DS 2

le 14/12/2024
Durée : 2h

Le devoir est constitué d'un exercice (sur 5 points) et d'un problème.

Exercice : Prévention des collisions aériennes

Cet exercice aborde l'enregistrement des plans de vol de différentes compagnies aériennes.
Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut 30,48 cm) au-dessus de la surface isobare à 1013,25 hPa. L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres «FL »(*flight level*) suivies de l'altitude en centaines de pieds : «FL310 »désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence.

Eurocontrol est l'organisation européenne chargée de la navigation aérienne, elle gère plusieurs dizaines de milliers de vol par jour. Toute compagnie qui souhaite faire traverser le ciel européen à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, Eurocontrol peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc.

Nous modélisons (de manière très simplifiée) les plans de vol gérés par Eurocontrol sous la forme d'une base de données comportant deux tables :

– la table `vol` qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes

- `id_vol` : numéro du vol (chaine de caractères) ;
- `depart` : code de l'aéroport de départ (chaine de caractères) ;
- `arrivee` : code de l'aéroport d'arrivée (chaine de caractères) ;
- `jour` : jour du vol (de type date, affiché au format `aaaa-mm-jj`) ;
- `heure` : heure de décollage souhaitée (de type time, affiché au format `hh:mi`) ;
- `niveau` : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
AF1204	CDG	FCO	2016-05-02	07:35	300
AF1205	FCO	CDG	2016-05-02	10:25	300
AF1504	CDG	FCO	2016-05-02	10:05	310
AF1505	FCO	CDG	2016-05-02	13:00	310

Figure 1 Extrait de la table `vol` : vols de la compagnie Air France entre les aéroports Charles-de-Gaule (Paris) et Léonard-de-Vinci à Fiumicino (Rome)

– la table `aeroport` qui répertorie les aéroports européens ; elle contient les colonnes

- `id_aero` : code de l'aéroport (chaine de caractères) ;
- `ville` : principale ville desservie (chaine de caractères) ;
- `pays` : pays dans lequel se situe l'aéroport (chaine de caractères).

id_aero	ville	pays
CDG	Paris	France
ORY	Paris	France
MRS	Marseille	France
FCO	Rome	Italie

Figure 2 Extrait de la table `aeroport`

Les types SQL `date` et `time` permettent de mémoriser respectivement un jour du calendrier grégorien et une heure du jour. Deux valeurs de type `date` ou de type `time` peuvent être comparées avec les opérateurs habituels(=, <, <=, etc.). La comparaison s'effectue suivant l'ordre chronologique.

1. Écrire une requête SQL qui fournit les codes des aéroports situés en France.

2. Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.

3. Écrire une requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport dont la principale ville desservie est Paris le 2 mai 2016

4. Que fait la requête suivante ?

```
SELECT id_vol
FROM vol
     JOIN aeroport AS d ON d.id_aero = depart
     JOIN aeroport AS a ON a.id_aero = arrivee
WHERE
     d.pays = 'France' AND
     a.pays = 'France' AND
     jour = '2016-05-02'
```

5. Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples (Id_1 , Id_2) des identifiants des vols dans cette situation.

Problème

I. Présentation

On considère le problème suivant. Une pyramide est constituée de disques, chacun d'entre eux portant un nombre. On cherche dans cette pyramide un chemin de score maximal.
Plus précisément, une telle pyramide est représentée dans la figure 1. Un chemin doit partir du disque du haut et arriver dans l'un des disques de la base en suivant les arcs. Un tel chemin est représenté en grisé dans la figure 2. Le score d'un chemin est la somme des nombres portés par les disques traversés. Le chemin grisé de la figure 2 a donc un score de 10.
Nous allons envisager plusieurs stratégies pour résoudre ce problème.
Dans un premier temps, nous mettons en place les structures de données permettant de traiter le problème.

