DS3 MP2I : Grands rectangles

OCaml impératif ; Bash

Aucun appareil électronique n'est autorisé.

Solution. □

1 Bash

Question 1.

Répondre aux questions suivantes :

1. Quelle commande (la plus concise possible) utiliser pour revenir à la racine de son `home` ?

   Solution. `cd` □

2. Quelle commande utiliser pour se placer dans le répertoire `Documents` (situé dans le répertoire personnel) ?

   Solution. `cd Documents` □

3. On est dans un répertoire contenant des fichiers et sous-répertoires. Quelle commande permet d'afficher la liste détaillée (droits, propriétaire, taille, date) en incluant les fichiers cachés ?

   Solution. `ls -al` □

4. On souhaite créer un lien symbolique nommé `raccourci` pointant vers le fichier `/usr/bin/python3`. Quelle commande utiliser ?

   Solution. `ln -s /usr/bin/python3 raccourci` □

5. On souhaite créer un lien physique nommé `copie` vers le fichier `rapport.txt` (dans le répertoire courant). Quelle commande utiliser ?

   Solution. `ln rapport.txt copie` □

6. Dans `/home`, on cherche tous les **répertoires** dont le nom contient **au moins 5 lettres** (minuscules ou majuscules), et **ne commence pas par un chiffre**. Écrire une commande qui affiche la liste des chemins trouvés.

   Solution. `find /home -type d -name '[!0-9]*[a-zA-Z]*[a-zA-Z]*[a-zA-Z]*[a-zA-Z]*'` □

7. On veut écrire la chaîne `Bonjour` dans le fichier `message.txt` **en écrasant** son contenu s'il existe déjà. Quelle commande utiliser ?

   Solution. `echo Bonjour > message.txt` □

8. On veut ajouter la chaîne `Bonjour` à la fin du fichier `message.txt` **sans effacer** son contenu. Quelle commande utiliser ?

   Solution. `echo Bonjour » message.txt` □

9. On veut afficher la liste des fichiers du répertoire courant mais ne garder que les fichiers qui sont des répertoires. On utilise obligatoirement une commande de listing avec la bonne option et un pipe vers une commande de recherche de motif.
   Contrainte : aucune commande dont le nom commence par un `f`.

   Solution. `ls -l | grep '^d'` □

10. Le miniscript `miniscript.sh` est fourni mais n'est pas exécutable. Quelle commande utiliser pour le rendre exécutable par son propriétaire ?

    Solution. `chmod u+x miniscript.sh` □

2 Grands rectangles

Ce problème 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.

Sauf mention du contraire, le nombre n désigne dans tout ce qui suit la longueur d'un tableau.

Une image en noir et blanc et décrite par une matrice de 0 et 1. La valeur 0 désigne une case noire et, 1, une case blanche.

Les complexités demandées sont toujours des complexité temporelles au pire.

2.1 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 de taille n dont les cases contiennent 0 ou 1. 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.

Nous utilisons le tableau suivant comme exemple :

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

Question 2.

Écrire une fonction `nombreZerosDroite (i:int) (tab:int array) : 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 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 $\max\{ji+1 | i \leq j < n \text{ et } \forall k \in [i,j], tab.(k) = 0\}$

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.

Solution. Code

```
let nombreZerosDroite (i:int) (tab:int array) : int =
  let n = Array.length tab in
  let j = ref 0 in
  while i + !j < n && tab.(i + !j) = 0 do
    incr j
  done;
  !j
;; (*O(n-i)*)
```

□

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, nous partons 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 $0 + 2 = 2$.

Nous examinons 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.

Question 3.

Écrire une fonction `nombreZerosMaximum (tab :int array): int` 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 en temps $O(n)$

Solution. Code

```
(* complexité en O(n) car chaque élément n'est parcouru qu'une fois *)
let nombreZerosMaximum (tab :int array): int =
  let i = ref 0 in
  let m = ref 0 in
  let n_tab = Array.length tab in
  while !i < n_tab do
    let n = nombreZerosDroite !i tab in
    if n > !m then
      m := n;
    i := !i + n + 1
  done;
  !m
;;
```

□

2.2 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 tra vaillerons sur des tableaux carrés et nous supposerons donné un tableau *tab*2 de taille $n \times n$.

