- TP 1 : Intersection de segments -

Le but de ce TP est d'implémenter l'algorithme d'intersection de segments fonctionnant en $O(n \log n)$ vu en cours.
Le langage à utiliser est laissé libre. Toutefois, des primitives d'affichage, de tri, une classe gérant les arbres rouges et noirs, ainsi que des trames de programme sont fournies en C++ à l'adresse :

http ://www.lirmm.fr/∼bessy/AlgoGeo/accueil.html

- Intersections de segments -

Les points du plan sont repérés par leurs coordonnées supposées entières et appartenant à un domaine $[0, x_{max}] \times [0, y_{max}]$. Pour l'affichage en postscript, on prendra $x_{max} = 612$ et $y_{max} = 792$. Les logiciels *evince*, *gv*, *ggv* ou *kghostview* permettent un affichage standard des documents postscript.

- Exercice 1 - Génération de points du plan -

Ecrire une fonction *SegmentsAuHasard* qui génère n segments dans le plan (n est fixé à 10). Les segments sont stockés dans le tableau *segments[2*n][2]*. Le point le plus à gauche du segment i est stocké dans *segments[2*i]* et ses coordonnés, (x_i, y_i), sont choisies aléatoirement dans $[20, 400] \times [100, 700]$. Les coordonnées du second sommet du segment i sont stockées dans *segments[2*i+1]* et sont choisies aléatoirement dans $[x_i + 1, x_i + 100] \times [y_i - 50, y_i + 50]$.
Une fois les segments générés, la procédure *AffichageSegments*, appelée dans le *main*, génère le fichier *Segments.ps*.

- Exercice 2 -

Compléter les fonctions *CleInferieure* et *Intersectent*.
La fonction *CleInferieure* renvoie vrai si et seulement si le point p_3 est à gauche du segment p_1p_2 orienté de p_1 à p_2.
La fonction *Intersectent* renvoie vrai si et seulement si les segments p_1p_2 et p_3p_4 s'intersectent.
Pensez à tester vos fonctions...

- Exercice 3 - Test d'intersection -

Compléter la fonction *Intersection* qui renvoie vrai si et seulement si il existe une intersection dans l'ensemble de segments stockés dans le tableau *segments*.
Le tableau *Tri* contient les extrémités des segments triées par abscisse croissante. Ces sommets sont repérés par leur indice dans le tableau *segments*, le segment i ayant pour extrémités les sommets $2i$ et $2i + 1$.
La variable *ordre* est un arbre rouge et noir codant un ordre total et qui supporte les primitives suivantes :

- *ordre.Insere(j)* insère le segment j dans l'ordre.
- *ordre.Predecesseur(j)* renvoie le numéro du segment précédant j dans l'ordre codé par l'arbre, si j correspond au segment minimum de l'ordre, la fonction renvoie -1.

- *ordre.Successeur(j)* renvoie le numéro du segment suivant j dans l'ordre codé par l'arbre, si j correspond au segment maximum de l'ordre, la fonction renvoie -1.
- *ordre.Supprime(j)* supprime le segment j dans l'ordre.

- Exercices supplémentaires -

- Exercice 4 - Polygone simple -

Ecrire une fonction qui prend en entrée une suite de n segments et qui teste si ces segments forment un polygone simple ou non. Votre fonction devra fonctionner en temps $O(n \log n)$.

- Exercice 5 - Points d'intersection -

Ecrire une fonction qui prend en entrée une suite de n segments et qui imprime toutes les intersections formées par ces segments. Votre fonction devra fonctionner en temps $O((n+k)\log(n+k))$, où k est le nombre d'intersections formées par tous les segments. Pour cela, vous pouvez utiliser le code fourni pour gérer les ARN, mais en le modifiant possiblement.

- Exercice 6 - Intersection de disques -

Implémenter l'algorithme de recherche d'intersections de disques vu en TD. Dans un premier temps, on ne cherchera qu'à détecter une seule intersection, puis on les trouvera toutes.