ÉCOLE POLYTECHNIQUE – ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

ÉCOLE SUPÉRIEURE DE PHYSIQUE ET DE CHIMIE INDUSTRIELLES

CONCOURS D'ADMISSION 2014 **FILIÈRE MP HORS SPÉCIALITÉ INFO**
FILIÈRE PC

COMPOSITION D'INFORMATIQUE – B – (XEC)

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Grands rectangles

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 (fonction ou procédure) 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 $\mathcal{O}(f(n))$, 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 candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Partie I. 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 dont les cases contiennent 0 ou 1. Nous supposerons que les indices du tableau sont compris entre **1** et **n**. Le but de cette partie est de trouver le nombre maximal de 0 contigüs (c'est à dire figurant dans des cases consécutives) dans le tableau.

Dans cette partie nous utiliserons le tableau suivant comme exemple :

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
tab[**i**]	0	0	1	1	0	0	0	1	0	0	0	1	1

Question 1 Écrire une fonction nombreZerosDroite(i, tab, n) prenant en paramètres le tableau tab de taille n ainsi qu'un indice **i** compris entre **1** et **n**. Cette fonction devra renvoyer le nombre de cases contigües 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 $\mathbf{i} = \mathbf{4}$ nous obtenons 0 et pour l'indice $\mathbf{i} = \mathbf{6}$ 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 contigüs, 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 **1** 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 $\mathbf{1} + \mathbf{2} = \mathbf{3}$. On examine ensuite l'indice **4** qui contient encore un 1 puis arrivons à l'indice **5** 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 $\mathbf{5} + \mathbf{3} = \mathbf{8}$ etc ...

Question 2 Écrire une fonction nombreZerosMaximum(tab, n) qui renvoie le nombre maximal de 0 contigüs 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 $\mathcal{O}(n)$.

Partie II. 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 tableau bidimensionnel sera adressé par $tab2$[**i**][**j**] et **i** représentera le numéro de ligne dans nos exemples.

Nous utiliserons le tableau suivant comme exemple :

i \ j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	0	1	1	0	0	0	1
2	0	0	0	0	1	0	0	1
3	1	0	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	0	1	1	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	1	1	1	0	1	0	1
8	0	0	1	1	0	0	0	1

Méthode naïve La première méthode proposée consiste à prendre une cellule $(\mathbf{i}, \mathbf{j})$ 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 $(\mathbf{i}, \mathbf{j})$ aura comme coin supérieur droit la cellule $(\mathbf{i'}, \mathbf{j'})$ avec $i' \leq i$ et $j' \geq j$. Par exemple, un rectangle d'aire maximale de coin inférieur gauche la cellule $(\mathbf{i}, \mathbf{j}) = (\mathbf{4}, \mathbf{3})$ aura comme coin supérieur droit la cellule de coordonnées $(\mathbf{3}, \mathbf{7})$.

Question 3 Écrire une fonction rectangleHautDroit$(tab2, n, \mathbf{i}, \mathbf{j})$ qui renvoie l'aire d'un rectangle d'aire maximale rempli de 0 et de coin inférieur gauche $(\mathbf{i}, \mathbf{j})$. On cherchera la solution la plus efficace possible. Pour $\mathbf{i} = \mathbf{4}$ et $\mathbf{j} = \mathbf{3}$ sur le tableau exemple ci-dessus, la fonction devra renvoyer la valeur 10.

Question 4 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 III, d'accélérer les fonctions précédentes.

Pour chaque cellule $(\mathbf{i}, \mathbf{j})$, nous allons calculer le nombre $c_{i,j}$ de cellules contigües contenant 0 au dessus de la cellule $(\mathbf{i}, \mathbf{j})$, cette dernière étant comprise. Ce nombre est tel que les cellules (i, j), $(i-1, j)$, $\ldots$, $(i - c_{i,j} + 1, j)$ 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*. Le tableau *col* suivant est obtenu à partir du tableau exemple.

i \ j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1	1	0	0	1	1	1	0
2	2	2	1	1	0	2	2	0
3	0	3	2	2	1	3	3	0
4	1	0	3	3	2	4	4	0
5	2	1	0	0	3	5	5	0
6	3	2	0	1	4	6	6	0
7	4	0	0	0	5	0	7	0
8	5	1	0	0	6	1	8	0

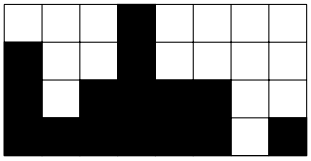
Question 5 Écrire une fonction colonneZeros$(tab2, n)$ qui renvoie un tableau bidimensionnel d'entiers *col* de taille $n \times n$ et tel que $col[\mathbf{i}][\mathbf{j}]$ donne le nombre de cellules contigües contenant 0 au dessus de $(\mathbf{i}, \mathbf{j})$, cette cellule étant comprise. On apportera un soin particulier à programmer la fonction la plus rapide possible.