Nous utiliserons le tableau suivant comme exemple :

i/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	1	0	0	0	1
6	0	1	1	1	0	1	0	1
7	0	0	1	1	0	0	0	1

Listing 1 – Traduction en OCaml

```
let tab2 = [|
  [| 0; 0; 1; 1; 0; 0; 0; 1 |];
  [| 0; 0; 0; 0; 1; 0; 0; 1 |];
  [| 1; 0; 0; 0; 0; 0; 0; 1 |];
  [| 0; 1; 0; 0; 0; 0; 0; 1 |];
  [| 0; 0; 1; 1; 0; 0; 0; 1 |];
  [| 0; 0; 1; 1; 0; 0; 0; 1 |];
  [| 0; 1; 1; 1; 0; 1; 0; 1 |];
  [| 0; 0; 1; 1; 0; 0; 0; 1 |]
|]
;;
```

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 $(2, 6)$.

Question 4.

Écrire une fonction `rectangleHautDroit (tab2: int array array) (i:int) (j: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. Pour $i = 3$ et $j = 2$ sur le tableau exemple ci-dessus, la fonction devra renvoyer la valeur 10.

Solution.

```
(*
on parcourt la ligne en cherchant le nb de zeros à droite
on parcourt les lignes au dessus en allant le plus loin possible à droite
on calcule pour chaque ligne l'aire du rectangle
on prend le max des aires trouvées
*)
let rectangleHautDroit (tab2: int array array) (i:int) (j:int):int =
  if tab2.(i).(j) = 1 then  0
  else begin
      (* m : aire maximale trouvée
         nz : nombre de zéros à droite courant *)
      let nz = nombreZerosDroite j tab2.(i) in  (* O(n-j) *)
      let m = ref nz and
          nz_ref = ref nz in
      try
        (* on parcourt les lignes au-dessus *)
        for ligne = i - 1 downto 0 do (* i passages *)
          let h = i - ligne + 1 (*hauteur*) in
          (* on met à jour le nombre de zéros à droite *)
          nz_ref :=
            min (nombreZerosDroite j tab2.(ligne)) !nz_ref;  (* O(n-j) *)
          let a = h * !nz_ref (*aire*) in
          if a > !m then
            m := a
          else if a = 0 then
            (* plus aucun rectangle possible *)
            raise Exit
        done;
        raise Exit
      with Exit -> !m
    end
;;
```

n désigne la longueur des lignes.

Dans le pire des cas on entre dans la boucle `for` et on la parcourt jusqu'au bout : i passages. Pour chaque passage, un calcul de nombre de zéros à droite en $O(n - j)$ plus quelques calculs de coûts négligeables.

Donc complexité en $O(i(n - j))$. □

Question 5.

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 ?

Solution. Chaque calcul de rectangle en position (i, j) a une complexité $O(i(n - j)$ qu'on majore ne $O(n^2)$. Or il y a n^2 coins inférieurs droits de sous-rectangles. Donc on trouve une complexité $O(n^4)$.

On parcourt les n^2 cases.

Pour chacune, on calcule rectangleHautDroit(tab,i,j), majoré en $O(n^2)$ (grossièrement)
Alors $O(n^4)$ majore la complexité. Mais ce n'est peut-être pas aussi gros.
Plus finement : on sait que $O(i(n-j))$ est la complexité de l'appel `rectangleHautDroit tab i j`.

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} i(n-j) = \sum_{i=0}^{n-1}\sum_{k=1}^{n} ik = \sum_{i=0}^{n-1} i\sum_{k=1}^{n} k = \frac{n(n-1)}{2} \times \frac{n(n+1)}{2}$$

Et on obtient le produit de deux polynômes de degré 2 en n. D'où un $O(n^4)$. □

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 (i, j), nous allons calculer le nombre $c_{i,j}$ de cellules contigües contenant 0 au dessus de la cellule (i, j), cette dernière étant comprise. Ce nombre est tel que les cellules $(i, j), (i-1, j), \ldots, (i-c_{i,j}+1, j)$ 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	**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	0	4	6	6	0
6	4	0	0	0	5	0	7	0
7	5	1	0	0	6	1	8	0

Question 6.

Écrire une fonction `colonneZeros(tab:int array array):int array array` qui prend en paramètre *tab*, un tableau de taille $n \times n$ et renvoie un tableau bidimensionnel d'entiers *col* de taille $n \times n$ et tel que $col.(i).(j)$ donne le nombre de cellules contigües contenant 0 au dessus de (i, j), cette cellule étant comprise.

