TP2 ACT

Matthieu Caron et Armand Bour

vendredi 25 septembre 2015

Question 1 La premiere polyligne ne représente pas une ligne de toit puisqu'elle possède une ligne en diagonale.
La seconde polyligne est bien une ligne de toit.
La troisieme n'est pas une ligne de toit.
La quatrieme n'est pas une ligne de toit.

Question 2 La condition pour qu'une polyligne soit une ligne de toit, deux couples consecutifs doivent avoir un x ou un y en commun. L'idée est d'alterner l'égalité entre les x et les y. Par exemple, si on a le couple (x,y1)(x,y2) alors le point suivant sera (x1,y2).

Question 3 Il correspond à la ligne de toit de la figure B.

Question 4 **Avant tout :** n est le nombre d'immeuble x la plus grande abscisse est donc la largeur du tableau et y la hauteur du tableau

On distingue 4 phases :

La première phase consiste à créer et remplir le tableau :
Dans cette première phase on dispose de notre liste de bâtiments, un bâtiment étant un triplet $(debut, hauteur, fin)$
Il nous faut d'abord trouver l'abscisse et la hauteur maximum pour créer notre tableau de booléen. On fait alors un parcours de la liste en $O(n)$. Ensuite, on crée notre tableau 2D de $False$ on a donc une complexité en $O(x * y)$. Enfin *rajouter nos bâtiments, ce qui nous coûte le plus* en terme de complexité. Quel est notre pire cas ? On a nos batiments qui sont tous de hauteur maximum et de largeur maximum la complexité sera donc de $O(n * x * y)$.
La deuxième phase consiste à chercher le début de la ligne de toit. Elle a une complexité de $O(x)$. On commence en $(0, 0)$ et on parcourt le tableau horizontalement en incrémentant les abscisse jusqu'à tomber sur une case avec la valeur $True$.
La troisième phase consiste à se déplacer dans le tableau vers le haut ou vers la droite. Par défaut, on se déplace vers le haut jusqu'à que la case suivante aie la valeur $False$, auquel cas on commence à se déplacer vers la droite, et, de façon similaire, jusqu'à que la case suivante aie la valeur $False$. Dans ce cas, soit la case supérieure a également la valeur $False$, auquel cas on passe à la quatrième phase, soit elle a la valeur $True$, auquel cas on reprend la troisième

phase.
La quatrième phase consiste à se déplacer dans le tableau vers le bas, jusqu'à que la case suivante aie une valeur de $False$, auquel cas on reprend la troisième phase.
L'algorithme se termine lorsque on a parcouru tout le tableau horizontalement. Par conséquent, la complexité des phases 3 et 4 est de $O(n)$ (avec n la longueur de la ligne de toit). En conclusion si on compte le remplissage de tableau la complexité se fait en $O(x * y + x + n)$.

Question 5 Dans ce nouvel algorithme, il n'y a plus de remplissage de tableau. On possède notre ligne de toit (initialement vide) et notre liste de batiments. On ajoute a chaque tout un batiments à la ligne de toit. On a donc imaginé une fonction qui pour un batiment retourne un ligne de toit, donc en temps constant.

```
def retourneListeToit(batiment):
  (a,b,c) = batiment
  return [(a,b),(c,0)]
```

La fonction du dessus se fait donc en $O(1)$ et du coup pour chaque batiment on fusionne la ligne de toit avec la ligne de toit du batiment, donc on fait n fois la fusion qui est en $O(n)$ donc la complexité est en $O(n^2)$

Question 6 et 7 : explications L'implémentation est faite en python, la fusion se fait en $O(n)$ avec n le nombre de points dans les lignes de toits passées en arguments de la fonction. On sait qu'on ne fait pas plus d'étapes car on parcours tout simplement chacune des deux liste une fois. Enfin pour ce qui est de la création de la ligne de toit on va a chaque fois diviser par deux le problème ce qui explique la complexité en $log(n)$ pour en suite fusionner les sous problème. On a donc une complexité totale de $O(n * log(n)$. Actuellement notre fonction (création de ligne de toit en fonction d'une liste de batiments passé en argument) n'est pas récursive terminale, ce qui signifie qu'à partir d'une certaine taille de la liste de batiments, la fonction va planter à cause d'un stack overflow. On a conscience de ça mais on a toujours pas trouvé de solution récursive terminale.