Devoir surveillé #2

Vendredi 2025-12-19 ; durée : deux heures

Ce sujet est partiellement repris de l'épreuve de *composition d'informatique B* (filière MP hors spécialité info, filière PC) XEC (École Polytechnique, ENS Cachan, ESPCI) 2014.
Les différences principales sont l'ajout d'une nouvelle partie indépendante portant sur les bases de données, l'utilisation exclusive de Python comme langage de programmation, l'ajout de deux questions, et quelques modifications mineures du texte et d'une question.

Grands rectangles et cartographie

Ce sujet porte sur la détection de zones rectangulaires monochromatiques dans une image. Imaginons par exemple une image en noir et blanc, le problème est d'en trouver la plus grande zone noire. Ces zones monochromatiques peuvent être représentées de manière compacte en retenant les cordonnées des coins et la couleur interne. En décomposant une image selon ces grands rectangles, il est possible de la compresser efficacement, les zones monochromatiques pouvant être représentées de manière très compacte.
La complexité, ou le temps d'exécution, d'un programme P est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend d'un paramètre n, on dira que P a une complexité en $O(f(n))$, s'il existe $K > 0$ tel que la complexité de P est au plus $Kf(n)$, pour toute instance de taille n, pour tout n. Lorsqu'il est demandé de garantir une certaine complexité, le ou la candidate devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Les requêtes de la partie I doivent être écrites en langage SQL. Toutes les autres parties sont à traiter en utilisant le langage de programmation Python.
Dans tout le sujet, on pourra admettre le résultat d'une question précédente même si celle-ci n'a pas été traitée.

Partie I. Bases de données

Dans cette partie on étudie une base de données de photographies aériennes utilisées en cartographie. Les informations que l'on souhaite pouvoir manipuler à travers cette base sont les zones photographiées (découpées en *tuiles*), les aéronefs utilisés, les organismes de cartographie commanditant ou réalisant les prises de vues, et les photos elles-mêmes.
La base de données est constituée des tables suivantes :

- **tuile** : (id : entier ; latN : flottant ; longE : flottant ; long : flottant ; larg : flottant)
- **org** : (id : entier ; nom : texte)
- **aeronef** : (id : entier ; nom : texte)
- **photo** : (id : entier ; id_tuile : entier ; id_aeronef : entier ; id_org : entier ; annee : entier)

On donne ci-dessous un exemple de données pour chacune de ces tables. Attention : les informations contenues dans ces exemples sont données à titre indicatif : vous ne pouvez pas les utiliser pour répondre aux questions !

tuile

id	latN	longE	long	larg
1	45.187	5.725	2000	2000
2	45.415	6.184	2000	2000

org	
id	nom
1	Institut Géographique National
2	Ordnance Survey

aeronef	
id	nom
1	Beechcraft Super King Air 200 T
2	Boeing B-17
3	Hurel-Dubois HD-34

photo				
id	id_tuile	id_aeronef	id_org	annee
1	1	1	1	2025
2	1	2	1	1979
3	2	3	1	1984
4	2	1	1	2025
5	1	2	2	1979

Étude de la base de données

1. Donnez (en justifiant) le type de l'association « a été prise par » entre une photo (aérienne) et un aéronef.
2. Les informations à propos des tables ci-dessus n'identifient pas les éventuelles clefs étrangères qui y sont présentes. Donnez une clef étrangère de la table **photo** et la clef candidate qu'elle référence.

On aimerait enrichir la base de données pour y stocker l'information des flottes d'aéronefs utilisés par les différents organismes : chaque organisme utilise un ou plusieurs aéronefs d'un ou plusieurs type, et chaque type d'aéronef peut être utilisé par un ou plusieurs organismes.

3. Proposez une modification de la base de données (qui y ajoute une ou plusieurs tables, et/ou modifie une ou plusieurs tables existantes) permettant de stocker cette information. Cette modification devra entre autres permettre de répondre à des questions comme « quels sont les noms d'aéronefs utilisés par . . . » ; « combien d'aéronefs . . . sont utilisés par . . . ».