On apportera un soin particulier à programmer la fonction la plus rapide possible.

Solution. Code

```
(*
on peut créer res un tableau de zéros sauf la première ligne
qui contient 1 - tab.(0).(j) pour toute colonne
puis dans la boucle, on remplit la ligne i+1 en examinant la ligne i
*)
let colonneZeros tab =
  let n = Array.length tab in
  let m = Array.length tab.(0) in

  (* création de res, tableau n x m initialisé à 0 *)
  let res = Array.make_matrix n m 0 in

  (* première ligne : 1 si tab.(0).(j) = 0, sinon 0 *)
  for j = 0 to m - 1 do
    res.(0).(j) <- if tab.(0).(j) = 0 then 1 else 0
  done;

  (* remplissage des lignes suivantes *)
  for i = 1 to n - 1 do
    for j = 0 to m - 1 do
      if tab.(i).(j) = 0 then
        res.(i).(j) <- res.(i - 1).(j) + 1;
```

```
    done
  done;

  res
;;
```

□

Question 7.

Quelle est la complexité de l'appel `colonneZeros tab` ?

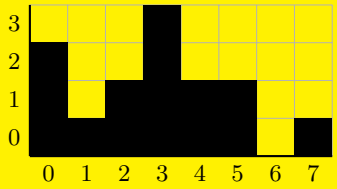
Solution. Double boucle, le corps de boucle est en $O(1)$. par conséquent, complexité quadratique. □

2.3 Algorithme optimisé

Rectangle d'aire maximale dans un histogramme Les *histogrammes* sont des colonnes de pixels noirs partant d'une même droite horizontale dans un tableau de pixels noirs et blancs. On dit qu'un rectangle (noir) *est inclus dans un l'histogramme* H s'il est uniquement constitué de pixels noirs de H formant un rectangle.

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 : on peut le décrire par un tableau *histo* de taille n contenant des entiers et décrivant la hauteur des colonnes (rappel : les colonnes sont noires).

Ci-dessous, on représente un histogramme et son implémentation par un tableau d'entiers :



Listing 2 – Implémentation de l'histogramme ci-dessus

```
let histo = [| 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 $0 \leq l.(i) \leq i$. L'ensemble des indices $l.(i)$ forme un tableau que nous allons calculer.

Dans notre exemple nous avons $l.(2) = 2$ 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 $histo.(1) < histo.(5)$.

Adoptons la convention suivante : Si $0 \leq x \leq z \leq n$ et $0 \leq y \leq t \leq n$, on note $R((x, y), (z, t))$ le rectangle dont le coin inférieur gauche est la case (x, y) et le coin supérieur droit est (z, t). La valeur $l.(i)$ est alors le numéro de la colonne la plus à gauche de i telle que le rectangle (de hauteur $histo.(i)$) $R((l.(i), 0), (i, histo.(i)))$ est inclus dans l'histogramme.

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 $i = 0 \dots n - 1$.

Pour calculer $l.(i)$ on pose $j = i$ et on fait décroître j tant que les colonnes sont plus grandes de la manière suivante :

Algorithme 1 : Calcul de $l.(i)$

