- TP 2. Algorithme de Kruskal. -

Le but de ce TP est de calculer, pour un ensemble V de points du plan, un arbre couvrant $T = (V, A)$ qui vérifie que la somme des longueurs des arêtes de A est minimale. Le calcul de cet arbre s'effectue par l'algorithme de Kruskal.

Langage. Programme en C++. Votre programme pourra contenir :

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;
typedef struct coord{int abs; int ord;} coord;

int
main()
{
  int n;             //Le nombre de points.
  cout << "Entrer le nombre de points: ";
  cin >> n;
  int m=n*(n-1)/2;   // Le nombre de paires de points.
  coord point[n];    // Les coordonnees des points dans le plan.
  int edge[m][3];    // Les paires de points et le carre de leur longueur.
  int arbre[n-1][2]; // Les aretes de l'arbre de Kruskal.

  return 0;
}
```

Ce début de code est récupérable là : `http://www.lirmm.fr/~montassier/hlin501/tp2.cc`

!!!!! Pensez à tester chaque code produit sur de petits exemples !!!!!

- Exercice 1 - Création d'un ensemble aléatoire V de n sommets dans le plan.
L'ensemble de sommets V est $\{0, \ldots, n-1\}$. L'ensemble des positions des éléments de V est stocké dans un tableau de coordonnées **point** de taille n vérifiant que **point**[i].abs est l'abscisse du sommet i, comprise entre 0 et 612, et **point**[i].ord est l'ordonnée du sommet i, comprise entre 0 et 792. Pour information, le format US-Letter, par défaut sur de nombreux afficheur postscript, a pour dimension 612 points par 792 points...

Écrire une fonction *void pointrandom(int n, coord point[])* qui engendre aléatoirement le tableau **point**.

- Exercice 2 - Création du tableau des distances.
Écrire une fonction *void distances(int n, int m, coord point[], int edge[][3])* qui engendre le tableau **edge** de taille $m \times 3$ de telle sorte que :

- Pour chaque paire $\{i, j\}$ avec $i < j$, il existe un k qui vérifie **edge**[k][0] $= i$ et **edge**[k][1] $= j$.
- L'entrée **edge**[k][2] est le carré de la distance euclidienne du point correspondant au sommet i au point correspondant au sommet j.

- Exercice 3 - Tri du tableau edge.

Écrire une fonction *void tri(int m, int edge[][3])* qui trie le tableau **edge**, selon l'ordre croissant des valeurs de **edge**[k][2]. Le but de ce TP n'étant pas le tri, on pourra se limiter à un simple tri à bulles (tant qu'il existe deux entrées consécutives qui ne sont pas croissantes, on les inverse).

- Exercice 4 - Calcul de l'arbre couvrant de poids minimum.

Écrire une fonction *void kruskal(int n, int edge[][3], int arbre[][2])* qui applique l'algorithme de Kruskal au tableau d'arêtes **edge** et construit le tableau **arbre** qui contient les $n-1$ arêtes de l'arbre couvrant de poids minimum. On pourra reprendre la fonction composantes du TP1, et y apporter des modifications mineures.

- Exercice 5 - Affichage.

Utiliser la fonction `affichageGraphique` que vous pouvez trouver à l'adresse

http ://www.lirmm.fr/∼montassier/hlin501/affichage.cc

afin d'afficher le résultat dans le fichier `Exemple.ps`.
L'appel se fera par *affichageGraphique(n,point,arbre) ;*

- Exercice 6 - Pour aller plus loin.

Apporter les améliorations ou modifications suivantes :

- Si ce n'est pas déjà le cas, utiliser la version optimisée de l'agorithme *composante* pour implémenter l'algorithme de Kruskal.
- Améliorer les performances de votre algorithme en utilisant un tri plus efficace, par exemple tri fusion.
- Montrer que les arêtes de l'arbre obtenu ne peuvent se croiser.
- Utiliser d'autres distances (Manhattan, sup,...) pour créer votre arbre.

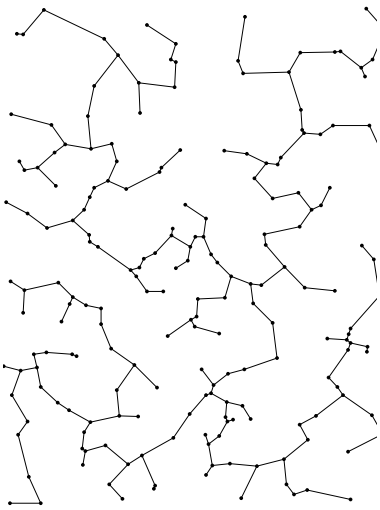

FIGURE 1 – Un exemple d'arbre de Kruskal.