Épreuve d'informatique de l'X - 2014 - MP/PC

Grands rectangles

Partie I. Recherche unidimensionnelle

Question 1. On se place dans la i^{e} case du tableau. Ensuite, on se déplace vers la droite tant que l'on rencontre des zéros et que l'on n'est pas arrivé au bout du tableau (test de la boucle `while` en ligne 3). En ligne 7, on retourne le nombre de cases contiguës du tableau contenant 0 à droite de la case d'indice i.

```
1 nombreZerosDroite := proc(i,tab,n)
2   local compteur := 0, k := i;
3   while k <= n and tab[k] = 0 do
4      # On incrèmente le compteur de zéros
5      compteur := compteur +1;
6      k := k+1;
7   od;
8   return(compteur);
9 end;
```

Question 2. On part de la position $i = 1$. La variable `res` va contenir le résultat. À un instant donné, elle contient la longueur de la plus grande plage de zéros contigus de la partie du tableau explorée.
Tant que l'on n'a pas atteint le bout du tableau (boucle des lignes 3 à 7), on compte le nombre de zéros à droite de la position i (ligne 4). Si ce nombre est plus grand que `res` (donc plus grand que le nombre de zéros contigus de la partie du tableau déjà explorée), on réactualise la variable `res`.
La quantité $i+nbZeros$ est plus grande que n ou bien correspondant à une case contenant un 1. On doit donc continuer l'exploration à partir de la position $i + nbZeros + 1$. On actualise la variable i en conséquence (ligne 6).
Chaque case du tableau est au plus visitée une seule fois, la complexité en bien linéaire.

(PSI*)

```
1 nombreZerosMaximum := proc(tab,n)
2   local res := 0, i := 1, nbZeros;
3   while i <= n do
4     nbZeros := nombreZerosDroite(i,tab,n);
5     if nbZeros > res then res := nbZeros fi;
6     i := i+nbZeros+1
7   od;
8   return(res);
9 end;
```

Partie II. De la 1D vers la 2D

Question 3 : On veut déterminer l'aire d'un rectangle d'aire maximale rempli de zéros dont le coin inférieur gauche est (i, j). On part de la position (i, j). Les lignes 3 et 4 permettent de déterminer l'aire et la largeur du plus grand rectangle rempli de zéros, de hauteur 1 dont le coin inférieur gauche est (i, j). Tant que l'on n'a pas atteint le bout du tableau et qu'il y a bien un zéro en position (k, j) (boucle des lignes 5 à 12), on compte le nombre de zéros à droite de la position j en k^{e} ligne du tableau (ligne 6). On détermine la largeur maximale du rectangle rempli de zéros, de hauteur $i - k + 1$ et de coin inférieur gauche (i, j) (ligne 7 et 8). Si l'aire de ce rectangle (calculée à la ligne 9) est plus grande que l'aire maximale des rectangles précédemment étudiés, on actualise la variable `surface` (ligne 10). En sortie de boucle, la variable `surface` contient le résultat attendu. [1]

```
1  rectangleHautDroit := proc(tab2,n,i,j)
2      local Surface, k := i-1, nbZerosLigne, SurfaceTmp, Largeur;
3      Largeur := nombreZerosDroite(j,tab2[i],n);
4      Surface := Largeur;
5      while(k > 0 and tab2[k][j] = 0) do
6         nbZerosLigne := nombreZerosDroite(j,tab2[k],n);
7         if nbZerosLigne < Largeur
8               then Largeur := nbZerosLigne fi;
9         SurfaceTmp := Largeur * (i-k+1);
10        if SurfaceTmp > Surface then Surface := SurfaceTmp fi;
11        k := k - 1;
12     od;
13     return(Surface)
14 end;
```

1. On a supposé que tab2[k] retournait bien la k^{e} ligne du tableau `tab2`.

Question 4 : Dans l'approche naïve, pour chaque cellule (i, j), on calcule l'aire d'un rectangle d'aire maximale rempli de 0 et de coin inférieur gauche (i, j) à l'aide de la fonction `rectangleHautDroit`. Parmi toutes les surfaces calculées, on garde la plus grande.
Le coût de la fonction `rectangleHautDroit` pour le sommet (i, j) est en $\mathcal{O}(i(n-j+1))$ (on a $i(n-j+1)$ cases à explorer au maximum). Le coût total est donc $\mathcal{O}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} i(n-j+1)\right) = \mathcal{O}\left(\left(\frac{n(n+1)}{2}\right)^2\right) = \mathcal{O}(n^4)$.

