

DL 1

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 coordonnées 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 d'un algorithme dont la taille des données est n s'exprime en $\mathcal{O}(f(n))$, c'est-à-dire qu'il existe une constante $K > 0$ telle que le nombre d'opérations élémentaires nécessaires à l'exécution du programme soit inférieur à $Kf(n)$.

Partie I. Recherche unidimensionnelle

On étudie dans cette partie le problème de reconnaissance de forme sur des tableaux unidimensionnels. On suppose donnés un entier n et un tableau `tab` de longueur n (indexé de 0 à $n - 1$) dont les cases contiennent 0 ou 1. On cherche à déterminer le nombre maximal de 0 figurant dans des cases consécutives du tableau.

Par exemple, pour le tableau suivant :

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$tab.(i)$	0	0	1	1	0	0	0	1	0	0	0	1	1

`nombreZerosDroite 5 tab = 2` et `nombreZerosMaximum tab = 3`.

Question 1 Ecrire une fonction `nombreZerosDroite : int -> int array -> int` telle que `nombreZerosDroite i tab` renvoie le nombre de cases contiguës du tableau contenant 0 à droite de la case d'indice i (comprise).

Question 2 Ecrire une fonction `nombreZerosMaximum : int array -> int` calculant le nombre maximal de 0 contiguës. Cette fonction devra avoir une complexité linéaire par rapport à n (c'est-à-dire en $\mathcal{O}(n)$).

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. On utilisera le tableau suivant comme exemple :

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	1	1	0	0	0	1
1	0	0	0	0	1	0	0	1
2	1	0	0	0	0	0	0	1
3	0	1	0	0	0	0	0	1
4	0	0	1	1	0	0	0	1
5	0	0	1	0	0	0	0	1
6	0	1	1	1	0	1	0	1
7	0	0	1	1	0	0	0	1

Méthode naïve

La première méthode consiste à prendre une cellule (i, j) et à trouver un rectangle d'aire maximale dont le coin inférieur gauche est cette cellule. Suivant l'orientation proposée, un rectangle d'aire maximale de coin inférieur gauche (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 $(2, 6)$.

Question 3 Ecrire une fonction `rectangleHautDroit : int array array -> int -> int -> 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.

Par exemple, pour $i = 3$ et $j = 2$, 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

L'algorithme précédent parcourt de nombreuses fois les mêmes cases afin de vérifier la présence d'un 0 ou d'un 1. On va donc effectuer un précalcul de certaines valeurs pour pouvoir 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) , celle-ci comprise. Ce nombre est tel que les cellules $(i, j), (i-1, j), \dots, (i-c_{i,j}+1, j)$ contiennent 0 et la cellule $(i-c_{i,j}, j)$ contient 1 ou n'existe pas. Ces valeurs $c_{i,j}$ seront rangées dans un tableau `col`. Dans l'exemple précédent, le tableau `col` est le suivant :

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

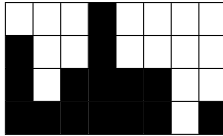
Question 5 Ecrire une fonction `colonneZeros : int array array -> int array array` qui renvoie un tableau bidimensionnel d'entiers dont la case (i, j) contient le nombre de cellules contiguës contenant 0 au-dessus de (i, j) , celle-ci étant comprise. On s'efforcera de programmer la fonction la plus rapide possible.

Indiquer la complexité de la fonction `colonneZeros`.

Partie III. Algorithme optimisé

Une nouvelle étape dans l'algorithme réside dans la résolution du problème du calcul d'un rectangle d'aire maximale inscrit dans un histogramme. On suppose 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. On l'illustrera avec l'exemple suivant :

i	0	1	2	3	4	5	6	7
<code>histo[i]</code>	3	1	2	4	2	2	0	1



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 l'exemple proposé, $L[2] = 2$, $L[5] = 2$.

L'idée consiste à calculer successivement les valeurs de $L[i]$ pour i allant de 0 à $n-1$. 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.
 - Si $histo[j-1] \geq histo[i]$ alors affecter $j = L[j-1]$ et continuer.

Question 7 Montrer que l'algorithme précédent calcule correctement les valeurs de L .

Justifier que l'algorithme fonctionne en temps $\mathcal{O}(n)$ dans le cas particulier du tableau `histo` = $[1, 2, 3, \dots, n-1, n, n, n-1, \dots, 2, 1]$ de taille $2n$.

Question 8 Ecrire une fonction `calculeL : int array -> int array` qui renvoie en temps $\mathcal{O}(n)$ le tableau L de taille n tel que $L[i]$ est l'indice défini précédemment.

On supposera aussi connue une fonction `calculeR : int array -> int array` qui renvoie en temps $\mathcal{O}(n)$ le tableau R de taille n tel que $R[i]$ est l'indice j maximal tel que $j \geq i$ et $histo[k] \geq histo[i]$ pour tout k tel que $i \leq k \leq R[i]$.

Question 9 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 inclus dans l'histogramme.

Question 10 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 $i \in \{l, \dots, r\}$ tel que $histo[i] = h$, $L[i] = l$ et $R[i] = r$.

Question 11 En déduire une fonction `plusGrandRectangleHistogramme : int array -> int` qui calcule l'aire d'un plus grand rectangle inscrit dans l'histogramme. Indiquer la complexité de cette fonction.

Partie IV. Conclusion

En revenant au problème initial du calcul d'un rectangle de 0 d'aire maximale dans une matrice en deux dimensions, on remarque 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 12 Ecrire une fonction `rectangleToutZero : int array array -> int` qui détermine un rectangle d'aire maximale rempli de 0 dans le tableau et en renvoie son aire.

Question 13 Indiquer la complexité de cette fonction.