ACT

TP1: Diviser pour Régner : La ligne des toits

GIBIER François | MAZURE Antoine

Q1)

1.1)

- (2, 0)(2, 5)(4, 4)(4, 7)(5, 7)(5, 0)
  Cette polyligne n'est pas une ligne de toits car une seule coordonnée peut changer entre deux points. Ici on passe de (2, 5) à (4, 4) ce qui n'est pas une ligne horizontale ou verticale.

- (2, 0)(1, 4)(4, 4)(4, 7)(5, 7)(5, 0)

Fusion

Cette polyligne n'est pas une ligne de toits car pareil qu'à la une, on ne peut pas passer directement de (2, 0) à (1, 4), il faudrait un point intermédiaire (2, 4) ou (1, 0).

- (2, 0)(2, 5)(4, 5)(4, 7)(5, 7)(5, 0)
  Cette polyligne est une ligne de toits.

- (2, 0)(2, 5)(4, 5)(4, 7)(5, 7)(6, 7)(5, 0)
  Cette polyligne n'est pas une ligne de toits car on passe de (6, 7) à (5, 0) directement (cf: 1 et 2).

- (2, 0)(2, 5)(4, 5)(4, 8)(4, 7)(5, 7)(5, 0)
  Cette polyligne n'est pas une ligne de toits, le point (4, 7) ne peut pas exister car on a déjà un point en (4, 7) avec le segment (4, 5)-(4, 8), on ne peut pas monter puis redescendre sur la même abscisse.

1.2)

Pour qu'une polyligne soit une ligne de toits, il faut que:

- L'ordonnée du premier et du dernier point soit 0.
- Qu'une seule des deux coordonnées ne change entre chacun des points.
- Les points doivent être triés selon les x croissants.
- Il ne doit pas y avoir 3 points consécutifs sur la même abscisse ou ordonnée.

1.3)

(1, 1)(5, 13)(9, 20)(12, 27)(16, 3)(19, 0)(22, 3)(25, 0) est à la base
(1, 0)(1, 1)(5, 1)(5, 13)(9, 13)(9, 20)(12, 20)(12, 27)(16, 27)(16, 3)(19, 3)(19, 0)(22, 0)(22, 3)(25, 3)(25, 0)

Pour passer de la représentation classique à la représentation compacte il faut:

- Supprimer le premier point (il indique seulement l'abscisse du premier toit car l'ordonnée est forcément 0, or le second point l'indique aussi).
- Ensuite, on supprime tous les points modifiant l'absisse.

Il faut donc supprimer tous les points d'indice pairs.

Q2)

On prend comme taille de la fenêtre de visualisation w (largeur) et h (hauteur).

On itère sur toute la table de dimension w * h et on doit vérifier si la case fait partie d'une ligne de toits en itérant sur les triplets.
On aurait donc w * h * n opérations pour remplir la table de booléens, soit une complexité de $\theta(w * h * n)$ dans le pire des cas (si on trouve qu'une case est à 1, pas besoin de parcourir les autres bâtiments).

Ensuite pour parcourir "intelligemment" cette table, on doit partir de l'origine de la table (0, 0) et on longe l'axe des abscisses vers la droite jusqu'à arriver à la première case occupée.
Ensuite, on monte jusqu'à arriver à un zéro, ce qui signifie que l'on doit aller à droite, enfin dès qu'on arrive à un zéro, on doit d'abord tenter de monter pour vérifier si il y a un toit plus haut, sinon on doit redescendre.
Et on répète ce processus jusqu'à arriver à la fin de la table, soit le point (w, 0).

Pour généraliser, on part de l'origine:

- 1. On longe l'axe des abscisses vers la droite.
- 2. Dès qu'on trouve une case occupée :
  - Si la case du dessus est occupée, on longe l'axe des ordonnées vers le haut.
  - Sinon on longe l'axe vers le bas.
- 3. Dès qu'on arrive sur une case inoccupée, on répète le processus.

À chaque changement de direction, on ajoute un point à la liste de points de la ligne de toits (si on veut la version compacte de la ligne de toits, on ajoute que les points quand on s'apprête à aller à droite).

