Programmation Effective – TD 11 : Intersections dans un ensemble de segments 2D

matthieu.gallet@ens-lyon.fr

mardi 29 avril 2008

1 Intersections dans un ensemble de segments 2D

On considère dans cette partie un ensemble W de n segments L_i, et on cherche à calculer un ensemble $L = \{I_j\}$ de points d'intersections de ces segments. Il y a dans le pire des cas $\mathcal{O}(n^2)$ intersections, plus exactement $n(n-1)/2$ si tous les segments se croisent. Dans le pire des cas, calculer toutes ces intersections demandera un temps $\mathcal{O}(n^2)$. Un algorithme simple et brutal consiste à prendre chacune des $\mathcal{O}(n^2)$ paires de segments et à voir s'il y a intersection. Si dans le pire des cas, la complexité est optimale, on peut utiliser des méthodes plus intelligentes lorsqu'il y a assez peu d'intersections. Nous allons voir un algorithme très classique et nettement plus performant, l'algorithme de Bentley-Ottman.

Cet algorithme est également connu sous le nom de *sweep line algorithm*, autrement dit une ligne qui va balayer les différents segments pour détecter plus rapidement les intersections. L'idée est donc d'utiliser une droite, notée SL qui va faire un balayage, mettons de gauche à droite, et noter quand elle passe sur un des segments L_i et quand elle tombe sur une intersection. SL va donc également être une structure de données stockant les segments qui la croisent. Le cœur du problème est d'implémenter efficacement ce balayage.

Pour ce faire, on commence par trier les extrémités E_{i0} et E_{i1} des segments, généralement par abscisses croissantes puis par ordonnées croissantes. Si l'on suit cet ordre, SL fera bien un balayage de gauche à droite. À tout instant de l'algorithme, SL va commencer par intersecter un segment à son extrémité gauche (la structure de données sous-jacente va donc devoir stocker ce segment à ce moment), puis à son extrémité droite, et SL doit alors le supprimer de sa base de données, tout en maintenant sa liste triée par un ordre « dessus/dessous ». Ainsi, quand on ajoute ou supprime un segment, il faut d'abord déterminer sa place, par exemple par une recherche dichotomique, qui se fait, comme chacun sait en temps $\mathcal{O}(\log n)$. Enfin, quand deux segments se croisent, on échange leurs positions dans la liste triée.

On garde donc une queue d'événements X dont les éléments vont changer la liste SL. Initialement, X contient toutes les extrémités des segments, dans l'ordre du balayage. Cependant, quand une intersection entre deux segments est trouvée, elle est ajoutée à X, toujours dans le même ordre, en faisant attention à ne pas ajouter deux fois les mêmes intersections. Quand un insère un segment dans SL, on détermine leurs intersections potentielles avec les autres segments de SL, et on ajoute les intersections effectives à la file d'attente des événements. Quand on traite un événement correspondant à une intersection, on l'ajoute à la liste L et on réorganise SL.

Demeure la question de la détermination des intersections :

- Quand un segment est ajouté à SL, on détermine s'il intersecte ses voisins immédiats,
- Quand un segment est supprimé de SL, ses anciens voisins supérieurs et inférieurs deviennent alors voisins et on détermine s'ils s'intersectent.
- Lors d'une intersection, les deux segments échangent leurs positions, et on a alors de nouvelles intersections à déterminer ;

Les opérations à réaliser sur SL sont donc l'ajout, la recherche, l'échange et la suppression. Une structure adaptée est donc un arbre binaire équilibré, et on obtient une complexité totale en $\mathcal{O}\left((n+k)\log n\right)$.

Un pseudo-code complet est donné par l'Algorithme 1.

2 Applications

Résolvez tout de même les problèmes suivants :
- Mobile Phone Coverage (`http://acm.uva.es/p/v6/688.html`).
- The Skyline Problem (`http://acm.uva.es/p/v1/105.html`),
- Urban Elevations (`http://acm.uva.es/p/v2/221.html`),
- Polygon Inside A Circle (`http://acm.uva.es/p/v104/10432.html`),
- Ancient Village Sports (`http://acm.uva.es/p/v104/10451.html`),
- Useless Tile Packers (`http://acm.uva.es/p/v100/10065.html`),

Algorithm 1 Bentley-Ottman()

ENTRÉES: P liste de points du plan $\mathcal{P}$
  Initialiser une file d'attente X contenant toutes les extrémités des segments
  Trier X dans l'ordre lexicographique (x, y)
  Initialiser un arbre binaire équilibré vide SL
  Initialiser L la liste vide des intersections.
  tantque X est non vide **faire**
    Soit E le prochain événement de x
    si E est une extrémité gauche **alors**
      Soit seg_E le segment correspondant à E
      Ajouter seg_E à SL
      Soit seg_A le segment au-dessus de seg_E dans SL
      Soit seg_B le segment en-dessous de seg_E dans SL
      si $I = seg_A \cap seg_E$ existe **alors**
        Ajouter I à X
      finsi
      si $I = seg_B \cap seg_E$ existe **alors**
        Ajouter I à X
      finsi
    sinon si E est une extrémité droite **alors**
      Soit seg_E le segment correspondant à E
      Soit seg_A le segment au-dessus de seg_E dans SL
      Soit seg_B le segment en-dessous de seg_E dans SL
      Supprimer seg_E de SL
      si $I = seg_A \cap seg_B$ existe et n'est pas déjà dans X **alors**
        Ajouter I à X
      finsi
    sinon si E est une intersection **alors**
      $L \leftarrow L \cup E$
      Soit seg_{E1} et seg_{E2} (avec seg_{E1} au-dessus de seg_{E2}) les segments se coupant en E
      Échanger leurs positions pour que seg_{E2} soit au-dessus de seg_{E1}
      Soit seg_A le segment au-dessus de seg_{E2} dans SL
      Soit seg_B le segment en-dessous de seg_{E1} dans SL
      si $I = seg_A \cap seg_{E2}$ existe et n'est pas déjà dans X **alors**
        Ajouter I à X
      finsi
      si $I = seg_B \cap seg_{E1}$ existe et n'est pas déjà dans X **alors**
        Ajouter I à X
      finsi
    finsi
    Supprimer E de X
  fin tantque
  renvoyer L