Projet noté – Intersection de segments

Alexandra ZAHARIA (alexandra.zaharia@u-psud.fr)

Étant donné un ensemble de segments dans le plan définis par les coordonnées de leurs extrémités, ce projet vise à déterminer les paires de segments qui s'intersectent ainsi que les coordonnées de leurs points d'intersection.

1 Le problème

La FIGURE 1 ci-contre montre six segments (numérotés de 0 à 5 et définis par les coordonnées des leurs extrémités). La liste des paires de segments qui s'intersectent ainsi que les coordonnées de leurs points d'intersection est aussi fournie dans la figure.

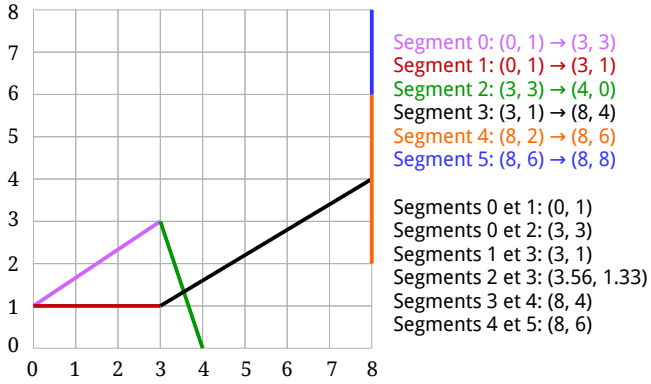

FIGURE 1 – Segments dans le plan et leurs points d'intersection

1.1 Les données d'entrée

Vous disposerez d'un fichier d'entrée contenant un entier positif n sur la première ligne, suivi de n lignes désignant chacune un segment dans le plan. Un segment est défini par les coordonnées de ses extrémités : il s'agit de quatre réels séparés par des espaces, sous la forme $x_1\ y_1\ x_2\ y_2$. Un exemple de fichier d'entrée associé à la FIGURE 1 pourrait être celui dans la FIGURE 2a.

```
6
0 1 3 3
0 1 3 1
3 3 4 0
3 1 8 4
8 2 8 6
8 6 8 8
```

(a) Fichier d'entrée

```
0 1 0.00 1.00
0 2 3.00 3.00
1 3 3.00 1.00
2 3 3.56 1.33
3 4 8.00 4.00
4 5 8.00 6.00
```

(b) Fichier de sortie

FIGURE 2 – Exemple de fichiers d'entrée et de sortie associés à l'exemple dans la FIGURE 1

1.2 La sortie

Après avoir déterminé quelles sont les paires de segments qui s'intersectent, vous allez créer un fichier de sortie contenant sur chaque ligne une paire de segments S_i et S_j qui s'intersectent, ainsi que les coordonnées (x, y) du point d'intersection de S_i et S_j. Vous représenterez les segments S_i et S_j par les indices qui leur sont associés dans le fichier d'entrée, à partir de 0. Ainsi, les indices des segments sont des entiers positifs, et les coordonnées x et y sont des réels (que vous allez afficher avec deux chiffres après la virgule). Les quatre valeurs sur chaque ligne seront séparées par des espaces.

Sur l'exemple du fichier d'entrée ci-dessus (voir 1.1), le segment allant de (0, 1) à (3, 3) a l'indice 0, et le segment allant de (8, 6) à (8, 8) a l'indice 5. Sur ce même exemple, le fichier de sortie attendu est celui de la FIGURE 2b.

2 Déterminer si deux segments s'intersectent

2.1 Le produit vectoriel

Étant donnés deux vecteurs $\vec{u} = (x_1, y_1)$ et $\vec{v} = (x_2, y_2)$, leur produit vectoriel se définit comme suit :

$$u \times v = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$$

2.2 Orientation relative

Par rapport à un vecteur donné $\vec{p}$, on peut identifier des vecteurs en sens horaire et antihoraire. La FIGURE 3 en donne une représentation explicite.