Le programme suivant applique la méthode décrite. La variable `surfaceMax` contient la plus grande des surfaces parmi les rectangles associés aux sommets explorés.

```
1  Surface := proc(tab2,n)
2      local surfaceMax := 0, i, j,tmp;
3      for i from 1 to n do
4          for j from 1 to n do
5            tmp := rectangleHautDroit(tab2,n,i,j);
6            if tmp > surfaceMax then surfaceMax := tmp fi;
7          od;
8      od;
9      return(surfaceMax);
10 end;
```

Question 5 : On remarque que dans le tableau *tab*, à partir de la deuxième ligne, le nombre de cellules contiguës contenant 0 au dessus de (i, j) est égal à 0 si il y a un 1 en position (i, j) et est égal au nombre de cellules contiguës contenant 0 au dessus de $(i-1, j)$ augmenté de 1 sinon.
On exploite la remarque précédente dans le programme `colonneZeros`. On crée une matrice vide de taille $n \times n$ en ligne 3. On commence par remplir la première ligne du tableau (boucle des lignes 4 à 9). Le nombre de zéros au dessus de la position $(1, j)$ est égal à 1 s'il y a un zéro en position $(1, j)$ et est égal à 0 sinon.
Ensuite, on remplit le tableau ligne par ligne en utilisant la propriété mis en évidence précédemment (boucle des lignes 10 à 17). On retourne le tableau résultat en ligne 18.

```
1  colonneZeros := proc(tab2,n)
2      local col, i, j;
3      col := Matrix(n,n);
4      for j from 1 to n do
5          if tab2[1,j]=0
6              then col[1,j] := 1
7              else col[1,j] := 0
8          fi;
9      od;
10     for i from 2 to n do
11         for j from 1 to n do
12             if evalb(tab2[i,j]=0)
13                 then col[i,j] := 1 + col[i-1,j]
14                 else col[i,j] := 0
15             fi;
16         od;
17     od;
18     return(col);
19 end;
```

Question 6 : On explore une et une seule fois les cases du tableau `tab2` et on effectue au plus deux opérations élémentaires par cases. La complexité du programme `colonneZeros` est optimale en $\mathcal{O}(n^2)$.

Partie III. Algorithme optimisé

Question 7 : L'algorithme est décrit par :

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

Dans la boucle « Répéter », soit on s'arrête soit j reçoit $L[j-1]$ et par définition $L[j-1] \leqslant j-1$, donc la suite des j étudiés pour un i donné est strictement décroissante. La boucle s'arrête en un nombre fini d'étapes.

On veut montrer que le $L[i]$ calculé par la boucle est correct. On suppose que $L[1]$,...,$L[i-1]$ ont été calculés correctement.

Pour montrer l'exactitude d'une boucle, il faut mettre en évidence un « invariant de boucle », c'est-à-dire une propriété qui reste vraie à chaque passage dans la boucle.

On considère la propriété $\mathcal{P} : \forall k \in [\![j, i]\!],\ histo[k] \geqslant histo[i]$.

On suppose la propriété vraie à l'entrée de la boucle.

- Si $j = 1$ et $k \in [\![j, i]\!],\ histo[k] \geqslant histo[i]$ alors $L[i] = 1$, le programme s'arrête et retourne la bonne valeur pour $L[i]$.
- Si $j > 1$ et $histo[j-1] < histo[i]$ comme $\forall k \in [\![j, i]\!],\ histo[k] \geqslant histo[i]$. $L[i] = j$ est bien le plus petit entier j' satisfaisant $1 \leqslant j' \leqslant i$ et $\forall k \in [\![j', i]\!],\ histo[k] \geqslant histo[i]$.
  Le programme s'arrête et retourne la bonne valeur.
- Si $j > 1$ et $histo[j-1] > histo[i]$.
  On a $\forall k \in [\![L[j-1], j-1]\!],\ histo[k] \geqslant histo[j-1] \geqslant histo[i]$ et $\forall k \in [\![j, i]\!],\ histo[k] \geqslant histo[i]$. Donc $\forall k \in [\![L[j-1], i]\!],\ histo[k] \geqslant histo[i]$, en donnant à j la valeur $L[j-1]$ la propriété est toujours satisfaite.

La boucle va donc s'arrêter en donnant la bonne valeur à $L[i]$.

conclusion : L'algorithme calcule correctement les valeurs de L.

Pour $histo = [1, 2, 3, ..., n, n, n-1, ..., 2, 1]$:

- Pour $i = 2$ à n, comme $histo[i-1] < histo[i]$, la boucle conditionnelle n'est exécutée qu'une seule fois et $L[i]$ reçoit i.
- Pour $i = n+1$, comme $histo[n] = histo[n+1]$ et que l'on a l'inégalité $histo[n-1] = n-1 < n = histo[n+1]$, la boucle conditionnelle n'est exécutée que deux fois et $L[n+1]$ reçoit n.
- Pour $i = n+2$ à $2n$, le calcul de $L[i]$ se fait en au plus trois étapes. j reçoit la valeur $L[i-1] = 2n+2-i$, puis $L[2n+1-i] = 2n+1-i$. Puis $L[i]$ reçoit $2n+1-i$ et la boucle conditionnelle s'arrête.

conclusion :

L'algorithme fonctionne en $\mathcal{O}(n)$ sur le cas particulier donné par l'énoncé.