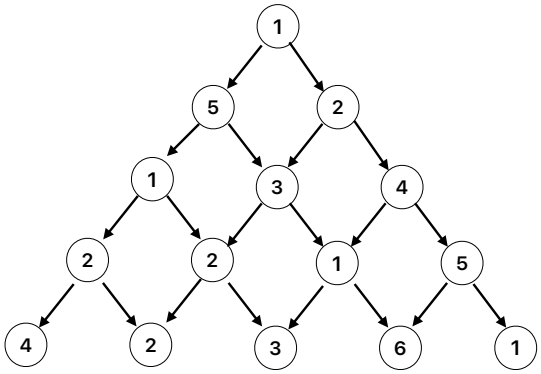


Figure 1 - Une pyramide

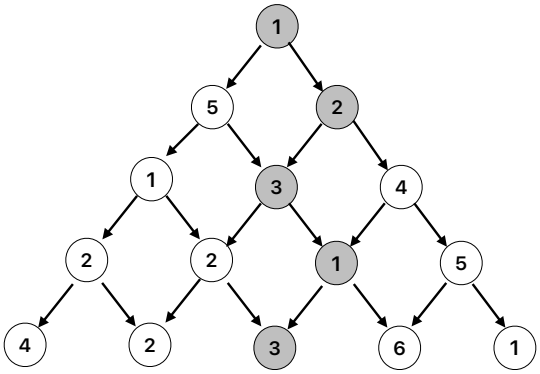


Figure 2 - Un chemin dans cette pyramide

II. Représentation des données du problème

On représente une pyramide par une liste de listes `p`, la liste `p[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 pyramide de la figure 1 est représentée par la liste `[[1],[5,2],[1,3,4],[2,2,1,5],[4,2,3,6,1]]`.
On obtient ainsi un système de repérage dans une pyramide : un disque est repéré par sa ligne (on utilisera également le terme niveau) et par sa position sur la ligne (on numérote ces positions à partir de la gauche en partant de 0). Par exemple, dans la pyramide de la figure 1, le disque situé en position 2 sur la ligne 3 porte le nombre 1.
Un chemin dans une pyramide est représenté par une liste d'entiers : la liste des positions des disques empruntés par le chemin à chaque niveau. Par exemple, le chemin grisé sur la figure 2 est représenté par la liste `[0,1,1,2,2]` : le 1 est à la position 0 de la ligne 0, le 2 à la position 1 de la ligne 1, le 3 à la position 1 de la ligne 2, le 1 à la position 2 de la ligne 3 et le 3 à la position 2 de la ligne 4.

1. Dessiner la pyramide représentée par la liste de listes `[[3],[1,2],[4,9,4],[2,6,1,7]]`.
   Griser (ou entourer les disques en une couleur différente) dans cette pyramide le chemin représenté par `[0,0,1,2]`.
   Calculer le score de ce chemin.

On souhaite dans un premier temps écrire un fonction qui vérifie qu'une liste donnée représente bien un chemin et un autre fonction qui calcule le score d'un chemin.
On remarque qu'une liste `ch` représente un chemin dans une pyramide `p` si et seulement si :

- elle est de même longueur que `p` ;
- elle commence par un 0 ;
- lorsque a et b sont deux éléments consécutifs de `ch`, alors $b = a$ ou $b = a + 1$.

2. Écrire une fonction `est_chemin(ch,p)` qui prend en paramètre une liste d'entiers `ch` et une pyramide `p` (tous deux supposés non vides) et qui renvoie `True` si `ch` est un chemin valable dans `p` et `False` sinon.

3. Écrire une fonction `score(ch,p)` qui prend en paramètre un chemin `ch` et une pyramide `p` (on ne demande pas de vérifier que ce chemin est valable) et qui renvoie le score de ce chemin.

4. On pourrait (en théorie) résoudre ce problème en calculant le score de chaque chemin et en déterminant le score maximal.
   En notant n le nombre de niveaux de la pyramide `p`, combien y a-t-il en tout de chemins possibles ?
   Quel type de complexité cette méthode exhaustive donnerait ?

III. Stratégie gloutonne

On va tout d'abord résoudre le problème de score maximale par une stratégie gloutonne. À chaque étape, on choisit le disque suivant qui porte le nombre de plus élevé.

5. Compléter la fonction `score_glouton(p)` qui prend en entrée un pyramide `p` et qui renvoie le score obtenu par cette stratégie dans cette pyramide ainsi que le chemin emprunté pour obtenir ce score.

```
def score_glouton(p):
    n = len(p)
    i = 0 # niveau
    pos = 0 # position dans le niveau
    ch = [pos]
    score = p[i][pos]
    while i+1 < n:
        if ...                :
            pos = ...
        # sinon, pos ne change pas
        i += 1
        ch.append(....)
        score = ...
    return score
```

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

7. Quel est, en fonction de la hauteur (nombre de niveaux) n de la pyramide, la complexité de cette méthode ?

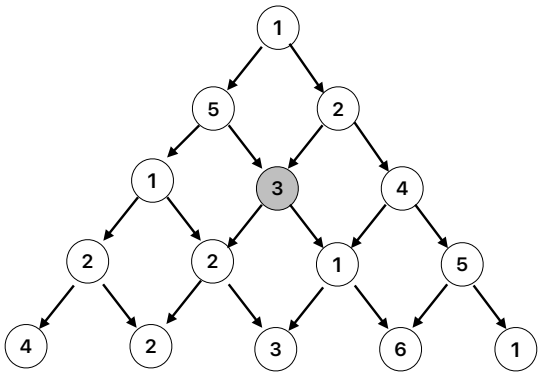
IV. Algorithme récursif naïf

On peut résoudre ce problème de manière récursive.
En effet, considérons une pyramide `p` de hauteur n. On va généraliser le problème et chercher à déterminer le score maximal que l'on peut obtenir en partant d'un disque quelconque de `p` et en allant jusqu'au dernier niveau.
Rappelons qu'un disque est repéré par son niveau i ($0 \leq i < n$) et sa position j dans ce niveau ($0 \leq j \leq i$). On notera $M(p, i, j)$ le score maximal que l'on peut obtenir en partant du disque situé en position j sur la ligne i et en descendant jusqu'à un disque de la base. On adopte également la notation suivante : le nombre porté par le disque situé en position j sur la ligne i est noté p_{ij} (dans la liste de listes `p`, avec les conventions adoptées précédemment, il s'agit de `p[i][j]`).
Notre objectif final est donc de déterminer $M(p, 0, 0)$. Remarquons que si $i = n - 1$, alors $M(p, i, j)$ est égal à p_{ij}.
Pour les disques qui ne sont pas situés sur le dernier niveau, on fait la remarque suivante : le score maximal $M(p, i, j)$ est obtenu en rajoutant p_{ij} au maximum des scores maximaux obtenus en partant de chacun des deux disque accessibles à partir du disque en position (i, j). Plus précisément :

$$M(p, i, j) = p_{ij} + \max\{M(p, i + 1, j), M(p, i + 1, j + 1)\} \ .$$

On illustre cela sur la figure suivante :