Soient $p = (x_p, y_p)$ un point dans le plan et S un segment défini par ses deux extrémités $u = (x_u, y_u)$ et $v = (x_v, y_v)$. On définit $p - u = (x_p - x_u,\ y_p - y_u)$ et $v - u = (x_v - x_u,\ y_v - y_u)$ comme étant respectivement les vecteurs $\overrightarrow{up}$ et $\overrightarrow{uv}$. On calcule le produit vectoriel $(p - u) \times (v - u)$:

- Si le produit vectoriel est positif, alors $\overrightarrow{up}$ est dans le sens horaire par rapport à $\overrightarrow{uv}$.
- Si le produit vectoriel est négatif, alors $\overrightarrow{up}$ est dans le sens antihoraire par rapport à $\overrightarrow{uv}$.
- Si le produit vectoriel vaut zéro, alors les points u, v et p sont colinéaires.

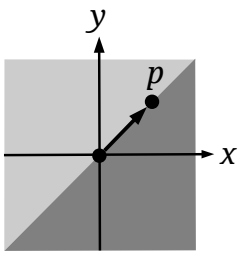

FIGURE 3 – **Orientation relative.** La région gris foncé contient les vecteurs dans le sens horaire par rapport à p, alors que la région gris clair contient les vecteurs dans le sens antihoraire par rapport à p.

2.3 Critère d'intersection

Soient les segments S et T. S est défini par ses extrémités $u = (x_u, y_u)$ et $v = (x_v, y_v)$. T est défini par ses extrémités $w = (x_w, y_w)$ et $t = (x_t, y_t)$.

On dit que S **enjambe** T si les points u et v ont des orientations relatives différentes par rapport à T, c'est-à-dire si :

- par rapport à $\overrightarrow{wt}$, $\overrightarrow{wu}$ est dans le sens horaire et $\overrightarrow{wv}$ est dans le sens antihoraire, ou
- par rapport à $\overrightarrow{wt}$, $\overrightarrow{wv}$ est dans le sens horaire et $\overrightarrow{wu}$ est dans le sens antihoraire.

Les segments S et T s'intersectent si :

- S enjambe T et T enjambe S, ou
- une extrémité de S se trouve sur le segment T ou une extrémité de T se trouve sur le segment S.

Remarque. On exclut ici les segments qui se superposent (la FIGURE 4 en montre les deux cas).

FIGURE 4 – Cas exclus

3 Déterminer les coordonnées du point d'intersection

3.1 L'équation de la droite passant par un segment

Par deux points distincts dans le plan passe précisément une droite. Soient $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$ deux points. L'équation de la droite passant par p_1 et p_2 est $y = mx + b$, où :

- $m = \frac{y_2 - y_1}{x_2 - x_1}$ est la **pente** de la droite ;
- $b = y_1 - mx_1 = y_2 - mx_2$ est l'**ordonnée à l'origine**.

S'il s'agit d'un segment vertical (dans ce cas, $x_1 = x_2$), la pente n'est pas définie. L'équation de la droite passant par un segment vertical est $x = b$, où b vaut $x_1 = x_2$.

3.2 Le coordonnées du point d'intersection

Soient S et T deux segments qui s'intersectent (voir 2.3) dans un point $p = (x, y)$. Les coordonnées de p seront déterminées en résolvant le système de deux équations linéaires à deux inconnues donné par les équations des droites passant par S et T.

4 Le projet

4.1 Les structures de données

Il va falloir définir certaines structures de données (référez-vous au TP n° 5 sur les structures) :

- Un point dans le plan est décrit par ses deux coordonnées réelles.
- Un segment dans le plan est décrit par ses deux extrémités (deux points dans le plan).
- L'équation d'une droite passant par un segment est décrite par deux valeurs réelles correspondant respectivement à la pente et à l'ordonnée à l'origine, si le segment n'est pas vertical. La structure devrait donc disposer d'un membre booléen permettant de distinguer les droites passant par des segments verticaux.

4.2 Astuces

Le fichier d'entrée contient n segments que vous devez stocker. Pour chaque segment il va falloir retenir la pente et l'ordonnée à l'origine associés à la droite passant par ses extrémités, ou le fait que le segment est vertical. L'idée serait d'utiliser deux tableaux de structures (l'un pour les segments et l'autre pour les équations associées à ces segments). La taille maximale de ces tableaux sera définie par une directive du préprocesseur. Vous devez vérifier lors de la lecture du fichier d'entrée que le nombre de segments que vous allez lire ne dépasse pas la taille maximale de vos tableaux.