Voici un codage possible (non demandé par l'énoncé) de la fonction `calculeL`.

```
1  calculeL := proc(histo,n)
2     local i, j, L, ok;
3     L := array(1..n);
4     for i from 1 to n do
5          j := i;
6          ok := true;
7          while ok do
8             if j=1
9                then L[i] := 1; ok := false;
10               else if histo[j-1] < histo[i]
11                         then L[i] := j; ok := false
12                         else j := L[j-1]
13                    fi;
14            fi;
15         od;
16    od;
17    return(L);
18 end;
```

Question 8 : Par définition de $L[i]$ et $R[i]$, on a pour tout j dans $[\![L[i], R[i]]\!]$, $histo[j] \geqslant histo[i]$ donc 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 9 : Si le rectangle d'aire maximale commence à l'indice ℓ, termine à l'indice r et a pour hauteur h alors nécessairement $\forall j \in [\![\ell, r]\!],\ histo[j] \geqslant h$. Si $\forall j \in [\![\ell, r]\!],\ histo[j] \geqslant h+1$ alors le rectangle commençant à l'indice ℓ, terminant à l'indice r et ayant pour hauteur $h+1$ serait composé uniquement de zéros et aurait une aire strictement plus grande que le rectangle d'aire maximale, ce qui est contradictoire. Donc, $\exists i \in [\![\ell, r]\!] \,/\, histo[i] < h+1$. Pour un tel indice i_0, par double inégalité, on a $histo[i_0] = h$.
Comme $\forall i \in [\![\ell, r]\!],\ histo[i] \geqslant h = histo[i_0]$, par définition de $L[i_0]$ et $R[i_0]$, on a $L(i_0) \leqslant \ell$ et $R[i_0] \geqslant r$. Si $L[i_0] < \ell$ ou $R[i_0] > r$ alors le rectangle commençant à l'intervalle $L[i_0]$, terminant à l'indice $R[i_0]$ et de hauteur $h = histo[i_0]$ serait inclus dans l'histogramme et aurait une aire strictement plus grande que le rectangle d'aire maximal, ce qui est contradictoire.

Donc $\boxed{\exists i_0 \in [\![\ell, r]\!] \,/\, histo[i_0] = h,\ L[i_0] = \ell \text{ et } R[i_0] = r.}$

Question 10 : On commence par calculer les tableaux R et L associés à l'histogramme $histo$ (ligne 3 et 4). Ensuite, pour chaque i de $[\![1, n]\!]$, on calcule l'aire du rectangle de hauteur $h[i]$ entre les indices $L[i]$ et $R[i]$. Ce rectangle est inclus dans l'histogramme, d'après la question 8 et d'après la question 9 l'aire du plus grand rectangle est l'aire de l'un de ces rectangles. La boucle des lignes 5 à 8 permet de calculer l'aire du plus grand des rectangles.

```
1  plusGrandRectangleHistogramme:=proc(histo,n)
2      local L, R, i, S, Smax := 0;
3      L := calculeL(histo,n);
4      R := calculeR(histo,n);
5      for i from 1 to n do
6        S := histo[i]*(R[i]-L[i]+1);
7        if S > Smax then Smax := S; fi;
8      od;
9      return(Smax);
10 end;
```

Les calculs des tableaux L et R se font en $\mathcal{O}(n)$. Dans la boucle des lignes 5 à 8, le nombre d'opérations élémentaires est inférieur à 6 donc le coût de la boucle est en $\mathcal{O}(n)$. Donc, la complexité de la fonction `plusGrandRectangleHistogramme` est en $\mathcal{O}(n)$.

Partie IV. Conclusion

Question 11 : On commence par calculer le tableau `col` définie à la question 5 avec la fonction `colonneZeros` (ligne 3). Chaque ligne du tableau `col` est traitée comme un histogramme. Dans la boucle des lignes 4 à 7, on recherche l'aire du plus grand rectangle ayant son coin inférieur gauche en i^{e} ligne (ligne 5). Parmi toutes les aires calculées, on garde la plus grande (ligne 6). À la ligne 8, on retourne donc l'aire du plus grand rectangle rempli de zéros.

```
1  rectangleToutZero := proc(tab2,n)
2      local col, i, S, Smax := 0;
3      col := colonneZeros(tab2,n);
4      for i from 1 to n do
5          S := plusGrandRectangleHistogramme(col[i],n);
6          if S > Smax then Smax := S fi;
7      od;
8      return(Smax)
9  end;
```

Question 12 : La complexité de la fonction `colonneZeros` en ligne 3 est en $\mathcal{O}(n^2)$. La boucle des lignes 4 à 6 est parcourue n fois et le coût de la fonction `plusGrandRectangleHistogramme` est en $\mathcal{O}(n^2)$ donc le coût global de la boucle est en $\mathcal{O}(n^2)$. On en déduit que la complexité de la fonction `rectangleToutZero` est en $\mathcal{O}(n^2)$.

E. Le Nagard - XMPPC2014cor (composé avec LaTeX le 16/6/2014)