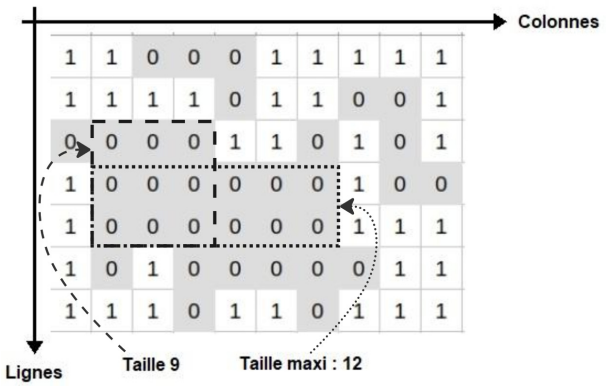
Exercice 1

Cet exercice utilise les listes Python mais seules les opérations qui suivent sont autorisées. Si `L` est une liste :

- `len(L)` renvoie la longueur de la liste L, c'est-à-dire son nombre d'éléments, complexité en O(1) ;
- `L[i]` désigne l'élément d'indice `i` de `L` (où `i` est compris entre 0 et `len(L)-1`), complexité en O(1) ;
- `L.append(e)` modifie la liste `L` en lui ajoutant l'élément `e` en dernière position, complexité en O(1) ;
- `L.pop()` renvoie le dernier élément de la liste `L` (supposée non vide) et supprime l'occurrence de cet élément en dernière position de la liste, complexité en O(1) ;
- les itérations sur une liste peuvent être effectuées directement (une liste Python est un objet itérable) ou à l'aide de la fonction range ;
- les listes peuvent être créées avec des boucles, ou en compréhension (par exemple `[0 for i in range(5)]`) ou avec une syntaxe de la forme `[0]*5`.

D'autre part, les fonctions `max` et `min` ne peuvent être utilisées que pour considérer le maximum ou le minimum de deux éléments (et non pour une liste).

Le but de ce problème est de détecter le plus grand rectangle dans une image. Plus précisément, on se donne un tableau à deux dimensions dont les valeurs sont 0 ou 1 et on cherche un rectangle constitué uniquement de 0 d'aire maximale (l'aire étant le nombre total de cases de ce rectangle) :



L'image sera représentée par une liste de taille p (nombre de lignes) dans laquelle chaque élément est une liste de taille q (nombre de colonnes) constituée de 0 et de 1. L'exemple de départ est représenté par la liste :

```
G = [[1, 1, 0, 0, 0, 1, 1, 1, 1, 1],
     [1, 1, 1, 1, 0, 1, 1, 0, 0, 1],
     [0, 0, 0, 0, 1, 1, 0, 1, 0, 1],
     [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
     [1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
     [1, 0, 1, 0, 0, 0, 0, 0, 1, 1],
     [1, 1, 1, 0, 1, 1, 0, 1, 1, 1]]
```

On appellera «grille» une liste `G` du type précédent, on appellera «ligne» d'une telle grille l'un des éléments de cette liste `G` (les lignes sont numérotées à partir de 0). Par exemple, la ligne 2 de `G` est :

```
>>> G[2]
[0, 0, 0, 0, 1, 1, 0, 1, 0, 1]
```

On définit de même les colonnes de la grille. On désignera par G_{ij} l'entier `G[i][j]` d'une grille `G`.

► **Q1.** Écrire une fonction `dimensions(G)` qui prend en argument une liste de listes d'entiers, qui vérifie que chaque ligne comporte le même nombre d'entiers et qui renvoie un couple `(p,q)` correspondant aux dimensions (lignes puis colonnes) de `G` et qui renvoie `(0,0)` lorsque les lignes ne sont pas toutes de même taille. Par exemple :

```
>>> dimensions([[0,0,0], [0,0,0], [0,0,0], [0,0,0]])
(4, 3)
>>> dimensions([[0,0,0], [0,0,0], [0,0,0], [0,0]])
(0, 0)
```

► **Q2.** Écrire une fonction `EstRectangle(G, l1, l2, c1, c2)` qui prend en argument une grille, ainsi que 4 entiers et qui renvoie `True` lorsque tous les entiers G_{ij} pour `l1` $\leqslant i <$ `l2` et `c1` $\leqslant j <$ `c2` sont nuls et `False` sinon (on suppose que les arguments sont cohérents avec la grille et on ne demande pas de le vérifier dans cette fonction).