Question 6 Quelle est la complexité de la fonction colonneZeros ?

Partie III. 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 à sa droite.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
histo[**i**]	3	1	2	4	2	2	0	1



L'idée est de calculer un indice $L[\mathbf{i}]$ pour chaque colonne **i**, tel que $L[\mathbf{i}]$ est le plus petit entier **j** satisfaisant : $\mathbf{1} \leq \mathbf{j} \leq \mathbf{i}$ et $histo[\mathbf{k}] \geq histo[\mathbf{i}]$, pour tout **k** tel que $\mathbf{j} \leq \mathbf{k} \leq \mathbf{i}$.

Notons que $\mathbf{1} \leq L[\mathbf{i}] \leq \mathbf{i}$. Dans notre exemple nous aurons $L[\mathbf{3}] = 3$ car la colonne **2** est strictement plus petite que la colonne **3**. Nous aurons par ailleurs $L[\mathbf{6}] = \mathbf{3}$ car $histo[\mathbf{3}] \geq histo[\mathbf{6}], histo[\mathbf{4}] \geq histo[\mathbf{6}], histo[\mathbf{5}] \geq histo[\mathbf{6}]$ mais $histo[\mathbf{2}] < histo[\mathbf{6}]$.

Nous pourrions calculer les valeurs $L[\mathbf{i}]$ de manière naïve mais cela ne permettrait pas d'améliorer notre algorithme. Nous allons donc procéder autrement.

On calcule successivement les valeurs de $L[\mathbf{i}]$ pour $\mathbf{i} = \mathbf{1} \ldots \mathbf{n}$. Pour calculer $L[\mathbf{i}]$ on pose $\mathbf{j} = \mathbf{i}$ et on fait décroître **j** tant que les colonnes sont plus grandes de la manière suivante :

Répéter :
- Si $\mathbf{j} = \mathbf{1}$ alors affecter $L[\mathbf{i}] = \mathbf{1}$ et **terminer**.
- Sinon :
  - Si $histo[\mathbf{j} - \mathbf{1}] < histo[\mathbf{i}]$ alors affecter $L[\mathbf{i}] = \mathbf{j}$ et **terminer**.
  - Sinon $histo[\mathbf{j} - \mathbf{1}] \geq histo[\mathbf{i}]$ alors affecter $\mathbf{j} = L[\mathbf{j} - \mathbf{1}]$ et **continuer**.

Question 7 Montrer que l'algorithme précédent calcule correctement les valeurs de L. De plus, montrer que l'algorithme fonctionne en temps $\mathcal{O}(n)$ sur le cas particulier du tableau $histo = [1, 2, 3, \ldots, n-1, n, n, n-1, \ldots, 1]$ de taille $2n$.

On supposera donnée une fonction $\mathsf{calculeL}(histo, n)$ qui renvoie en temps $\mathcal{O}(n)$ le tableau L de taille n tel que $L[\mathbf{i}]$ est l'indice défini précédemment. On supposera aussi donnée une fonction équivalente $\mathsf{calculeR}(histo, n)$ qui renvoie en temps $\mathcal{O}(n)$ un tableau R dont les valeurs $R[\mathbf{i}] \geq \mathbf{i}$ qui de manière analogue est l'indice maximal tel que $histo[\mathbf{k}] \geq histo[\mathbf{i}]$ pour tout **k** tel que $\mathbf{i} \leq \mathbf{k} \leq R[\mathbf{i}]$.

Question 8 Justifier que pour tout **i**, le rectangle commençant à l'indice $L[\mathbf{i}]$, terminant à l'indice $R[\mathbf{i}]$ et de hauteur $histo[\mathbf{i}]$ est inclus dans l'histogramme.

Question 9 Soit un rectangle d'aire maximale inscrit dans l'histogramme. Supposons que ce rectangle commence à l'indice $\mathbf{l}$, termine à l'indice $\mathbf{r}$, et a pour hauteur h. Montrer qu'il existe $\mathbf{i} \in [\mathbf{l} \dots \mathbf{r}]$ tel que $histo[\mathbf{i}] = h$, $L(\mathbf{i}) = \mathbf{l}$, et $R(\mathbf{i}) = \mathbf{r}$.

Question 10 En déduire une fonction plusGrandRectangleHistogramme$(histo, n)$ qui calcule et renvoie l'aire d'un plus grand rectangle inscrit dans l'histogramme. Donner la complexité de votre fonction plusGrandRectangleHistogramme.

Partie IV. 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 5 peut être interprétée comme un histogramme.

En utilisant cette analogie, il est possible de proposer une méthode efficace de résolution du problème.

Question 11 Écrire la fonction rectangleToutZero$(tab2, n)$ qui calcule un rectangle d'aire maximale rempli de 0 dans le tableau $tab2$ et en renvoie son aire.

Question 12 Quelle est la complexité de votre fonction ?

* *
*