Léane Texier

TP1 - Diviser pour régner - Le plus grand rectangle

Question 1 : Une remière approche

Un rectangle de surface maximale respectant les contraintes a nécessairement deux sommets de la forme (xi, 0), (xj,0) avec 0<=i<j<=n-1. En effet, une des contraintes indique que le rectangle qu'on souhaite que la base de c rectangle soit sur l'axe des x. Cela implique donc que deux sommets soient de la forme (xi, 0), (xj,0) avec 0<=i<j<=n-1.

La surface du rectangle de surface maximale respectant les contraintes et dont deux sommets sont (xi, 0), (xj,0) est (xj-xi) * h avec h la hauteur du plus petit point situé entre les abscisses xi et xj. S'il n'y a aucun point, h est alors la hauteur maximale entrée en paramètre.

Algorithme en n^3 (rect_q1) :
* Je considère que l'aire maximale est au minimum de 0 donc je met maxi = 0.
* Je rajoute ensuite les points (0,0) et (l,0). La liste de points est alors de taille n+2.
* Pour chaque couple (i,j) on cherche le point le plus bas entre ceux deux points. S'il n'y en a pas, on considère que la hauteur est donc h. i est compris entre 0 et n et j est compris entre i et n+1 car on sait que 0<=i<j<=n+1. On calcule pour chaque couple (i,j) l'aire du rectangle grâce à la formule ci-dessus. Puis, on regarde si l'aire qu'on vient de trouver est plus grande que les autres que nous avons trouvé avant. Si oui, nous mettons sa valeur dans la variable maxi qui contient la valeur de l'aire maximale du rectangle trouvés jusqu'à présent.
=> maxi contient alors l'aire du plus grand rectangle

Algorithme en n^2 (rect_q1_v2):
Cet algorithme est assez similaire à celui en n^2.
La seule différence est que la hauteur minimale entre deux points est enregistrée au fur et à mesure que l'on augmente le j en comparant simplement la valeur de l'ancienne hauteur minimale avec la hauteur du point actuel j qui sera lors de la prochaine boucle entre i et le nouveau j.

Question 2 : Diviser pour régner

Algorithme diviser pour régner (diviser_pour_regner):
Cet algorithme enlève tout d'abord les points d'ordonnée 0 ou h car ils ne pourront jamais être dans le rectangle que l'on cherche. Nous décrémentons n en fonction afin que n représente toujours le nombre de points présents dans notre liste.
La deuxième partie de l'algorithme (rect_q2) cherche alors l'aire du plus grand rectangle respectant les contraintes.
* Si le nombre de points est égale à 0, cela signifie qu'il y a aucun point dans notre rectangle. On calcule alors son aire.
* Sinon, on cherche le point de hauteur minimale (forcément supérieure à h et inférieure à 0 grâce à la première partie de l'algorithme). Pour cela, on parcourt tous les points présents dans notre rectangle, en cherchant l'ordonnée la plus faible.
* Après avoir trouvé le point de hauteur minimale, nous calculons l'aire du rectangle ayant comme hauteur la hauteur minimale trouvée. Puis, nous calculons l'aire du plus grand rectangle présent à gauche de ce point et également celui à droite de ce point.
* Pour finir, nous prenons la plus grande aire entre ces différents rectangles.

Cet algorithme est beaucoup plus efficace que les précédents. En effet, sur les jeux de données proposés, il ne prend que quelques secondes tandis que les précédents prenaient beaucoup plus de temps. Cela est du au fait que la complexité de cet algorithme est plus faible et ……. (paralle ?)…….

Dans le meilleur des cas, l'algorithme est en
Dans le pire des cas, l'algorithme est en

Question 4 : Linéaire ?

Algorithme linéaire (rect):

* Tout d'abord nous supprimons les points d'ordonnée 0 ou h car ils ne pourront jamais être dans le rectangle que l'on cherche. Nous décrémentons n en fonction afin que n représente toujours le nombre de points présents dans notre liste.
* Parallèlement à cela, nous calculons la distance maximale entre 2 points consécutifs qui n'ont pas leur ordonnée égale à 0 ou h suivant leurs abscisses . Dans le cas du premier point, nous prenons la distance entre son x et 0. Dans le cas du dernier point, nous prenons la distance entre son x et l.
* Nous calculons alors l'aire du plus grand rectangle qui a deux autres de ses points de la forme (xi, h), (xj,h) avec 0<=i<j<=n-1. Sa hauteur est donc h et sa largeur est la plus grande distance entre 2 points consécutifs trouvés ci-dessus.
* Nous ajoutons ensuite les points (0,0) et (l,0). La liste de points est alors de taille n+2. Nous mettons donc à jour le n.
* Boucle while :
    * Si ma liste temporaire de points est vide ou que l'ordonnée de mon dernier point de cette liste est plus petit que le courant, alors on ajoute le point à la liste temporaire et on passe au point suivant. Cette liste temporaire comporte ainsi une sous-liste de mes points mais ayant tous une ordonnée supérieure ou égale à son précédent.
    * Sinon, cela signifie que l'ordonnée de mon point actuel est inférieur au dernier de ma liste. J'enlève alors le dernier point de ma liste temporaire et je calcule l'aire du rectangle ayant pour hauteur l'ordonnée du point que je viens de retirer de ma liste temporaire et pour abscisse la différence d'abscisse entre l'abscisse du point que je viens de retirer de ma liste temporaire et mon point actuel.