La complexité du parcours n'est donc plus entièrement liée au nombre de bâtiments mais plutôt à la longueur totale de la ligne de toits qu'on peut appeler l (cependant le nombre total d'opérations dépendra toujours du nombre de toits car il va impliquer plus de changements de directions).

On aurait donc w * h * n + l calculs soit une complexité totale de $\theta(n * w * h + l)$.

Cette méthode est facile à implémenter mais est très coûteuse en mémoire car on doit avoir une table de w * h cases de booléens.
De plus, pour poser les bâtiments, on doit parcourir toute la table, et ce en itérant sur les triplets, ce qui est loin d'être optimal.
Enfin, nous n'avons pas besoin de poser les toits sur une grille pour connaître les points de la ligne de toits, parcourir toute la ligne de toits est donc très coûteux et inutile.

Q3)

```
def insert_building(roof_line, triplet):
  x1, h, x2 = triplet;
  new_line = new roof_line();

  i = 0;
  n = size(roof_line);

  # On ajoute les points avant le nouvel immeuble.
  while i < n and roof_line[i].x < x1:
    add_point(new_line, roof_line[i]);
    i++;
```

```
# On ajoute le point (x1, h) si la ligne est vide ou le point n'est pas caché par la ligne.
if is_empty(new_line) or new_line[-1].y != h:
  add_point(new_line, (x1, h));


# On parcourt les points existants de la ligne de toits qui sont couverts par l'immeul
while i < n et roof_line[i].x <= x2:
  if roof_line[i].y > h:
    add_point(new_line, roof_line[i]);
  i++;

# On ajoute la fin de l'immeuble si on ne l'a pas encore ajouté.
if new_line[-1].x != x2:
  add_point(new_line, (x2, 0));


# Enfin, on ajoute les points de la ligne de toits si il y en a encore.
while i < n:
  add_point(new_line, roof_line[i]);
  i++;

return new_line;
```

On aurait ensuite une fonction prenant en paramètre plusieurs lignes de toits et qui construirait la ligne en ajoutant chacun des toits.

On parcourt tous les points de la ligne de toits actuelle pour insérer l'immeuble. Si n est le nombre de points dans la ligne de toits, l'insertion d'un immeuble est donc en $O(n)$.
Si on a m immeubles, chacun étant inséré dans une ligne de toits pouvant avoir jusqu'à n points, la complexité totale est de l'ordre de $O(m * n)$, où m est le nombre d'immeubles et n est la taille moyenne de la ligne de toits à chaque étape.

Q4)

Pour fusionner deux lignes, il faut itérer sur ces lignes jusqu'à ce que l'une ou l'autre soit vide.
Il faut stocker les deux hauteurs courantes des lignes et toujours ajouter les points avec la hauteur maximale des deux lignes mais en faisant attention à n'ajouter un point que si la hauteur à changé (par exemple au lieu d'ajouter les points (10, 15), (12, 15), on ajoute que (10, 15)).

L'algorithme **fusion** se trouve à la fin du fichier **roof_line.c**, avant la fonction **create_roof_line**.

Q5)

Pour créer une ligne de tois avec l'algorithme de fusion, on doit diviser tous les batiments jusqu'à en avoir un seul, ce batiment sera donc une ligne de toits ayant seulement deux points (x1, h) et (x2, 0).
On reconstruit ensuite la ligne de tois en fusionnant toutes les lignes deux à deux avec leur longueur qui va doubler à chaque fois dans le pire des cas (des points vont disparaître).

L'algorithme de construction de lignes de toits se trouve à la fin du fichier **roof_line.c**.

Pour le calcul de complexité j'ai décidé de compter le nombre de tour de boucles sur les lignes de toits.

Equation de récurrence :

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ n + 2T(\frac{n}{2}) & \text{sinon} \end{cases}$$

D'après le master theorem :

$\log_2 2 = 1$ on est dans le cas $d = \log_b a$, on a donc une complexité en $O(n \log n)$

On a en réalité un peu moins de $n \log_2 n$ parcours des points car l'algorithme de fusion fait disparaitre certains points des lignes, ce qui fait qu'on ne re parcourt pas certains points dans les fusions de niveaux supérieurs. On aurait réellement $n \log_2 n$ calculs dans le pire des cas (que la ligne forme un escalier symétrique verticalement avec l'axe de symétrie au milieu de la ligne).

