Exercice Préliminaire – Projet de Structures de Données II

A. Dubrulle, J. Main

31 janvier 2019

1 Introduction

Nous nous intéressons ici au problème suivant : « étant donnés un arbre BSP T représentant une scène dans le plan $\mathbb{R}^2$ et deux points x et y du plan, déterminer si le segment $[x, y]$ est dans l'arbre T ».

Posons les notations qui seront utilisées tout au long de la résolution de l'exercice. On dénote par T à la fois la racine et l'arbre correspondant.

Si T est une feuille, on dénote par S_T l'ensemble des (fragments de) segments contenus dans cette feuille. Notez que S_T contient au plus un élément (par définition de BSP).

Si T est un noeud interne, on note D l'hyperplan affin stocké en ce noeud. Ce dernier est donné par une équation $f_D(x_1, x_2) = 0$ où $f_D : \mathbb{R}^2 \to \mathbb{R} : (x_1, x_2) \mapsto ax_1 + bx_2 + c$ pour certains $a, b, c \in \mathbb{R}$. Ceci sépare le plan en trois parties : la droite D,

$$D^+ = \{(x_1, x_2) \in \mathbb{R}^2 \mid f_D(x_1, x_2) > 0\}$$

et

$$D^- = \{(x_1, x_2) \in \mathbb{R}^2 \mid f_D(x_1, x_2) < 0\}$$

Ceci motive la notation T^- (resp. T^+) pour le fils gauche (resp. droit) de T représentant les fragments contenus dans D^- (resp. D^+). L'ensemble des segments contenus dans D est noté S_T de la même manière que pour les feuilles.

2 Raisonnement mathématique

2.1 Cas de base

La question présente deux cas de base exclusifs : l'arbre T est réduit à une feuille ou les points x, y délimitant le segment recherché sont contenus dans l'hyperplan stocké en la racine de T.

Si T est réduit à une feuille, il suffit de tester si le segment stocké en cette feuille (s'il existe) correspond à $[x, y]$. Si ce n'est pas le cas ou que la feuille est vide, alors l'algorithme retourne faux.

Si l'arbre T n'est pas réduit à une feuille et que les points x et y sont contenus dans D, alors par convexité de D, le segment est contenu dans D. Le problème se réduit alors à la recherche d'une donnée au sein d'une liste chaînée.

2.2 Cas général

Supposons que T possède deux fils (ie. T n'est pas une feuille). Discutons les différents cas.

Si $x, y \in D^+$ (resp. D^-), alors le segment $[x, y] \subseteq D^+$ (resp. D^-) par convexité. Par définition d'arbre BSP, le segment recherché est contenu dans T si et seulement s'il est dans T^+ (resp. T^-). Il suffit donc de poursuivre la recherche récursivement dans T^+ (resp. T^-).

Si $x \in D$ et $y \in D^+$ (resp. D^-), alors le segment $[x, y]$ est contenu dans le demi-plan D^+ (resp. D^-). Il suffit donc de poursuivre la recherche récursivement comme précédemment. Le cas $y \in D$ et $x \in D^+$ (resp. D^-) est analogue au cas précédent.

Il reste deux cas à considérer : le premier est $x \in D^+$ et $y \in D^-$ et le second est $x \in D^-$ et $y \in D^+$. Les deux cas étant similaires, nous supposerons que $x \in D^+$ et $y \in D^-$ (les calculs sont indépendants du cas considéré). Le segment $[x, y]$ intersecte D en un point qu'on nommera z.

Savoir si le segment $[x, y]$ se trouve dans T revient donc à savoir si les segments $[x, z]$ et $[z, y]$ (le segment est fragmenté) sont éléments de T et par conséquent de faire deux recherches récursivement (cfr. cas précédent) et de retourner vrai si et seulement si les deux appels retournent vrai.

Cependant, il faut connaître z pour appliquer ce raisonnement. Nous allons donc montrer comment déterminer z à partir de x, y et f_D. Posons $g : \mathbb{R}^2 \to \mathbb{R}$ la fonction définie par $g(x_1, x_2) = ax_1 + bx_2$ pour tout $(x_1, x_2) \in \mathbb{R}^2$. Nous avons que $f_D = g + c$ et la fonction g est linéaire.