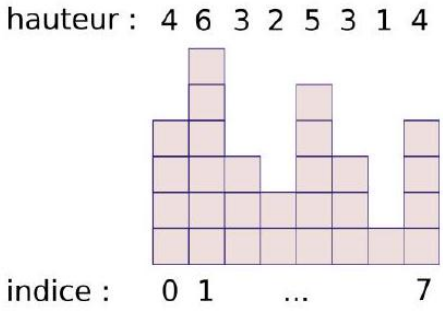
► **Q3.** Proposer un jeu de tests (deux ou trois tests utilisant `assert` par exemple) pour illustrer la fonction précédente en justifiant brièvement ce choix.

► **Q4.** Quelle est la complexité (temporelle) de la fonction `EstRectangle` ?

► **Q5.** *Version brute*
Écrire une fonction `PlusGrandRectangle(G)` qui renvoie la taille du plus grand rectangle (de 0) dans la grille `G` en testant tous les rectangles possibles à l'aide de boucles imbriquées.

► **Q6.** Si on suppose que `G` est une grille carrée de taille $n \times n$, donner une ordre de grandeur de la complexité temporelle de cette fonction `PlusGrandRectangle` (en $O(n^{\alpha})$).

On va maintenant s'intéresser à un cas particulier du problème : on se donne un histogramme de longueur n, c'est-à-dire une liste d'entiers naturels `H` dans laquelle chaque entier représente la hauteur d'une colonne de cet histogramme :



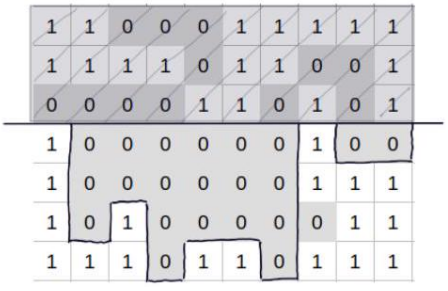
Cet histogramme sera représenté par une liste `H=[4,6,3,2,5,3,1,4]`. Dans ce qui suit, la longueur de la liste `H` sera notée n.

► **Q7.** On se donne une liste d'entiers `H` de taille n représentant un histogramme et on note $\text{hmax}_{i,j}$ la hauteur minimale (la valeur minimale de `H`) entre les positions i et j (comprises). Expliquer brièvement pourquoi, pour chaque valeur de i, on a les relations suivantes :

$$\text{hmax}_{i,i} = \texttt{H}[i] \quad \text{et} \quad \text{hmax}_{i,j} = \begin{cases} 0 & \text{si } j < i \\ \min\left(\text{hmax}_{i,j-1}, \texttt{H}[j]\right) & \text{si } j > i \end{cases}.$$

► **Q8.** Écrire une fonction `CalculHauteurs(H)` qui prend en argument une liste `H` (de taille n) et qui renvoie une liste de listes `L` (avec n listes de longueur n) telle que `L[i][j]` contienne $\text{hmax}_{i,j}$.

Nous reprenons désormais le problème d'origine. On se donne une grille G et on cherche le plus grand rectangle de 0 dans cette grille. On remarque que si ce plus grand rectangle commence en ligne i, alors ce plus grand rectangle est le plus grand rectangle d'un histogramme obtenu en retirant les lignes 0 à $i-1$ de la grille G et dont les hauteurs sont les zéros sous cette base (on regarde vers le bas par rapport à ce qu'on a fait avant) :



On est donc ramené à chercher le plus grand rectangle dans l'histogramme `H=[0,3,2,4,3,3,4,0,1,1]`.

La première étape consiste à transformer la grille afin de pouvoir obtenir rapidement les histogrammes «sous une ligne».

On veut créer une nouvelle grille, de la même taille que celle de départ mais telle que, en case (i, j), on ait le nombre de 0 consécutifs au dessous de la case (i, j) (cette case étant comprise). Pour une case en ligne i, colonne j, on a donc deux possibilités :

– la case contient 1 : la valeur sera 0 ,

– la case contient 0 : le nombre de 0 consécutifs en dessous vaut donc 1 de plus que celui en ligne $i+1$, même colonne.