Faites attention à ne pas afficher en sortie la même paire de segments deux fois (par exemple, si vous affichez la paire de segments d'indices i et j, vous n'afficherez pas la paire j et i).

Rappelez-vous qu'un fichier en-tête (`.h`) contient typiquement les déclarations de variables, les déclarations et définitions de structures de données, les directives de préprocesseur ainsi que les déclarations des fonctions qui seront définies dans les fichiers `.c` associés (ayant le même nom). Par ailleurs, un fichier `.c` doit inclure son en-tête associé pour pouvoir utiliser, par exemple, les structures de données définies dans le fichier en-tête.

Lorsque un fichier en-tête est inclus par plusieurs fichiers `.c` (ce qui sera le cas par exemple pour l'en-tête décrivant un segment), le compilateur considérera qu'il s'agit de re-déclarations de symboles et refusera de produire l'exécutable. Ceci est normal car chaque directive `#include` se traduit dans une opération de copier/coller du contenu du fichier `.h` à la place de la directive `#include` correspondante. Pour éviter ce problème, il faudra faire des *include guards*. Ceci consiste à instruire le préprocesseur à n'inclure un fichier en-tête que s'il n'a pas déjà été inclus. Par exemple, supposons que vous avez un fichier `entete.h` contenant uniquement la définition d'une structure appelée `foo` et possédant un membre entier `bar`. Faire les *include guards* vous donnera le fichier `entete.h` suivant :

```
1 #ifndef ENTETE_H
2 #define ENTETE_H
3
4 struct foo {
5     int bar;
6 };
7
8 #endif
```

Les lignes 1 et 2 définissent (`#define`) un nouveau symbole appelé `ENTETE_H` s'il n'est pas déjà défini (`#ifndef`). Le contenu à proprement parler du fichier `entete.h` (la définition de la structure `foo`) sera donc inclus dans un fichier source `.c` uniquement si le symbole `ENTETE_H` n'a pas été défini auparavant.

(*Facultatif*) Il existe un type booléen en C. Si vous souhaitez l'utiliser, vous aurez besoin de l'en-tête `stdbool.h`.

(*Facultatif*) Vous pouvez également vous renseigner sur les paramètres en ligne de commande. Ceux-là vous permettront d'utiliser votre programme en lui spécifiant directement dans la console, par exemple, le nom du fichier d'entrée et le nom du fichier de sortie à obtenir. Il s'agira de fournir à votre fonction `main` deux arguments : `int main(int argc, char **argv)`.

4.3 L'organisation du projet

Votre projet comportera plusieurs fichiers `.c` et `.h`. Un seul des fichiers `.c` contiendra une fonction `main`.

Pensez à séparer les opérations similaires dans leurs propres fichiers `.c`/`.h`. Par exemple tout ce qui relève des entrées/sorties ne sera pas mis ensemble avec le test pour savoir si deux segments s'intersectent.

Quant aux fonctions, votre projet comportera au moins une fonction pour chacun des aspects évoqués dans les sections 1.1, 1.2, 2.1, 2.2, 2.3, 3.1 et 3.2.

4.4 Le `makefile`

Vous allez créer un `makefile` permettant de compiler le projet. Référez-vous au TP n° 6.

4.5 Évaluation

Vous m'enverrez par mail une archive contenant votre projet (les fichiers `.c`, `.h` et le `makefile`).

Ce projet est un travail **individuel**. La date limite du rendu du projet est fixée au dimanche le **22 janvier 2017** à minuit. Un projet rendu en retard ne sera pas corrigé. Votre projet doit pouvoir être compilé et exécuté sur les machines de TP du bâtiment 640.

En plus du fonctionnement du programme, votre code sera évalué selon des critères supplémentaires :

- La correction des résultats obtenus, ainsi que le temps d'exécution. À titre d'exemple, sur un fichier d'entrée contenant 10 000 segments, le programme devrait s'exécuter dans moins de deux minutes.
- Le code doit être lisible (indentation, espacement).
- Les noms des variables doivent être pertinents.
- Les passages importants du code doivent être commentés de sorte à ce qu'on puisse comprendre le code. (Évitez les commentaires inutiles du type "on incrémente i".)
- Votre `Makefile` devra invoquer l'option `-Wall` pour le compilateur `gcc`. Tous les avertissements seront pris en compte dans l'évaluation.