Rapport
Structures de données II
Line segment intersection

KERCKHOFS Guillaume
BRENART Thomas

Année académique 2020-2021
Bachelier en sciences informatiques

Table des matières

1 Introduction

Ce rapport consiste à résoudre le problème du Map Overlay. Le but de notre projet est donc de trouver des intersections entre des segments donnés par un fichier texte.
On va d'abord expliquer nos choix par rapport à l'implémentation de ce projet en commençant par les structures de données utilisée pour Q et T, s'en suivra des explications sur des plus petits dommaine tel que l'event point,la sweep line,...
Ensuite, on s'attardera sur l'explication et la complexité des algorithmes importants.
Enfin,on finira par le diagramme de classe, le guide de l'utilisateur ainsi que la conclusion.

2 Explication de l'implementation

Dans cette partie, on va expliquer nos différnts choix sur l'implémentation du projet que ce soit les structures utilisées, comment on les a utilisées ainsi que sur d'autres choix comme l'event point,la sweep line, les comparaisons,...

2.1 Arbre AVL

Tout d'abord, pour faire la structure Q et T qui nous était demandé,on a choisi d'utiliser l'arbre AVL. On va donc d'abord expliquer un peu son fonctionnement avant d'aller dans les détails pour chacune des structures.
L'abre AVL est dit équilibré car il a comme principe que chacun de ses noeuds à une balance qui vaut 0,1 ou -1. Cette balance est égale à $h_2 - h_1$ où h_2 est la hauteur den fils droit et h_1 celle de son fils gauche.
Les données sont triés dans l'arbre selon un certain ordre,celui-ci sera expliqué dans les sous-sections arbre Q et arbre T car ils fonctionnent tout les deux avec un ordre bien précis.

2.1.1 Arbre Q

Tout d'abord l'ordre utilisé dans l'arbre Q est défini par les coordonnées des points que l'on ajoute dans Q. On a 4 cas à prendre en compte.Prenons $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$,deux points. On a alors:

- $y_1 > y_2, P_1$ est alors considéré comme plus petit que P_2.
- $y_1 = y_2, x_1 < x_2$, alors P_1 est encore considéré comme le plus petit.
- $y_1 = y_2, x_1 = x_1$, les deux point sont égaux.
- Dans les autres cas, P_1 est plus grand.

Si un point est plus petit qu'un autre, il sera à sa gauche dans Q et inversément s'il est plus grand.

S'il y a égalité, on ajoutera juste le segment donné,s'il est non null, dans la liste des segments du même point dans Q car les doublons ne sont pas autorisés.

Cela permet d'avoir un ordre dans Q qui nous permettra de rajouter/chercher/retirer des points de Q et de connaître facilement le prochain point à gèrer, qui est le point le plus petit.Pour connaître ce point à analyser, il faut prendre le point qui est le plus à gauche en partant de la racine(Prendre le fils gauche du point courrant jusqu'à tomber sur un noeud sans fils gauche) et on aura donc le point le plus petit de l'arbre.
Les noeuds de Q ont donc chacun un fils gauche noté Q_{left},un fils droit Q_{right},une data Q_{data} qui est un Point et une liste regroupant tout les segments possèdant comme "upper-point" Q_{data}.Les références vers les fils peuvent être null, si les deux le sont,c'est que le noeud est une feuille.

Les algorithmes lié à cette structure seront détaillés dans la partie 3.1,page 6.

2.1.2 Arbre T

Pour T, contrairement à Q,il y a juste la data T_{data} qui est un Segment,le fils gauche T_{left} et droit T_{right}.

Une autre particularité de T est que les doublons sont autorisés. En effet dans T, T_{data} représente le segment stocké dans la feuille la plus à droite de son sous-arbre gauche.

Pour l'ordre défini dans T, on utilise un peu le même principe que pour Q mais au lieu de regarder si un point est plus petit qu'un autre, ici on regarde si un segment est plus petit qu'un autre. Pour cela, on va juste comparer les x des segments pour un y donné. Prenons le cas où nous avons deux segments non horizontaux.Pour un même y, on calcul la valeur de x,on a alors x_1, le x du premier segment et x_2 celui du deuxième tel que:

- $x_1 > x_2$,alors le premier segment est plus grand.
- $x_1 < x_2$, alors le premier segment est plus petit.
- $x_1 = x_2$, on doit calculer alors x_1^*, x_2^* pour un certain y^* qui est le y avec la valeurs la plus élevé des deux lower-point des segments comparés. On refait la comparaison des x, si on retombe sur une égalité, on a deux segments égaux.

Plaçons nous maintenant dans un cas où on a un segment horizontal,on doit alors comparer les x sauf que pour un certain y, le segment horizontal à un nombre illimé de valeurs pour x. On a donc choisit dans notre algorithme qu'on passerait en paramètre le x de l'event point (c.f section 2.2.2,page 6). On calcul donc juste la valeur de x de l'autre segment pour ce y.

On a alors x_{hor}, le x du segment horizontal et x_1 celui de l'autre segment.On les compare:

- $x_{hor} \geq x_1$,alors le segment horizontal est plus grand.
- $x_{hor} < x_1$, alors le segment horizontal est plus petit.

Le cas du $\geq$ est utile lors de la réinsertion dans Q après la gestion d'un event point. Cette utilité sera expliqué un peu plus loin. Si le deuxième segment est aussi horizontal, on est dans le cas où on compare les deux même segments car on suppose qu'aucun segments ne se superposent dans ce problème.

Maintenant qu'on sait comment comparer nos segments, on peut définir un ordre dans T avec pour un certain noeud T, un sous-arbre gauche T_{left} où tout les segments sont plus petit que le segment T_{data} et un sous-arbre gauche T_{right} où tout les segments sont plus grand que le segment T_{data}.

Cette ordre est utile lorsque l'on cherchera les intersections.En effet, lors de la recherche de la prochaine intersection, on cherche à trouver des segments qui sont prochent l'un de l'autre afin de vérifier si dans un futur proche il se croiseront. Avec notre ordre établi, les segments sont tous stockés dans des feuilles et pour une certaine feuille, si on regarde sa feuille à gauche(resp. droite), elle représente le segment le plus proche sur sa gauche(resp. droite) sur la map.