De plus, nous avons que $f_D(z) = 0$ et $\exists t \in [0, 1]$ tel que $z = tx + (1 - t)y$ [1]. Donc, par linéarité de g :

$$0 = f_D(z) = f_D(tx + (1 - t)y) = t(g(x) - g(y)) + g(y) + c \tag{1}$$

Par conséquent,

$$t = \frac{(-c - g(y))}{(g(x) - g(y))} = \frac{-(f_D(y) + 2c)}{f_D(x) - f_D(y)} \tag{2}$$

L'expression de t est bien définie car $g(x) \neq g(y)$ car $g(y) < -c < g(x)$.

3 Algorithme

La vérification de la présence d'un segment donné est détaillé à l'algorithme 1.

Les détails suivants sont omis afin de ne pas alourdir l'algorithme : l'appartenance dans les ensembles stockés en T (ensembles notés S_T) et le calcul de l'intersection du segment et de la droite stockée en T.

Afin de vérifier l'appartenance d'un segment à un ensemble, il suffit de parcourir l'ensemble, représenté ici par une liste chaînée, et de tester l'égalité entre les éléments et le segment en entrée.

Il suffit donc d'expliciter l'égalité de segments et d'appliquer un algorithme de recherche classique dans une liste chaînée. Deux segments sont égaux si et seulement s'ils ont les mêmes extrémités, c'est-à-dire :

$$[x_1, y_1] = [x_2, y_2] \iff (x_1 = x_2 \wedge y_1 = y_2) \vee (x_1 = y_2 \wedge y_1 = x_2)$$

1. En effet, on a par définition : $[x, y] = \{tx + (1 - t)y \mid t \in [0, 1]\} = \{ty + (1 - t)x \mid t \in [0, 1]\}$

Algorithme 1 Recherche_segment(T, x, y)

Entrée: Un arbre BSP T, deux points x et y du plan.
Sortie: Vrai si et seulement si le segment $[x, y]$ est contenu dans T.

1: **si** T est réduit à une feuille **alors**
2: **retourner** $[x, y] \in S_T$
3: **fin si**
4: **si** $f_D(x) > 0$ **alors**
5: **si** $f_D(y) \geq 0$ **alors**
6: **retourner** Recherche_segment(T^+, x, y)
7: **sinon**
8: $z \leftarrow D \cap [x, y]$
9: **retourner** Recherche_segment(T^+, x, z) $\wedge$ Recherche_segment(T^-, z, y)
10: **fin si**
11: **sinon**
12: **si** $f_D(x) = f_D(y) = 0$ **alors**
13: **retourner** $[x, y] \in S_T$
14: **sinon si** $f_D(y) \leq 0$ **alors**
15: **retourner** Recherche_segment(T^-, x, y)
16: **sinon**
17: $z \leftarrow D \cap [x, y]$
18: **retourner** Recherche_segment(T^-, x, z) $\wedge$ Recherche_segment(T^+, z, y)
19: **fin si**
20: **fin si**

Il reste à discuter le calcul de l'intersection d'un segment et d'une droite dans le cas où elle n'est pas vide. Or, ces considérations sont détaillées dans le raisonnement mathématique (se référer à l'égalité (2) et se rappeler que $z = tx + (1 - t)y$).

4 Complexité

4.1 Coûts locaux

Etudions le coût local dans une feuille. Par définition, S_T contient au plus une donnée. Donc, le coût local par feuille est en temps constant $O(1)$.

Intéressons-nous désormais au coût local par noeud interne. Dans un premier temps, si le segment est contenu dans la droite stockée dans le noeud, alors on a un coût linéaire en le nombre de fragments dans S_T (borné par le nombre de segments employés dans la construction de l'arbre).

Sinon, seules des opérations élémentaires sont effectuées (opérations arithmétiques, appels de fonctions et comparaisons), ce qui entraîne un coût local en $O(1)$.

4.2 Nombre d'appels récursifs (pire des cas)

Remarquons d'abord que dans l'algorithme, chaque noeud est visité au plus une fois. Par conséquent, le nombre de noeuds visités maximal correspond au nombre de noeuds de T. Il est possible que tous les noeuds de l'arbre soient visités ; si à chaque noeud interne traité, (le fragment considéré du) segment recherché intersecte la droite correspondant au

noeud, cela entraîne un appel récursif sur chacun des fils du noeud. Le nombre d'appels récursifs est en $O(N)$ où N correspond au nombre de noeuds de l'arbre passé en paramètre.

Par conséquent, on en déduit une majoration grossière de la complexité en $O(N \times n)$ où n est le nombre de segments stockés dans T, étant donné que chaque liste contient au plus n fragments.

