale maximale en considérant les épreuves jusqu'au rang k """
        if ..................: # la solution a déjà été calculée
            return ..................
        else:
            a= ............................ # valeur totale si on ne choisit pas l'épreuve k parmi celles jusqu'au rang k
            vk= ..................... # valeur de l'épreuve k
            a=.............................. # valeur totale si on choisit l'épreuve k parmi celles jusqu'au rang k
            dico[k]=............................ # mémoise la valeur maximale
        return ..................
    return ........................... # réponse du problème quand on considère toutes les épreuves
```