```
// Précondition : les l.(k) avec k < i sont connus
j ← i ;
while j > 0 do
    if histo[j − 1] < histo[i] then
        L[i] ← j ;
        terminer ;
    else
        j ← L[j − 1] ;
L[i] ← 0 ;
```

Notre algorithme applique la méthode ci-dessus à chaque i.

Question 8.

Si $i = 0$, établir que la boucle 1 calcule bien $l.(i)$.

Solution. **Pour cette question et la suivante, il s'agit de comparer l'affectation réalisée par l'algorithme avec la valeur attendue pour $l.(i)$.**

Si $i = 0$, le rectangle $R((0,0),(i,histo.(i)))$ est la première colonne de l'histogramme :

— Il est bien contenu dans l'histogramme.

— On ne peut pas lui ajouter de colonne à gauche puisqu'il n'y en a pas.

Donc $l.(i) = 0$. Ca tombe bien, c'est l'affectation réalisée par l'algorithme. □

Question 9.

Pour $n \geq i > 0$, établir que l'algorithme 1 calcule bien la valeur de $l.(i)$ si les $l.(k)$ sont connus pour $0 \leq k < i$.

Solution. Il s'agit encore une fois de comparer *ce qu'on veut* (le rectangle $R((l.(i),0);(i,histo.(i)))$ est contenu dans l'histogramme et on ne peut pas lui ajouter une colonne à gauche) avec *ce que donne l'algorithme* (la valeur que l'algorithme attribue à $l.(i)$) en énonçant clairement les deux.

Une preuve rigoureuse impose de raisonner avec un invariant pour la boucle `while` [1]

Soit $i > 0$ (le cas $i = 0$ a été traité en question 8).

On pose l'invariant suivant : à la fin du tour courant de la boucle `while`, la valeur j prise par la variable `j` vérifie que $R((j,0),(i,histo.(i)))$ est contenu dans l'histogramme.

Avant la boucle $j = i$, la barre verticale de l'histogramme au dessus de la case $(i,0)$ est bien contenue dans l'histogramme. Cas de base OK.

Supposons qu'un certain nombre de tour de boucles aient été effectués et que l'invariant soit vérifié pour la valeur courante de j (on a bien sûr $j \leq i$).

S'il y a un tour supplémentaire, alors :

— Si $histo.(j-1) < histo.(i)$, alors on ne peut pas ajouter une colonne de hauteur $histo.(i)$ à gauche du rectangle $R((j,0),(i,histo.(i)))$: le nouveau rectangle ne serait plus contenu dans l'histogramme.

Ainsi, $R((j,0),(i,histo.(i)))$ est contenu dans l'histogramme et on ne peut pas le prolonger à gauche. Il vient que $l.(i) = j$. Trop bien, c'est l'affectation réalisée par l'algorithme !

— Sinon $histo.(j-1) \geq histo.(i)$. Alors le rectangle $R((j-1,0),(i,histo.(i)))$ est contenu dans la concaténation des deux rectangles suivants :

1. L'invariant pour la boucle `for` (tout les $l.(k)$ sont bien calculés avant la colonne i), lui, est donné par le sujet

— $R((l.(j-1),0),(j-1,histo.(j-1))$ (lequel est contenu dans l'histogramme par hypothèse sur les $l.(k)$ avec $k<i$)

— et $R((j,0),(i,histo.(i))$ (lequel est contenu dans l'histogramme par HR sur le j courant).

Ainsi le rectangle $R((j,0),(i,histo.(i))$ est contenu dans l'histogramme. Donc l'invariant est respecté pour la nouvelle valeur prise par j (qui est $l.(j-1)$).

Si on sort de la boucle, c'est

— soit que $histo.(j-1)<histo.(i)$, et on a vu que la valeur calculée est alors la bonne ;

— soit que $j=0$. Le rectangle $R((j,0),(i,histo.(i))$ est inclus dans l'histogramme par HR. Et on ne peut pas le prolonger à gauche (pas de colonne à gauche de 0). Donc $l.(i)=0$. Ca tombe bien, c'est l'affectation réalisée par l'algorithme.

La correction est établie. □

Question 10.

Montrer que l'algorithme fonctionne en temps $O(n)$ sur le cas particulier du tableau `let histo = [|1, 2, 3, . . . , n - 1, n, n, n - 1,..., 1|]` de taille $2n-1$.

Solution. Le tableau est de taille $2n-1$ et non $2n$ comme je l'avais écrit par erreur. On pose $N=n-1$ (la position du centre du tableau). La taille du tableau s'exprime aussi comme $2N+1$.

On montre qu'il y a au plus un tour de boucle `while` pour tout $0\leq i\leq n-1$ et au plus 3 après.

On a par analyse immédiate de l'histogramme proposé que $hist.(N+k)=hist.(N-k)=N+1-k$ pour $n\geq k\geq 0$. On en déduit que :

— $l.(0)=0$ puisque c'est vrai pour n'importe quel histogramme ;

— $l.(k)=k$ si $0<k\leq N$ car $hist.(k-1)<hist.(k)$;

— $l.(N+k)=N-k$ si $0\leq k\leq N$ car

  — entre les colonnes $N-k$ et $N+k$ (qui sont de même hauteur) toutes les colonnes sont plus hautes ;

  — et soit il n'y a pas de colonne à gauche de $N-k$ (c.a.d $k=N$) soit cette colonne est strictement plus basse que la colonne $N+k$

Pour les valeurs 0 à N, on ne rentre qu'une fois dans la boucle car la colonne de gauche est strictement plus basse que la colonne de droite. Le temps de calcul pour la 1ere moitié du tableau est un $O(N)$ donc un $O(n)$.

Considérons la colonne $N+k+1$ avec $N-1\geq k\geq 0$. Elle est plus basse que la colonne $N+k$. Un premier tour de boucle impose $j\leftarrow l.(N+k)=N-k$ donc j vaut $N-k\geq 1$ à l'issue du 1er tour.

Au tour suivant $j-1=n-k-1$. Alors $hist.(j-1)=hist.(n-k-1)=n+1-k-1=hist.(n+k+1)$. Et donc on fait un tour de plus avec $j\leftarrow l.(j-1)=l.(n-k-1)=n-k-1$.

— si $j=0$, c'est à dire si $k+1=N$, alors n'entre pas dans la boucle. Et on observe qu'on a trouvé $l.(N+k+1)$ en 2 tours de boucles ;

— si $j>0$ alors $j-1=N-k-2\geq 0$. Il vient $hist.(N-k-2)=n+1-k-2<N+1-k-1=hist.(N+k+1)$. On sort donc de la boucle. On a obtenu $l.(N+k+1)$ en 3 tours de boucles.

Bref, pour toutes les valeurs de N à $2N$, on fait au plus 3 passages dans la boucle `while`.
Complexité en $O(N)$ donc $O(n)$.

□

On se donne le type des histogrammes :

