Complexité et Algorithmique - L3 2014-15
Projet

Date limite : Dimanche 19 avril 2015.
Composition des groupes à envoyer au plus tard le mercredi 28 janvier.

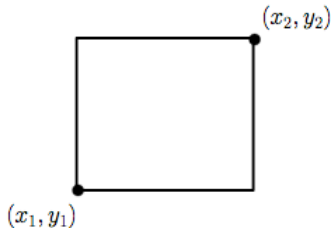
Le problème.

Un problème central dans le domaine de la géométrie algorithmique est de décider si deux objets se coupent. Cette opération est beaucoup utilisée, par exemple, en robotique pour vérifier qu'un robot évite tous les obstacles de son environnement. Puisque le robot et les obstacles peuvent être des objets très compliqués, il s'avère efficace de les approcher par les boîtes englobantes. Pour vérifier si deux objets se coupent, on vérifie d'abord si les boîtes englobantes se coupent, une opération peu coûteuse. Il est nécessaire de tester l'intersection des deux objets seulement si leurs boîtes englobantes se coupent.

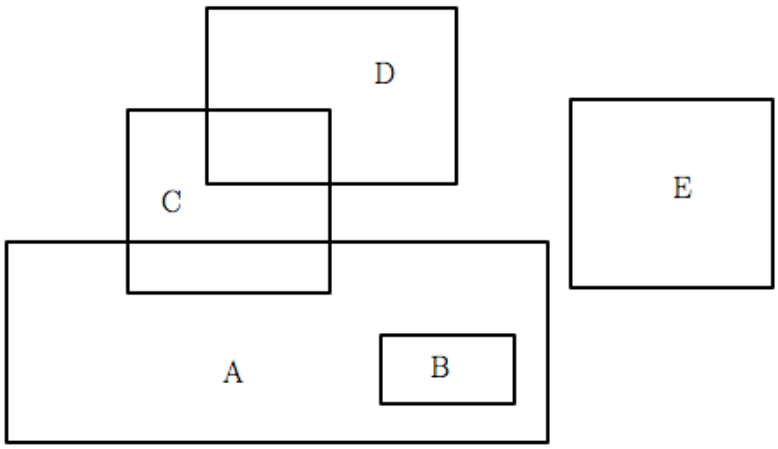
L'objectif de ce projet est de développer, programmer et analyser des algorithmes pour trouver toutes les intersections parmi un ensemble de boîtes. Pour simplifier, nous travaillons en deux dimensions, et donc, on s'intéresse aux intersections entre rectangles. De plus, on suppose que les cotés des rectangles sont parallèles aux axes x et y.

Nous allons étudier deux algorithmes dont l'un est une version très simplifiée d'un algorithme très efficace permettant de calculer les intersections de 500 000 boites 3D en moins de 10 secondes [Zomorodian et Edelsbrunner, Fast Software for Box Intersections, 2002].

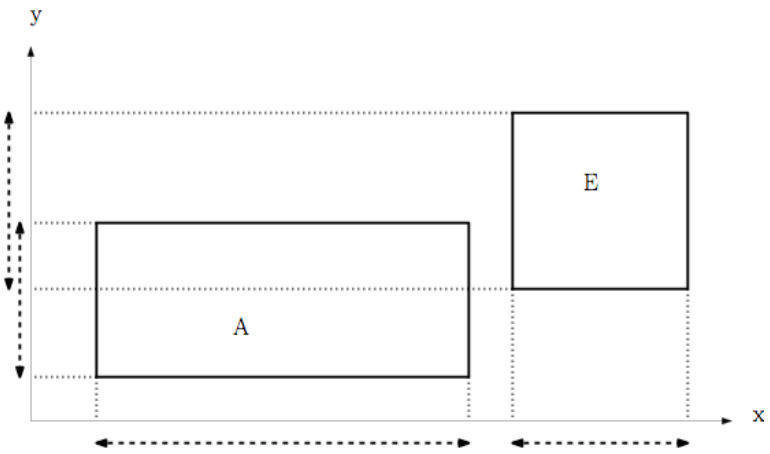
L'entrée de votre programme consiste en un ensemble de **n** rectangles. Chaque rectangle est représenté par 4 entiers, les coordonnées x et y des coins inférieur gauche et supérieur droit.



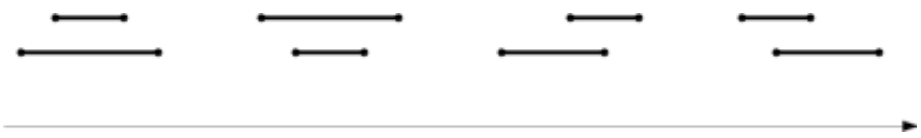
La sortie du programme est le nombre, **k ,** de paires de rectangles qui se coupent. Par exemple, dans la figure suivante, il y a k=3 paires de rectangles qui se coupent : A et B, A et C, C et D.



Pour tester si deux rectangles se coupent, on peut utiliser la propriété suivante : deux rectangles se coupent si et seulement si ils se coupent en x et en y. Par exemple, dans la figure suivante, on voit que A et E ne se coupent pas parce qu'ils ne se coupent pas en x (ils s'intersectent en y mais ce n'est pas suffisant).



