Recherche du plus grand rectangle blanc

1 Modalités

Le TP est à commencer à partir du **25 septembre** et à rendre **au plus tard le 25 octobre à 20h**. Il est à réaliser *en binôme exclusivement*, durant le temps libre. L'application TEIDE servira pour la formation des binômes et le rendu du TP.

Travail à rendre.

Il vous est demandé un document synthétique tenant sur **au maximum deux feuilles recto-verso** où vous ferez figurer les réponses aux questions ainsi que tout élément que vous jugeriez bon de préciser. Vous déposerez sur TEIDE dans une archive `.tar.gz` :

— le document synthétique en format électronique `.pdf`
— les sources dûment ***commentées*** de votre réalisation (*un code qui ne compile pas ne sera pas accepté !*),
— les jeux de tests (les programmes de tests et les fichiers contenant tous les dallages testés).

L'ensemble des programmes ayant servi à l'élaboration du TP doivent être dument commentés. Essayez d'attacher une grande importance à la lisibilité du code : il est indispensable de respecter les conventions d'écriture (*coding style*) du langage et de bien présenter le code (indentation, saut de ligne, *etc*) ; les identificateurs (nom de types, variables, fonctions, *etc*) doivent avoir des noms cohérents documentant leur rôle ; enfin, un commentaire se doit d'être sobre, concis et aussi précis que possible.

Barème indicatif : code (programmation et algorithmique) : 6 points ; code (forme) : 2 points ; tests : 6 points ; rapport : 6 points.

2 Sujet et questions

Problème. Il s'agit de rechercher dans un dallage rectangulaire, pavé de dalles blanches et de dalles noires, le plus grand (en nombre de dalles) rectangle blanc, c'est-à-dire ne contenant aucune dalle noire. Le sujet étudie plusieurs algorithmes de complexités différentes : d'une solution naïve, par recherche exhaustive, on arrive en s'aidant d'une structure de données intermédiaire à une complexité optimale.

On s'attachera pour chaque solution à étudier sa complexité en temps en comptant le nombre d'accès aux cases du tableau qui représente le dallage et sa complexité en mémoire par la taille des données auxiliaires, s'il en est.

Le dallage est représenté par un *tableau contenant des booléens*. Par convention, on prendra la valeur "vrai" pour une dalle blanche et la valeur "faux" pour une dalle noire.

Chaque solution devra correspondre à un programme qui en fonction du dallage passé en entrée, calculera puis affichera les coordonnées du plus grand rectangle blanc et sa taille (hauteur et largeur).

2.1 Tests fonctionnels

Les tests doivent impérativement essayer tous les petits cas (petits dallages) et les cas limites comme par exemple des pavages tout noirs ou tout blancs, *etc*. Selon les algorithmes, il existe d'autres cas limites

intéressants à tester. Chacun de ces cas de tests sera enregistré dans un *fichier*. Par exemple, le fichier suivant représente un dallage de 4 lignes et 5 colonnes contenant 5 dalles noires.

```
4 5
00000
01000
00101
00110
```

Question 1 Lecture de dallage dans un fichier : implémentez une fonction qui lit un fichier de la forme ci-dessus et qui crée le dallage correspondant. La fonction doit traiter comme un cas d'erreur le fait que le fichier n'est pas conforme à la description ci-dessus. Précisez les cas d'erreur que vous identifiez, traitez les et validez le traitement par des tests que vous préciserez.

2.2 Solution 1 : recherche naïve

La solution naïve de ce problème consiste à considérer tous les rectangles possibles en vérifiant pour chacun s'il est blanc.

Question 2 Solution 1

(a) Expliquez l'algorithme de la solution 1, notamment comment vous choisissez de représenter un rectangle et comment vous énumérez tous les rectangles à l'aide de cette représentation.

(b) Implémentez puis testez votre solution en précisant quels tests ont été réalisés.

(c) Donnez, en justifiant, les coûts en temps et en mémoire de votre algorithme en supposant un dallage carré de côté n.