```
type hist = int array;;
```

Question 11.

Écrire la fonction `calculeL (histo:hist) : int array` qui calcule le tableau l tel que $l.(i)$ est l'indice défini précédemment. La complexité doit être linéaire (on ne demande pas de le justifier).

```
# calculeL histo;;
- : int array = [|0; 0; 2; 3; 2; 2; 0; 7|]
```

Solution. Code

```
let calculeL histo =
  let n = Array.length histo in
  (* L est un tableau de longueur n, initialisé à 0 *)
  let l = Array.make n 0 in
  for i = 1 to n - 1 do
    let j = ref i in
    try
      while !j > 0 do
        if histo.(!j - 1) < histo.(i)
        then (
            l.(i) <- !j;
            raise Exit
        )
        else
          j := l.(!j - 1)    (* on fait un saut *)
      done;
      l.(i) <- 0
    with Exit -> ()
  done;
  l
;;
```

□

On suppose écrite également la fonction `calculeR (histo:hist) : int array` qui calcule le tableau r tel que la valeur $r.(i) \geq i$ est l'indice maximal tel que $histo.(k) \geq histo.(i)$ pour tout k tel que $i \leq k \leq r.(i)$. Sa complexité est supposée linéaire en la longueur de l'histogramme.

```
# calculeR histo;;
- : int array = [|0; 5; 5; 3; 5; 5; 7; 7|]
```

Solution. Code