Sur l'exemple de départ, on doit obtenir :

1	1	0	0	0	1	1	1	1	1
1	1	1	1	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1
1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
1	0	1	0	0	0	0	0	1	1
1	1	1	0	1	1	0	1	1	1

0	0	1	1	2	0	0	0	0	0
0	0	0	0	1	0	0	1	3	0
1	4	3	5	0	0	5	0	2	0
0	3	2	4	3	3	4	0	1	1
0	2	1	3	2	2	3	0	0	0
0	1	0	2	1	1	2	1	0	0
0	0	0	1	0	0	1	0	0	0

► **Q9.** Écrire une fonction `ColonneZero(G)` qui effectue ce travail et renvoie la grille avec les tailles des colonnes de zéro. Si G est de taille $p \times q$, la complexité de la fonction devra être en $O(p.q)$ (et donc en $O\left(n^2\right)$ si G est de taille $n \times n$).

► **Q10.** On suppose que l'on dispose d'une fonction `PgrHistogramme(H)` qui renvoie la taille du plus grand rectangle d'un histogramme avec une complexité linéaire par rapport au nombre de colonnes de l'histogramme.
En utilisant des fonctions vues précédemment, écrire une fonction `PgrUltime(G)` qui renvoie la taille du plus grand rectangle avec une complexité en $O\left(n^2\right)$ si G est de taille $n \times n$).

► **Q11.** Afin de comprendre l'intérêt d'un tel algorithme, on s'intéresse à des images que l'on représente numériquement sous la forme d'une liste de listes dont chaque case est un triplet d'entiers `(a, b, c)` dans $[\![0, 255]\!]$ correspondant au codage RGB du pixel (avec autant de colonnes par ligne).
Écrire une fonction `convertir(image)` qui prenne en argument une image sous ce format et renvoie une grille de 0 et de 1 : 0 si la moyenne des trois entiers est inférieure ou égale à 120 et 1 sinon.

Exercice 2 – base de données

Nous considérons une base de données gérant, des notes de colle, constituée de 3 tables.

La table `Creneaux` est constituée de 4 champs :

- `numero` : de type entier – clé primaire ;
- `colleur` : de type chaîne de caractères, nom du colleur (un même nom peut apparaître plusieurs fois mais désigne la même personne) ;
- `jour` : de type chaîne de caractères (`"lundi"`, `"mardi"`, etc.) ;
- `heure` : de type chaîne de caractères (`"12h"`, `"13h"`, etc.).

La table `Etudiants` est constituée de 4 champs :

- `id` : de type entier – clé primaire (identifiant numérique de l'étudiant) ;
- `nom` : de type chaîne de caractères ;
- `prenom` : de type chaîne de caractères ;
- `groupe` : de type chaîne de caractères (`"A"`, `"B"`, etc.).

La table `Colles` est constituée de 4 champs :

- `id` : de type entier – clé primaire (identifiant numérique de la colle) ;
- `idetud` : de type entier (identifiant numérique de l'étudiant) ;
- `numero` : de type entier (identifiant numérique du créneau de colle) ;
- `note` : de type flottant ;
- `semaine` : de type int (numéro de la semaine de colle).

► **Q12.** Écrire en SQL une requête donnant la liste des étudiants du groupe `"A"`.

► **Q13.** Écrire en SQL une requête donnant la liste des colleurs différents intervenant le jeudi.

► **Q14.** Écrire en SQL une requête donnant la liste des notes avec la date correspondante pour l'étudiant dont l'identifiant est le numéro 11, en classant les résultats par ordre décroissant des notes.

► **Q15.** Écrire en SQL une requête donnant la moyenne de l'étudiant nommé "MELLEREAU".

► **Q16.** Écrire en SQL une requête donnant le nom, le prénom de chaque étudiant, suivis de sa meilleure note.

► **Q17.** Écrire en SQL une requête donnant les noms des deux colleurs «ayant» les meilleurs moyennes.

► **Q18.** Écrire en SQL une requête donnant le nom, le prénom des étudiants ayant une moyenne supérieure ou égale à 14.