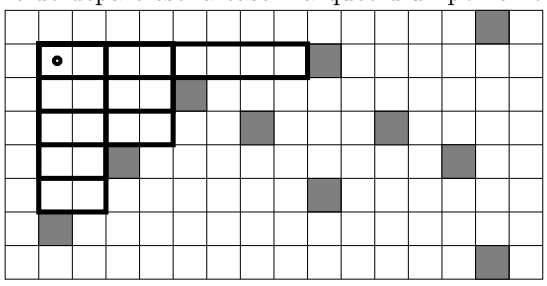
Cette première solution, coûteuse mais simple, pourra vous servir de référence lors des tests des solutions suivantes : il est préférable d'être convaincu de sa correction avant d'avancer dans le sujet !

2.3 Solution 2 : amélioration de la solution naïve

On peut réduire le coût de la solution 1, de la manière suivante. L'idée est qu'on part d'un rectangle composé d'une ligne (de hauteur 1), le plus large possible, et qu'on augmente sa hauteur petit à petit, tout en réduisant sa largeur lorsque c'est nécessaire. On évite ainsi d'avoir à revérifier pour chaque rectangle, l'ensemble de ses cases.

On énumère tous les coins supérieurs gauches possibles du rectangle. Pour un coin supérieur gauche donné, on parcourt toutes les possibilités pour la ligne inférieure du rectangle, en réduisant sa largeur au fur et à mesure. Pour la première ligne, on avance tant qu'on ne rencontre que des cases blanches sur la ligne. Pour la deuxième ligne, on recommence en partant de la gauche et on s'arrête dès qu'on a rencontré une case noire ou que l'on a atteint la colonne à laquelle on s'était arrêté à la ligne précédente. *Etc.*

Sur la figure suivante, on peut voir les limites de tous les rectangles testés par cette solution, lorsque le point de départ est la case marquée d'un point noir.



Question 3 Solution 2

(a) Eventuellement, décrivez les points de l'algorithme que vous avez eu à préciser et qui n'apparaissent pas dans l'énoncé.

(b) Implémentez puis testez la solution 2 en précisant quels tests ont été réalisés.

(c) Donnez, en justifiant, les coûts en temps et en mémoire de votre algorithme en supposant un dallage carré de côté n.

2.4 Solution 3 : utilisation d'un tableau intermédiaire

Il est possible de faire encore beaucoup mieux. Pour cela, on parcourt le dallage ligne par ligne et on maintient un tableau supplémentaire dans lequel on stocke pour chaque colonne, de combien de cases on peut remonter depuis la ligne courante sans rencontrer de case noire : on parle de la hauteur de la colonne et on appelle ce tableau `hauteurs`. On obtient facilement les valeurs du tableau `hauteurs` pour une ligne à partir de celles de la ligne précédente.

Par exemple, sur le dallage précédent, on commence avec un tableau rempli de 0, il devient pour la première ligne :

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

puis :

2	2	2	2	2	2	2	2	2	0	2	2	2	2	1	2	pour la deuxième ligne
3	3	3	3	3	0	3	3	3	1	3	3	3	3	2	3	pour la troisième ligne
4	4	4	4	4	1	4	0	4	2	4	0	4	4	3	4	pour la quatrième ligne

etc.

Une fois ce tableau rempli pour une ligne, il devient possible de calculer l'aire du plus grand rectangle dont le bord du bas se trouve sur cette ligne. On examine tous les rectangles de hauteur maximale de cette ligne en énumérant toutes les colonnes de début et toutes les colonnes de fin possibles. Ce faisant, on maintient la hauteur du plus grand rectangle valide entre cette colonne de début et cette colonne de fin (la plus petite hauteur des colonnes contenues entre ce début et cette fin).

Question 4 Solution 3

(a) Eventuellement, décrivez les points de l'algorithme que vous avez eu à préciser et qui n'apparaissent pas dans l'énoncé.

(b) Implémentez puis testez la solution 3 en précisant quels tests ont été réalisés.

(c) Donnez, en justifiant, les coûts en temps et en mémoire de votre algorithme en supposant un dallage carré de côté n.

2.5 Pile

La dernière solution utilise une pile comme structure intermédiaire de calcul.

Bien que dans la pratique, on utiliserait les fonctionnalités d'une bibliothèque adéquate, il vous est demandé, dans ce TP et en guise d'exercice, d'implémenter une structure de données "Pile" à l'aide d'une *liste simplement chaînée*.

Question 5 Pile

— Prévoyez *a minima* les primitives classiques de manipulation d'une pile (création d'une pile vide, test à vide, empiler — `pop`, dépiler — `push`). Préciser les préconditions d'utilisation, si nécessaire.