```
let calculeR (histo : hist) : int array =
  let n = Array.length histo in
  (* L est un tableau de longueur n, initialisé à 0 *)
  let r = Array.make n (n-1) in
  for i = n-2 downto 0 do
    let j = ref i in
    try
      while !j < n-1 do
        if histo.(!j + 1) < histo.(i)
        then (
          r.(i) <- !j;
          raise Exit
        )
        else
          j := r.(!j + 1)    (* on fait un saut *)
      done;
      r.(i) <- n-1
    with Exit -> ()
  done;
  r
;;
```

□

Question 12.

Justifier que pour tout i, le rectangle $R((l.(i), 0), (r.(i), histo.(i)))$ est inclus dans l'histogramme.

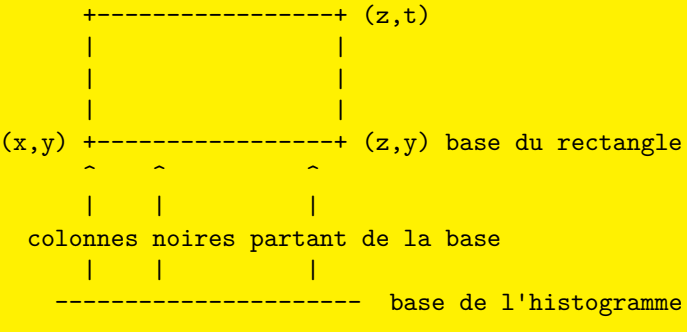
Solution. Les rectangles $R((l.(i), 0), (i, histo.(i)))$ et $R((i, 0), (r.(i), histo.(i)))$ sont inclus dans l'histogramme. Leur concaténation est $R((l.(i), 0), (r.(i), histo.(i)))$ est est inclus dans l'histogramme. □

Question 13.

Si $0 \leq x \leq z \leq n$ et $0 \leq y \leq t \leq n$, et si $R((x, y), (z, t))$ est inclus dans l'histogramme, montrer que $R((x, 0), (z, t))$ aussi.

Solution. Toute case noire d'un histogramme est sur une colonne noire partant de l'abscisse 0.

On peut donc ajouter y lignes horizontales en dessous de la ligne $[(x, y), (z, y)]$: elles seront toutes dans l'histogramme.



□

Il résulte de ce qui précède qu'un rectangle d'aire maximale inclus dans un histogramme « repose » sur la base de l'histogramme, donc est de la forme $R((x, 0), (z, t))$.

Question 14.

Soit un rectangle d'aire maximale inscrit dans l'histogramme.

Supposons que ce rectangle commence à l'indice g (gauche), termine à l'indice d (droit), et a pour hauteur h. Montrer qu'il existe $i \in [\![0, d]\!]$ tel que $histo.(i) = h$, $l.(i) = g$, et $r.(i) = d$.

Solution. Considérons un rectangle R d'aire maximale comme dans l'énoncé. C'est une succession de colonnes de hauteurs h. Si toutes les colonnes par lesquelles passe ce rectangle sont de hauteur $h' > h$, alors on peut ajouter une ligne et ce rectangle n'est pas d'aire maximale.

Soit $g \leq i \leq d$ l'indice d'une colonne de hauteur h. Alors toutes les colonnes sur sa gauche jusqu'à la position g sont de hauteur $\geq h$. Si $g = 0$, alors $g = l.(i)$. Sinon, la colonne $g - 1$ est de hauteur $h' < h$. Dans le cas contraire, on pourrait ajouter une colonne à notre rectangle : le nouveau rectangle obtenu serait contenu dans l'histogramme et on obtiendrait ainsi un rectangle d'aire supérieure strictement à celle de R. C'est absurde.

Il vient que $l.(i) = g$.

De même $r.(i) = d$. Et comme la hauteur de notre rectangle est $histo.(i) = h$, on a répondu à la question.

□

Question 15.

En déduire une fonction `plusGrandRectangleHistogramme (histo:hist) : int` qui calcule et renvoie l'aire d'un plus grand rectangle inscrit dans l'histogramme.

Donner la complexité de votre fonction plusGrandRectangleHistogramme.

```
# plusGrandRectangleHistogramme histo;;
- : int = 8
```

Solution. Code

```
let plusGrandRectangleHistogramme
      (histo:hist) : int =
  let n= Array.length histo in
  let l = calculeL histo and r = calculeR histo
  in let m = ref (histo.(0) * (r.(0)+1-l.(0))) in
     for i=1 to n-1 do
       let a = histo.(i) * (r.(i)+1-l.(i)) in
       if a > !m then m:=a;
     done;
     !m;;
```

Le calcul de la longueur d'un tableau est en $O(1)$, celle de l, r est un $O(n)$. Il y a $n-1$ passages dans la boucle pour un coût à chaque passage $O(1)$.

Au final la complexité est linéaire. □

2.4 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 16.

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

```
# rectangleToutZero tab2;;
- : int = 12
```

Solution. Code

```
  let rectangleToutZero(tab2: int array array) : int =
  let col = colonneZeros(tab2) in
  let n = Array.length col in
  let best = ref 0 in
  for i = 0 to n-1 do
    let aire = plusGrandRectangleHistogramme col.(i) in
    if !best < aire then best := aire;
  done;
  !best;;
```

□

Question 17.

Quelle est la complexité de votre fonction ?

Solution. La construction du tableau `col` par `colonneZeros` se fait en $O(n^2)$.

Ensuite, on boucle sur les lignes de `col` (n lignes) et, pour chacune, on calcule l'aire du plus grand rectangle de l'histogramme représenté par la ligne courante ($O(n)$). On renvoie l'aire maximum parmi toutes les lignes étudiées.

Cette étape est un $nO(n)$ soit $O(n^2)$.

Au final $O(n^2)$.

□