Interrogation de la base de données

4. Écrivez une requête SQL qui fournit toutes les informations à propos des photos prises en 2025.
5. Écrivez une requête SQL qui fournit toutes les informations à propos des photos prises par l'organisme s'appelant *Institut Géographique National*
6. Écrivez une requête SQL qui fournit toutes les informations à propos des photos prises l'année la plus ancienne (1979 pour les tables données en exemple).
7. Écrivez une requête SQL qui fournit pour chaque aéronef le nombre total de photos qu'il a prises. La réponse à la requête devra être constituées de deux colonnes : l'une donnant le nom des aéronefs, et la seconde nommée `nb_photos` donnant le nombre de photos.
8. Écrivez une requête SQL qui fournit pour chaque organisme le nombre de photos prises par année (où il a pris des photos). La réponse à la requête devra être constituée de trois colonnes : l'une donnant le nom d'un organisme, la seconde donnant une année, et la troisième nommée `nb_photos` le nombre de photos prises par l'organisme cette année là.
9. Écrivez une requête SQL qui fournit pour chaque organisme le nombre moyen de photos qu'il prend par année, où cette moyenne est calculée en prenant seulement en compte les années d'activité (où des photos ont été prises). La réponse à la requête devra être constituée de deux colonnes : l'une donnant le nom d'un organisme et la seconde la moyenne calculée.

Partie II. Recherche unidimensionnelle

Dans cette partie, nous allons étudier le problème de reconnaissance de forme sur des tableaux unidimensionnels. Nous supposerons donnés un entier n et un tableau *tab* de taille n représenté par une `list[int]` dont les cases contiennent 0 ou 1. Le but de cette partie est de trouver le nombre maximal de 0 contigus (c'est à dire figurant dans des cases consécutives) dans le tableau.
Dans cette partie nous utiliserons le tableau suivant comme exemple :

```
#  i : 0  1  2  3  4  5  6  7  8  9  A  B  C
tab = [0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1]
```

10. Écrire une fonction `nombreZerosDroite(i:int, tab:list[int], n:int)` prenant en paramètres le tableau *tab* de taille n ainsi qu'un indice i compris entre 0 et $n-1$. Cette fonction devra renvoyer le nombre de cases contiguës du tableau contenant 0 à droite de la case d'indice i, cette case étant comprise. D'un point de vue formel il s'agit de renvoyer 0 si la case d'indice i contient un 1 et sinon de renvoyer :

$$\max\{j-i+1 \text{ tel que } i \leq j \leq n \text{ et } \forall k \in [\![i,j]\!],\ tab[k]=0\}$$

Sur le tableau exemple précédent, en prenant comme indice i = 3 nous obtenons 0 et pour l'indice i = 5 votre fonction devra retourner 2. Notez que si la case i contient 1, alors votre fonction devra retourner 0.

Il est maintenant possible de réaliser un algorithme de complexité optimale utilisant la fonction précédente. En voici une brève description. Parcourons le tableau de gauche à droite et à chaque début de plage de 0 contigus, nous calculons la taille de cette plage puis repartons juste après elle. Ayant ainsi calculé la taille de toutes les plages de 0 il suffit de retenir celle de taille maximale. Sur l'exemple précédent, on part de l'indice 0 qui est un début de plage de 0 de taille 2. Nous continuons donc à chercher le prochain début de plage à partir de l'indice $0+2=2$. On examine ensuite l'indice 3 qui contient encore un 1 puis arrivons à l'indice 4 qui est le début d'une plage de 0 de taille 3. Nous allons donc chercher le prochaine début de plage à partir de l'indice $4+3=7$ etc.

11. Écrire une fonction `nombreZerosMaximum(tab:list[int])` qui renvoie le nombre maximal de 0 contigus en utilisant l'algorithme précédent. On attachera une attention particulière à ce que l'algorithme produit soit de complexité linéaire c'est à dire en temps $O(n)$.

Partie III. De la 1D vers la 2D

Cette partie consiste à développer un algorithme efficace pour détecter un rectangle d'aire maximale rempli de 0 dans un tableau bidimensionnel. Par mesure de simplification, nous travaillerons sur des tableaux carrés et nous supposerons donné un tableau *tab2* de taille $n \times n$. Un tel tableau bidimensionnel sera représenté par une `list[list[int]]`, avec `tab2[i][j]` l'élément de ligne i et colonne j.
Nous utiliserons le tableau suivant comme exemple :

```
#     j : 0  1  2  3  4  5  6  7     i
tab2 = [ [0, 0, 1, 1, 0, 0, 0, 1], # 0
         [0, 0, 0, 0, 1, 0, 0, 1], # 1
         [1, 0, 0, 0, 0, 0, 0, 1], # 2
         [0, 1, 0, 0, 0, 0, 0, 1], # 3
         [0, 0, 1, 1, 0, 0, 0, 1], # 4
         [0, 0, 1, 0, 0, 0, 0, 1], # 5
         [0, 1, 1, 1, 0, 1, 0, 1], # 6
         [0, 0, 1, 1, 0, 0, 0, 1] ]# 7
```

