

**Rapport**  
**Structures de données II**  
**Line segment intersection**

KERCKHOFS Guillaume  
BRENART Thomas

Année académique 2020-2021  
Bachelier en sciences informatiques

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Explication de l'implémentation</b>	<b>3</b>
2.1	Arbre AVL . . . . .	3
2.1.1	Arbre Q . . . . .	3
2.1.2	Arbre T . . . . .	4
2.2	Autres . . . . .	6
2.2.1	Sweep line . . . . .	6
2.2.2	Event-point . . . . .	6
2.2.3	Comparaison . . . . .	6
<b>3</b>	<b>Explication et complexité des algorithmes</b>	<b>7</b>
3.1	Arbre Q . . . . .	7
3.2	Arbre T . . . . .	7
<b>4</b>	<b>Diagramme de classe</b>	<b>7</b>
<b>5</b>	<b>Guide de l'utilisateur</b>	<b>7</b>
<b>6</b>	<b>Conclusion</b>	<b>7</b>

## 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 utilisées pour Q et T, s'en suivra des explications sur des plus petits domaines tel que l'évent 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'implémentation

Dans cette partie, on va expliquer nos différents 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'évent 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'arbre AVL est dit équilibré car il a comme principe que chacun de ses nœuds a une balance qui vaut 0, 1 ou -1. Cette balance est égale à  $h_2 - h_1$  où  $h_2$  est la hauteur du fils droit et  $h_1$  celle de son fils gauche.

Les données sont triées dans l'arbre selon un certain ordre, celui-ci sera expliqué dans les sous-sections arbre Q et arbre T car ils fonctionnent tous 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.
- $y_1 = y_2, x_1 = x_2$ , les deux points 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 inversement 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 gérer, 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 courant 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 possédant 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 liés à cette structure seront détaillés dans la partie 3.1, page 7.

### 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 si le noeud courant est un noeud.

On a donc aussi les segments stockés dans les feuilles. Tout les segments présents dans  $T$  sont dans une et une seule feuille, par contre, tout les segments de  $T$  ne sont pas présents dans un noeud (le noeud le plus à droite de l'arbre).

Cette particularité sera à prendre en compte lors de l'implémentation de l'arbre AVL.

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 illimité de valeurs pour  $x$ . On a donc choisit dans notre algorithme qu'on passerait en paramètre le  $x$  de l'évent 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 droite  $T_{right}$  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 proche 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.

Cela facilitera les recherches d'intersections. C'est d'ailleurs pour cette raison que comme c'était expliqué auparavant, un segment horizontal est considéré plus grand s'il a la même valeur de  $x$  avec un autre segment non horizontal, c'est pour pouvoir faire en sorte de tester les intersections futurs et que le segments ne soit pas mal mit dans  $T$ .

Les algorithmes lié à cette structure seront détaillés dans la partie 3.2, page 7.

## 2.2 Autres

Dans cette section ,on va expliquer certains termes que l'on utilisera plus tard pour plus de compréhension, les choix d'implémentation pour simplifier le travail,...

### 2.2.1 Sweep line

On commence donc par l'explication de ce qu'est la sweep-line. C'est une particularité surtout utilisé lors de l'affichage de la résolution qui permet d'avoir un rendu visuel sur ce que fait l'algorithme. Cette sweep-line se déplace du haut vers le bas de la carte.

Elle va donc s'arrêter à chaque point dans l'ordre dans lequel ils ont été analysés par l'algorithme en les mettant en évidence.

### 2.2.2 Event-point

Un event-point est un point de la carte et plus précisément, celui sur lequel se trouve l'algorithme à un certain moment. On a choisit d'utiliser cette variable car on trie nos données selon un certain y, qui provient de cette event-point. Cela nous permettra quand on s'occupera de l'event point précédent de retrouver nos segments sans problème car on aura la valeur du y selon laquelle ils sont triés.

L'event-point changera quand on devra réinsérer les données dans T et donc les segments réinsérer seront remis dans T dans un nouvel ordre.

### 2.2.3 Comparaison

Pour la comparaison, pour simplifier la lisibilité ainsi que pour plus de facilité, on a conçu une classe Comparasion. Le but étant d'implémenter les comparaisons de double avec une précision de  $1e-8$ . On a donc fait 4 méthodes, 3 sont des comparasion avec bt qui vérifie que  $x_1 > x_2$ , st vérifie que  $x_1 < x_2$  et equal vérifie l'égalité en utilisant la formule:

$$|x_1 - x_2| < 1e - 8.$$

La 4ème méthode est un mélange des 3 qui retourne 0 si égalité,-1 si  $x_1 < x_2$  et 1 sinon.

### **3 Explication et complexité des algorithmes**

#### **3.1 Arbre Q**

#### **3.2 Arbre T**

### **4 Diagramme de classe**

### **5 Guide de l'utilisateur**

### **6 Conclusion**