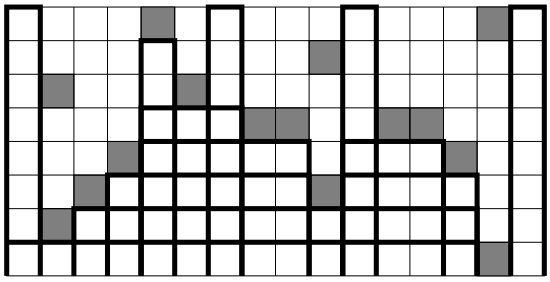
— *Encapsulez* proprement votre structure de données : l'utilisateur de la pile ne doit pas pouvoir modifier celle-ci, sinon en utilisant les primitives prévues à cet effet.

— Implémentez et testez en précisant quels tests sont réalisés.

2.6 Solution 4 : utilisation d'une pile intermédiaire

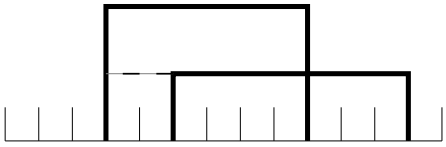
On peut encore faire mieux que la solution 3, en traitant chaque ligne en une seule boucle. On peut en effet remarquer que les différents rectangles qui peuvent être intéressants pour notre recherche sont imbriqués les uns dans les autres. On considère comme intéressants les rectangles qui sont maximaux, c'est-à-dire qu'on ne peut pas les agrandir en augmentant l'une des dimensions, que ce soit en largeur ou en hauteur, sans tomber sur une case noire.

Sur la figure suivante, sont représentées les limites des rectangles intéressants (maximaux) se terminant sur la dernière ligne.

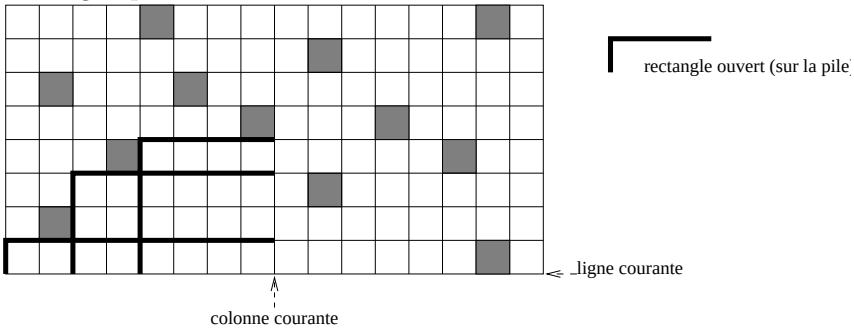


Ces rectangles sont imbriqués, dans le sens où un rectangle A dont le bord gauche est situé avant le bord gauche d'un rectangle B, a son bord droit qui est situé soit avant même le bord gauche du rectangle B, soit après le bord droit du rectangle B. On ne peux pas avoir le bord gauche d'un rectangle A, puis le bord gauche d'un rectangle B, puis le bord droit du rectangle A, puis le bord droit du rectangle B.

La situation suivante est par exemple impossible, car le rectangle le plus bas peut être agrandi vers la gauche (comme le montrent les pointillés), et n'est donc pas maximal.



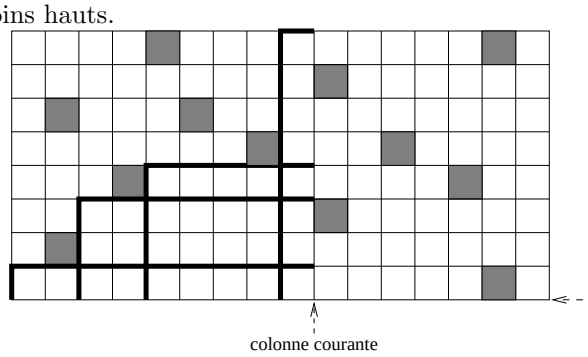
Comme dans la solution précédente, on parcourt le dallage ligne par ligne et on maintient le tableau `hauteurs` qui stocke pour chaque colonne, de combien de cases on peut remonter depuis la ligne courante sans rencontrer de case noire. En même temps, pour chaque ligne, on avance colonne par colonne et on ouvre des rectangles potentiellement maximaux.



