

## - TP 4. Plus courts chemins. Algorithme de Dijkstra. -

Le but de ce TP est de calculer un arbre de plus courts chemins (en terme de distance euclidienne) issu d'un sommet dans un graphe  $G$  dont les sommets sont des points du plan et les arêtes sont toutes les paires  $xy$  de sommets dont la distance est inférieure à une valeur fixée  $d_{max}$ .

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

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <fstream>
#include <math.h>

using namespace std;

int main(){
    int n;                //Le nombre de points.
    int m;                //Le nombre d aretes.
    cout << "Entrer le nombre de points: ";
    cin >> n;

    int dmax=50;          // La distance jusqu'a laquelle on relie deux points.

    vector<int> voisin[n]; // Les listes de voisins.
    int point[n][2];       // Les coordonnees des points.

    int d[n];              // La distance a la racine.
    int arbre[n-1][2];     // Les aretes de l'arbre de Dijkstra.
    int pere[n];           // La relation de filiation de l'arbre de Dijkstra.

    return 0;
}
```

Ce début de code est récupérable là : <http://www.lirmm.fr/~bessy/GLIN501/TP/tp4.cc>

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

### - Exercice 1 - Création du graphe.

Reprendre la fonction `void pointrandom(int n, int point[][2])` du TP2 qui engendre aléatoirement le tableau **point** représentant un ensemble aléatoire de  $n$  points dans le plan. Rappelons que **point** est de taille  $n \times 2$ , l'abscisse du point  $i$  (entre 0 et 612) est stockée dans **point** $[i][0]$  et l'ordonnée (entre 0 et 792) est stockée dans **point** $[i][1]$ .

Ecrire une fonction `void voisins(int n, int dmax, int point[][2], vector<int> voisin[])` qui pour tout sommet  $i$  construit la liste **voisin** $[i]$  vérifiant qu'un point  $j \neq i$  apparait dans **voisin** $[i]$  si et seulement si la distance euclidienne du point  $i$  au point  $j$  est au plus égale à  $d_{max}$ .

**- Exercice 2 - Affichage du graphe.**

Ecrire `void AffichageGraphe(int n, int point[][2], vector<int> voisin[])`, inspirée de la fonction d'affichage du TP2, qui permet d'afficher le graphe créé dans l'exercice 1 à l'aide d'un fichier *Graphe.ps*. Tester sur au moins 300 points.

**- Exercice 3 - Arbre de Dijkstra.**

Ecrire une fonction `void dijkstra(int n, vector<int> voisin[], int point[][2], int pere[])` sur le modèle de l'algorithme vu en cours. La racine de l'arbre des plus courts chemins est le sommet 0. En sortie, le tableau **pere** représente l'arbre des plus courts chemins. Ainsi, tout sommet  $i$  distinct de la racine et accessible depuis celle-ci vérifie que **pere**[ $i$ ] est différent de -1, valeur donnée à l'initialisation.

**- Exercice 4 - Affichage de l'arbre.**

Ecrire une fonction `int construirearbre(int n, int arbre[][2], int pere[])`, qui remplit le tableau **arbre** avec toutes les arêtes  $i$  **pere**[ $i$ ] et retourne le nombre  $k$  de ces arêtes (c'est à dire le nombre de points accessibles depuis la racine moins un.)

Utiliser la fonction `void AffichageGraphique(int n, int k, int point[][2], int arbre[][2])` du TP2 pour créer un fichier *Arbre.ps* qui représente l'arbre.

**- Exercice 5 - Pour aller plus loin.**

Répondre ou améliorer les points suivants :

- Lorsque  $d_{max}$  est très grand, que constatez-vous ?
- Les arêtes de l'arbre de Dijkstra peuvent-elles se couper ?
- Essayer des métriques différentes (sup, manhattan).
- Utiliser une structure de tas pour le calcul du sommet non traité à distance minimale.

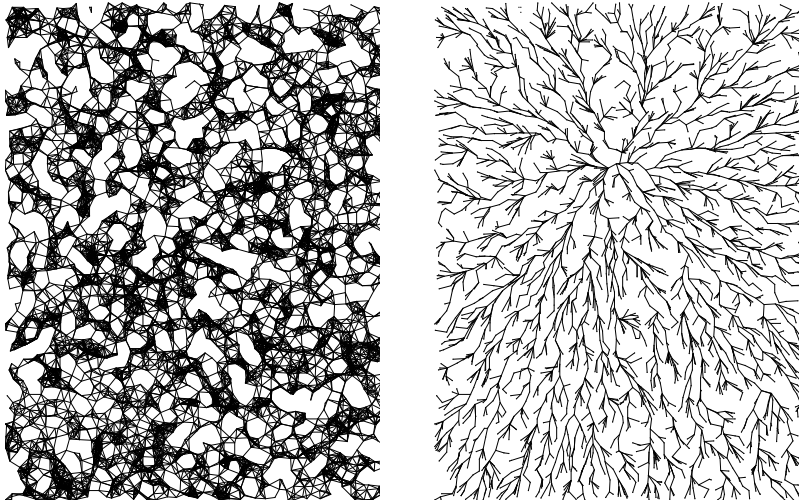


FIGURE 1 – Un exemple d'arbre de plus courts chemins.