Figure 3



Le sommet grisé est au niveau 2 en position 1 ($i = 2$ et $j = 1$). Pour connaître le score maximal à partir de ce sommet, il faut regarder les deux sommet auxquels on peut accéder à partir de celui-ci. On observe que $M(p, 3, 1) = 5$ et $M(p, 3, 2) = 7$. Il vaut donc mieux descendre à droite et on obtient $M(p, 2, 1) = p_{21} + M(p, 3, 2) = 3 + 7 = 10$.

8. Écrire une fonction récursive `scmax(p,i,j)` qui prend en paramètre une pyramide `p`, un niveau `i` et une position `j` (tous deux supposés valables) et qui renvoie le score maximal que l'on peut obtenir en partant du disque repéré par (i, j) et en descendant jusqu'au dernier niveau.
   On demande que cette fonction ne manipule que des entiers (pas de création de liste).

9. Déduire de la fonction précédente une fonction `calcule_score_max(p)` permettant de calculer le score maximum à partir du somment de la pyramide `p`.

10. Quelle est, en fonction de la hauteur n de la pyramide, la complexité de la fonction `calcule_score_max(p)` ?
    On justifiera brièvement la réponse.

V. 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 pyramide (notée `s` par la suite) de même dimension que `p` dont le disque (i, j) (position j sur le niveau i) est destinée à contenir la valeur de $M(p, i, j)$.
On rappelle que le principe de la programmation dynamique est de remplir la pyramide `s` de bas en haut en exploitant les équations établies au IV.

11. Représenter la pyramide `s` correspondant à la pyramide `p` de la figure 1.

12. Écrire une fonction `pyramide_nulle(n)` qui construit et renvoie une pyramide à n niveaux remplie de 0 (sous la forme décrite au I d'une liste de listes).

13. Écrire une fonction `prog_dyn(p)` qui prend en paramètre une pyramide `p` et qui renvoie le score maximal d'un chemin partant du sommet de `p` et arrivant à sa base.
Cette fonction devra construire, selon le principe de la programmation dynamique, une pyramide `s` de même taille que `p` qui contient sur le disque (i, j) la valeur de $M(p, i, j)$.

14. Quelle est la complexité de cette fonction en fonction du nombre n de niveaux de la pyramide `p` ?

En plus de déterminer le score maximal, on souhaite déterminer le chemin qui permet d'obtenir ce score.
Pour cela, outre la pyramide `s` contenant les scores, on va construire une pyramide `suiv` qui indique, dans chaque disque, quel est le disque suivant sur le chemin de score maximal. Plus précisément, le disque (i, j) de la pyramide `suiv` contient l'indice où il faut descendre au niveau $i + 1$ pour construire dans `p` un chemin de score maximal à partir du disque au niveau i en position j.
Ceci ne concerne pas les disques au niveau $n - 1$ (n étant la hauteur de la pyramide `p`), niveau de la base de la pyramide, qui resteront vides (ou plutôt contiendront 0).

15. Représenter la pyramide `suiv` correspondant à la pyramide `p` de la figure 1.

16. Écrire une fonction `prog_dyn_suiv(p)` qui applique l'algorithme décrit ci-dessus et qui renvoie le score maximal d'un chemin partant du sommet de `p` ainsi que la pyramide notée `suiv` dans cette description.

17. Détailler la reconstruction du chemin de score maximal pour la pyramide de la figure 1 à partir de la pyramide `suiv` construite à la question 15.

18. Écrire une fonction `reconstruit(suiv)` qui prend en paramètre une pyramide `suiv`, supposée obtenue par un appel à `prog_dyn_suiv(p)`, et qui renvoie le chemin de score maximal pour `p`.

VI. Résolution par mémoïzation

On résout enfin ce problème par 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 pyramide `s` 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.
Plus précisément, la fonction effectuant les appels récursifs sera une fonction auxiliaire `remplissage_s(i,j)` qui ne renvoie rien mais remplit selon le principe décrit ci-dessus la pyramide `s`.

19. Compléter la fonction suivante afin qu'elle permette d'obtenir le score maximal de la pyramide `p` selon le principe décrit ci-dessus.

```
def score_max_memo(p):
    n = len(p)
    s = pyramide_nulle(n)
    def remplissage_s(i,j):
        if ......     :
            if i == ....  :
                ....
            else:
                remplissage_s( ... , ... )
                remplissage_s( ... , ... )
                s[i][j] = ....
    remplissage_s(0,0)
    return ....
```