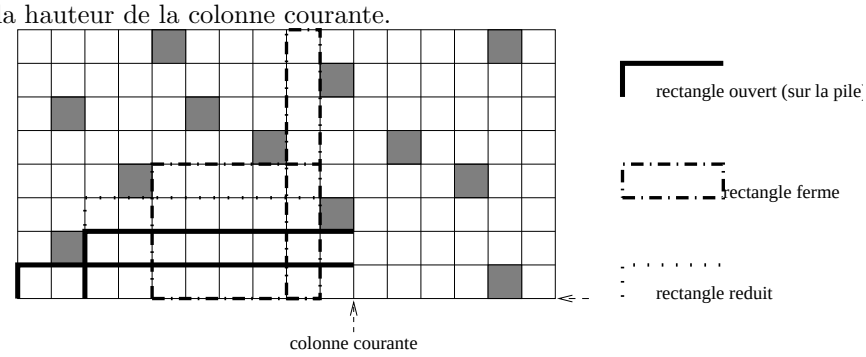
Pour stocker, les rectangles ouverts, on utilise une *pile* : quand on ouvre un rectangle, on empile sa colonne de début et sa hauteur. Les rectangles précédemment empilés n'ont pas à être touchés, tant qu'on n'aura pas fermé le rectangle que l'on vient d'empiler car tant qu'il est possible de lui ajouter des colonnes, il est forcément possible d'en ajouter également aux rectangles empilés précédemment, puisqu'ils sont moins hauts.

Quand on avance d'une colonne, *si la hauteur de la nouvelle colonne est plus grande que la précédente,* on ouvre un nouveau rectangle : les rectangles ouverts précédemment sont toujours valides, puisqu'ils sont

moins hauts.



Si la hauteur de la nouvelle colonne est plus petite que la précédente, on est obligé de fermer le rectangle se situant en haut de la pile : il n'est plus possible d'ajouter des colonnes en restant à sa hauteur et de continuer à l'agrandir. On ferme et on dépile alors tous les rectangles qui sont plus hauts que la colonne courante, sauf le dernier (s'il existe). Le dernier subit un traitement spécial : on le laisse sur la pile en réduisant sa hauteur à la hauteur de la colonne courante.



A chaque fois que l'on réduit ou que l'on ferme un rectangle du sommet de la pile, il faut calculer son aire et vérifier si c'est la plus grande qu'on n'ait rencontrée.

Une fois arrivé à la dernière colonne, on ferme bien sûr tous les rectangles ouverts.

Avec cette technique on a calculé l'aire de tous les rectangles maximaux dont le bord du bas peut se trouver sur la ligne courante. Le rectangle le plus grand étant un rectangle maximal, faire cette opération sur toutes les lignes donne effectivement l'aire du plus grand rectangle blanc.

Question 6 Solution 4

(a) Eventuellement, décrivez les points de l'algorithme que vous avez eu à préciser et qui n'apparaissent pas dans l'énoncé.

(b) Identifiez le type des éléments que doit stocker la pile.

(c) Implémentez puis testez la solution 4 en précisant quels tests vous avez réalisés.

Question 7 Coût de la solution 4 On remarque que pour chaque colonne, on empile au plus un rectangle. Comme on ne dépile pas plus de rectangles qu'on n'en empile, le nombre d'opérations sur la pile est au maximum de 2 fois le nombre de colonnes. On suppose un dallage carré de côté n.

(a) Reprenez le raisonnement ci-dessus et déduisez-en, le nombre d'opérations, pour une ligne fixée.

(b) Quel est le coût de la solution 4 ?

(c) Pourquoi ce coût est-il optimal ? (on ne peut pas avoir de solution plus efficace !).

2.7 Tests de performances

Question 8 Générateur de tests aléatoires : pour compléter les tests, vous implémenterez un générateur aléatoire de dallages carrés :

- fixez en paramètres la taille du dallage et le pourcentage moyen de dalles blanches ;
- pour chaque dalle, tirer uniformément au hasard si la dalle est blanche ou noire.

Question 9 Performances : utilisez les tests aléatoires sur de grands dallages pour, à l'aide d'un graphique,

- comparer le coût théorique de chaque solution à son coût mesuré,
- comparer les différences de performances des solutions entre elles.