Méthode naïve. La première méthode proposée consiste à prendre une cellule (i,j) et à trouver un rectangle d'aire maximale dont le coin inférieur gauche est cette cellule. Remarquez que suivant l'orientation donnée dans le tableau exemple ci-dessus, un rectangle d'aire maximale de coin inférieur gauche la cellule (i,j) aura comme coin supérieur droit la cellule (i',j') avec $i' \leq i$ et $j' \geq j$. Par exemple, un rectangle d'aire maximale de coin inférieur gauche la cellule $(i,j)=(3,2)$ aura comme coin supérieur droit la cellule de coordonnées $(2,6)$.

12. Écrire une fonction `rectangleHautDroit(tab2:list[list[int]], i:int, j:int)` qui renvoie l'aire d'un rectangle d'aire maximale rempli de 0 et de coin inférieur gauche (i, j). **On cherchera la solution la plus efficace possible.** Pour $i = 3$ et $j = 2$ sur le tableau exemple ci-dessus, la fonction devra renvoyer la valeur 10.

13. Expliquer comment trouver naïvement un rectangle d'aire maximale rempli de 0 en utilisant la fonction `rectangleHautDroit`. Quelle serait la complexité de cette approche en fonction de n ?

Un peu de précalcul. Notre algorithme parcourt de nombreuses fois les même cases afin de vérifier la présence d'un 0 ou d'un 1. Nous allons donc effectuer avant notre algorithme un précalcul de certaines valeurs ce qui nous permettra, dans la partie IV, d'accélérer les fonctions précédentes.
Pour chaque cellule (i, j), nous allons calculer le nombre $c_{i,j}$ de cellules contiguës contenant 0 au dessus de la cellule (i, j), cette dernière étant comprise. Ce nombre est tel que les cellules (i, j), $(i-1, j)$, ..., $(i-c_{i,j}+1, j)$ contiennent 0 et la cellule $(i-c_{i,j}, j)$ contient 1 ou bien cette cellule n'existe pas. Ces valeurs $c_{i,j}$ seront rangées dans un tableau *col* représenté par une `list[list[int]]` Python de mêmes dimensions que le tableau initial. Le tableau col suivant est obtenu à partir du tableau exemple.

```
#     j : 0  1  2  3  4  5  6  7     i
col  = [ [1, 1, 0, 0, 1, 1, 1, 0], # 0
         [2, 2, 1, 1, 0, 2, 2, 0], # 1
         [0, 3, 2, 2, 1, 3, 3, 0], # 2
         [1, 0, 3, 3, 2, 4, 4, 0], # 3
         [2, 1, 0, 0, 3, 5, 5, 0], # 4
         [3, 2, 0, 1, 4, 6, 6, 0], # 5
         [4, 0, 0, 0, 5, 0, 7, 0], # 6
         [5, 1, 0, 0, 6, 1, 8, 0] ]# 7
```

14. Écrire une fonction `colonneZeros(tab2:list[list[int]])` qui renvoie un tableau bidimensionnel d'entiers *col* de mêmes dimensions que *tab2* et tel que `col[i][j]` donne le nombre de cellules contiguës contenant 0 au dessus de (i, j), cette cellule étant comprise. **On apportera un soin particulier à programmer la fonction la plus rapide possible.**

15. Quelle est la complexité de la fonction `colonneZeros` ?

Partie IV. Algorithme optimisé

Rectangle d'aire maximale dans un histogramme Une nouvelle étape dans notre algorithme réside dans la résolution du problème du calcul d'un rectangle d'aire maximale inscrit dans un histogramme. Nous supposerons donné un histogramme c'est à dire un tableau *histo* de taille n contenant des entiers. Chaque valeur représentera la hauteur d'une barre de l'histogramme. Par exemple le tableau suivant est associé à l'histogramme dessiné ci-dessous.

```
#    i : 0  1  2  3  4  5  6  7
histo = [3, 1, 2, 4, 2, 2, 0, 1]
```