Calculs détaillés :

$T(n) = n + 2T(\frac{n}{2})$

Niveau 0 : n

Niveau 1 : n/2 + n/2 = n

Niveau 2 : n/4 + n/4 + n/4 + n/4 = n

...

$$T(n) = n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + \ldots + 2^k T(\frac{n}{2^k})$$

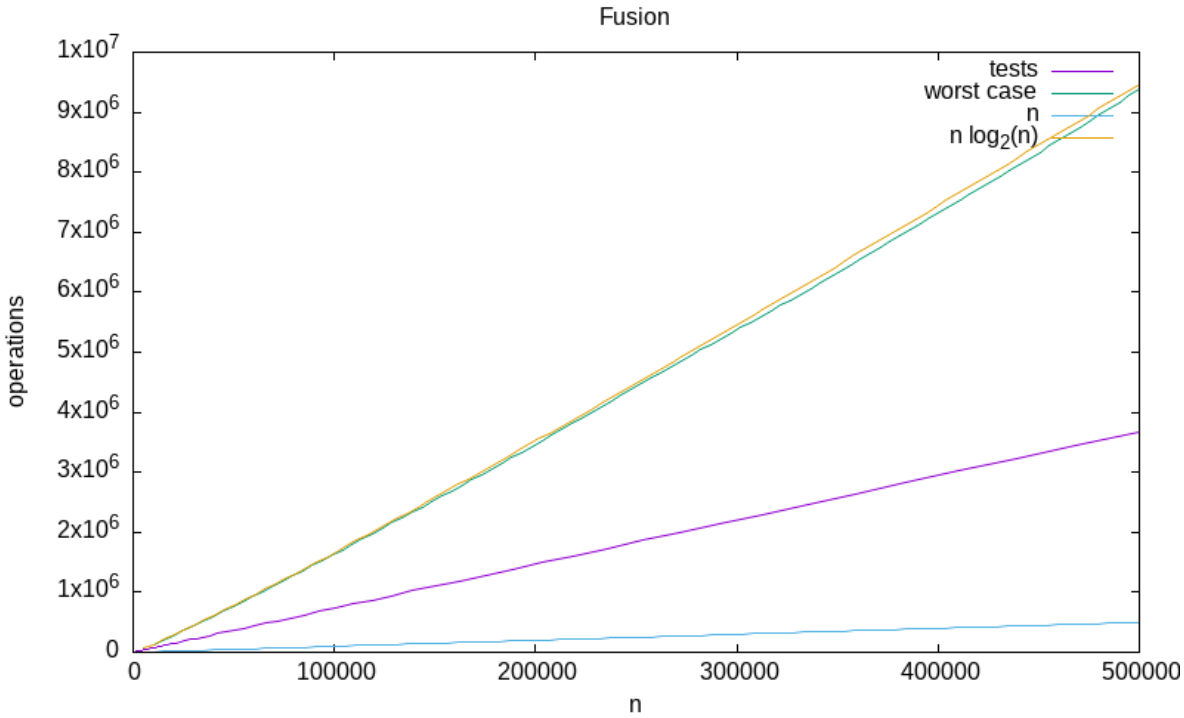
avec $k = \log_2 n$

Ce qui donne :

$$T(n) = \sum_{i=1}^{\log_2(n)} n = n \log_2 n$$

On a donc une complexité de $O(n \log n)$.

Voici le graphique représentant la complexité de l'algorithme de fusion avec les fichiers de tests mis à disposition ainsi que la complexité dans le pire des cas (voir algorithme **create_stairs_roof_line** dans le fichier main.c).



Dans le cas ou vous avez besoin de re-construire ce graphique, vous pouvez utiliser la commande `make` qui va compiler le programme, exécuter le main et les tests et générer le graphique. Il vous faut gnuplot d'installé et utiliser la commande `gnuplot plot_script.gnu` ou alors si vous avez Docker vous pouvez générer le graphique en utilisant la commande :

```
docker run --rm -v $(pwd):/work remuslazar/gnuplot plot_script.gnu
```

Il est aussi possible de générer le fichier svg correspondant à la ligne de toits, pour avoir un exemple, il faut décommenter une partie du fichier main.c.