On réduit donc le problème au problème de savoir si deux intervalles se coupent. Il y a quatre manières dont ça peut arriver, lesquelles sont illustrées dans la figure suivante :



On observe que deux intervalles se coupent si et seulement si l'extrémité gauche d'un des intervalles est contenue dans l'autre intervalle.

Un algorithme simple, qu'on appelle l'algorithme *ToutesLesPaires*, pour résoudre le problème consiste à tester chaque paire de rectangles, deux à deux.

Un deuxième algorithme, qu'on appelle l'algorithme *Balayage,* commence par trier les rectangles selon la coordonnée x de leur coin inférieur gauche ; soit L la liste des rectangles triés. On traite les rectangles dans cet ordre, en cherchant les paires de rectangles qui se coupent en x. Chaque fois que l'on trouve deux rectangles qui se coupent en x, on vérifie si ces rectangles se coupent en y aussi ; si oui, les deux rectangles se coupent.

Pour tirer profit du fait que les rectangles sont triés en x on utilise la stratégie suivante. Considérer les rectangles l'un après l'autre, dans l'ordre de L, en commençant par le premier. Soit R le rectangle courant. Balayer la liste L de rectangles dans l'ordre en partant de R et en s'arrêtant au premier segment, D, dont la coordonnée x du coin inférieur gauche est plus grande que la coordonnée x du coin supérieur droit de R. Parmi les rectangles après R dans la liste L, ceux qui coupent R en x sont exactement ceux après R et avant D. Maintenant, on recommence en remplaçant R par son successeur dans la liste.

Ce que vous devez faire.

1. Implantez les algorithmes *ToutesLesPaires* et *Balayage*. (Les algorithmes de tri utilisés devront être de complexité O(n log n).) Quelles sont les complexités dans le pire cas de ces deux algorithmes en fonction du nombre n de rectangles ?
2. Pouvez vous analyser la complexité (dans le pire cas) de *Balayage* en fonction de n et du nombre k_x de paires de rectangles qui se coupent en x ? C'est-à-dire borner asymptotiquement le nombre d'opérations par un O d'une fonction dépendant de n et $k_{x,,}$ qui soit plus petit que la complexité calculé en 1. lorsque k_x est « petit ».
3. Testez vos algorithmes sur des petits exemples de 4 ou 5 rectangles. Dessinez quelques jeux de tests et donnez la sortie correspondante de vos algorithmes.
4. Estimez empiriquement le temps de calcul en moyenne de vos algorithmes. Pour ce faire, générez des ensembles aléatoires de différents nombres de rectangles. Considérez séparément trois jeux de tests. Dans les trois jeux de tests, chaque coordonnée x et y du sommet bas-gauche est tirée aléatoirement dans l'intervalle [0,n] (*le plan dans lequel se répartissent les n segments grandit avec n*). Dans le jeu (i), les coordonnées x et y du sommet haut-droit sont également tirées aléatoirement dans l'intervalle [0,n]. Dans le jeu (ii), les rectangles sont des carrés de largeur et hauteur 1. Dans le jeu (iii), la largeur et la hauteur du rectangle sont tirées aléatoirement dans l'intervalle [1, √n]. *Attention, le n dans ces définitions de tirage aléatoire est le même n que celui du nombre de segments*.

   Dessiner un graphique qui montre le rapport entre n, le nombre de rectangles, et t, le temps de calcul. Quelle est la complexité empirique de vos algorithmes sur chacun des deux jeux de test : O(n), O(n log n), O(n √n), O(n^2), autre ? Quel algorithme est le plus efficace (discutez) ?

   Pensez à faire attention aux échelles dans vos graphiques et à expliquer vos choix d'échantillons de valeurs de n et nombres de tests effectués pour chaque n. Expliquez également comment vous estimez les complexités entre O(n), O(n log n), etc.

5. De la même manière, estimez empiriquement le nombre moyen de paires de rectangles qui se coupent dans chacun des trois cas (i), (ii), (iii). Fournir des graphiques.

Points importants.

Vous devez remettre un programme en C++ ou Java documenté (avec commentaires clairs et courts) ainsi qu'un document. Il est important que votre programme suive les principes de la programmation orientée objet.

Votre document doit être clair et structuré (faites attention à la présentation). Il doit contenir

- les renseignements nécessaires pour une bonne utilisation du programme
- une description des objets et des algorithmes
- un dossier de tests : les tests qui ont été effectués pour garantir le bon fonctionnement du programme; tests sur les rectangles aléatoires; les résultats obtenus en forme de graphique
- les réponses à toutes les questions posées.

La rigueur de la présentation et la qualité du document seront prises en compte dans l'évaluation.

Vous devez travailler en groupes de 5 et valider votre groupe par email au plus tôt. Votre projet doit être remis pour le 19 avril 2015 à minuit (à déposer dans l'interface de l'ENT). Les soutenances auront lieu le 29 avril 2015. Vous devrez prévoir 10 à 15 mn de présentation, laquelle sera suivie d'une quinzaine minutes de questions/discussion.