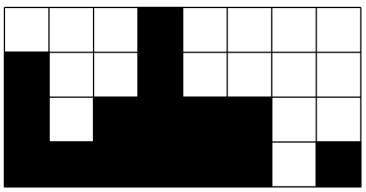

FIGURE 1 – Représentation graphique de l'histogramme `histo`

L'idée est de calculer un indice `L[i]` pour chaque colonne i, tel que `L[i]` est le plus petit entier j satisfaisant : $0 \leq j \leq i$ et `histo[k]` $\geq$ `histo[i]`, pour tout k tel que $j \leq k \leq i$.
Notons que $0 \leq$ `L[i]` $\leq i$. Dans notre exemple nous aurons `L[2] = 2` car la colonne 1 est strictement plus petite que la colonne 2. Nous aurons par ailleurs `L[5] = 2` car `histo[2]` $\geq$ `histo[5]`, `histo[3]` $\geq$ `histo[5]`, `histo[4]` $\geq$ `histo[5]` mais `histo[1]` $<$ `histo[5]`.

Nous pourrions calculer les valeurs `L[i]` de manière naïve mais cela ne permettrait pas d'améliorer notre algorithme. Nous allons donc procéder autrement.
On calcule successivement les valeurs de `L[i]` pour $i = 0 \dots n$. Pour calculer `L[i]` on pose $j = i$ et on fait décroître j tant que les colonnes sont plus grandes de la manière suivante :
Répéter :

- Si $j = 0$ alors affecter `L[i]` $= 0$ et terminer.
- Sinon :
  - Si `histo[j - 1]` $<$ `histo[i]` alors affecter `L[i] = j` et terminer.
  - Sinon (`histo[j - 1]` $\geq$ `histo[i]`) alors affecter `j = L[j - 1]` et continuer.

16. Sans justification, de quel paradigme algorithmique relève l'algorithme ci-dessus ?

17. Montrer que l'algorithme précédent calcule correctement les valeurs de L. De plus, montrer que l'algorithme fonctionne en temps $O(n)$ sur le cas particulier du tableau :

```
histo = [0, 1, 2, . . . , n - 2, n - 1, n - 2, . . . , 0]
```

de taille $2n-1$.

On supposera donnée une fonction `calculeL(histo:list[int])` qui renvoie en temps $O(n)$ le tableau `L` de taille n tel que `L[i]` est l'indice défini précédemment. On supposera aussi donnée une fonction équivalente `calculeR(histo:list[int])` qui renvoie en temps $O(n)$ un tableau `R` de valeurs `R[i]` $\geq$ i qui de manière analogue est l'indice maximal tel que `histo[k]` $\geq$ `histo[i]` pour tout k tel que $i \leq k \leq$ `R[i]`.

18. Justifier que pour tout i, le rectangle commençant à l'indice `L[i]`, terminant à l'indice `R[i]` et de hauteur `histo[i]` est inscrit dans l'histogramme.

19. Soit un rectangle d'aire maximale inscrit dans l'histogramme. Supposons que ce rectangle commence à l'indice ℓ, termine à l'indice r, et a pour hauteur h. Montrer qu'il existe $i \in [\![\ell, r]\!]$ tel que `histo[i]` $= h$, `L[i]` $= \ell$, et `R[i]` $= r$.

20. En déduire une fonction `plusGrandRectangleHistogramme(histo:list[int])` qui calcule et renvoie l'aire d'un plus grand rectangle inscrit dans l'histogramme. *Donner la complexité de votre fonction* `plusGrandRectangleHistogramme`.

Partie V. Conclusion

En revenant au problème initial du calcul d'un rectangle de 0 d'aire maximale dans une matrice en deux dimensions, remarquer que chaque ligne du tableau *col* calculé par la fonction `colonneZeros` de la question 14 peut être interprétée comme un histogramme.
En utilisant cette analogie, il est possible de proposer une méthode efficace de résolution du problème.

21. Écrire la fonction `rectangleToutZero(tab2:list[list[int]])` qui calcule un rectangle d'aire maximale rempli de 0 dans le tableau *tab2* et en renvoie son aire.

22. Quelle est la complexité de votre fonction ?

23. Indiquez comment la modifier afin qu'elle renvoie les coordonnées (i, j) d'un coin d'un rectangle atteignant l'aire maximale.

Fin du sujet