4.3 Nombre d'appels récursifs (cas simple)

Supposons que $[x, y]$ n'intersecte aucune droite contenue dans T. Alors l'algorithme suit un chemin de l'arbre. Nous avons donc au maximum h appels récursif où h est la hauteur de l'arbre. On en déduit donc une complexité en $O(h)$ avec h la hauteur de T.

4.4 Nombre d'appels récursifs (en moyenne)

Etudions de manière plus attentive la complexité. On se pose la question suivante : « Quel est le nombre moyen de segments s_j dans T tel que la droite passant par s_j intersecte $[x, y]$? »

Posons $s = [x, y]$.

Supposons dans la suite que s n'est inclus dans aucune droite stockée dans T.

Il arrive donc que le segment s intersecte certaines droites. Utilisons un raisonnement similaire à celui présenté dans [1]. On suppose que l'arbre BSP est construit à l'aide d'auto-partitions [2] et de manière aléatoire en supposant chaque permutation équiprobable. On suppose T construit sur base de $S = \{s_1, ..., s_n\}$. Posons pour s' un segment du plan $l(s')$ la droite passant par les extrémités de s' (elle contient donc s').

Soit $i \in \{1, ..., n\}$. On définit :

$$d(s, s_i) = \begin{cases} \#\{\text{segments entre } s \text{ et } s_i \text{ intersectant } l(s_i)\} & \text{si } l(s_i) \text{ intersecte } s \\ +\infty & \text{sinon} \end{cases}$$

Etudions la probabilité que $l(s_i)$ coupe s (notée $P(l(s_i) \text{ coupe } s)$), c'est-à-dire lors de la séparation du plan par $l(s_i)$, s est fragmenté. Supposons que $l(s_i)$ intersecte s (sinon la probabilité est nulle). On a donc $d(s, s_i)$ finie.

Soit $\sigma \in S_n$ (l'ensemble des permutations de n éléments, également appelé groupe symétrique d'indice n).

S'il existe $k \in \{1, \ldots, n\}$ tel qu'on coupe le plan par $l(s_k)$ avant de le couper par $l(s_i)$ et que s et s_i se retrouvent de part et d'autre de $l(s_k)$ alors $l(s_i)$ ne coupera pas s (car s_i et s ne se trouveront pas dans le même demi-plan).

En particulier, il faut que les indices des segments situés sur la droite $l(s_i)$ entre les segments s et s_i apparaissent après i dans σ. Il faut donc pour tous ces segments que leur indice k vérifie $\sigma(i) < \sigma(k)$ (c'est-à-dire la coupe selon $l(s_i)$ est effectuée avant celle par $l(s_k)$). Donc i doit apparaître en premier dans la permutation σ.

Par conséquent, on a la majoration :

$$\begin{aligned} P(l(s_i) \text{ coupe s}) &\leq \frac{\#\text{permutations commencant par 1 dans } S_{d(s,s_i)+1}}{\#\text{permutations dans } S_{d(s,s_i)+1}} \\ &= \frac{d(s, s_i)!}{(d(s, s_i) + 1)!} = \frac{1}{d(s, s_i) + 1}. \end{aligned}$$

2. C'est-à-dire les coupes du plan sont réalisées à partir de droites engendrées par un des segments de l'ensemble sur base duquel l'arbre est construit

Notez que les permutations commençant par 1 sont considérées pour clarifier les notations, étant donné qu'on considère les permutations commençant par un élément fixé.

Le nombre moyen de coupes effectuées sur s est donc, par linéarité de l'espérance :

$$\begin{aligned}\sum_{k=1}^{n} P((s_k) \text{ coupe } s) &\leq \sum_{k=1}^{n} \frac{1}{d(s, s_k) + 1} \\ &\leq 2 \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \frac{1}{k+1} \\ &\leq 2 \ln\left(\frac{n}{2}\right)\end{aligned}$$

La deuxième inégalité est une conséquence du fait que pour une distance donnée par rapport à s, il existe au plus deux segments réalisant cette distance.

Nous avons vu dans la section précédente que le coût local est en temps constant car on ne parcourt S_T que si T est une feuille par l'hypothèse faite (que s n'est inclus à aucune droite).

Ce qui nous permet de conclure que le nombre de noeuds où sont effectués deux appels récursifs est en $O(\ln(n))$. On en déduit que le nombre de chemins suivis dans l'arbre est en $O(\ln(n))$. Ceci fournit une complexité totale en $O(h \times \ln(n))$.

Références

[1] M. D. Berg, O. Cheong, and M. V. Kreveld, *Computational Geometry : Algorithms and Applications*, pp. 265–266. Springer Berlin Heidelberg, 2008.