TP2 : Arbre des plus courts chemins

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

Le fichier `tp2.txt` sur le serveur pédagogique contient l'entête ci-dessous que vous pouvez réutiliser ainsi que deux fonctions `afficheGraphe()` qui permet d'obtenir un fichier postscript du dessin du graphe et `afficheArbre()` qui permet d'obtenir un fichier postscript du dessin de l'arbre.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <fstream>
#include <math.h>
using namespace std;

const int n=4000;               // Le nombre de points.
const int dmax=20; // 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 arbre[n-1][2];              // Les aretes de l'arbre de Dijkstra .
int pere[n];     // La relation de filiation de l'arbre de Dijkstra.
```

Exercice 1- *Création et affichage du graphe*

Les sommets du graphe sont numérotés $\{0, 1, \ldots, n-1\}$ et leur position dans le plan, générée aléatoirement est stokkée dans le tableau `point[n][2]` ($point[i][0]$ est l'abscisse du point i, entre 0 et 611 et $point[i][1]$ est l'ordonnée entre 0 et 791).

Écrire la fonction `void generegraphe(int n, int point[][2])` qui remplit aléatoirement le tableau `point`.

Afficher ensuite le graphe à l'aide de la fonction `afficheGraphe(n, dmax, point)` qui construit le fichier `Graphe.ps`

Exercice 2- *Création des listes de voisins*

Écrire une fonction `void voisins()` qui pour tout sommet i construit la liste voisin[i] vérifiant qu'un point j apparaît dans voisin[i] si et seulement si la distance euclidienne du point i au point j est au plus égale à dmax.

Exercice 3- *Arbre de Dijkstra*

Écrire une fonction `void Disjkstra()` 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. Tout sommet i distinct de la racine et accessible depuis celle-ci vérifie $pere[i] \neq -1$, (-1 étant la valeur donnée à l'initialisation).

Exercice 4- *Affichage de l'arbre*

Écrire une fonction `int construitArbre()` qui remplit le tableau `arbre` avec toutes les arêtes $ipere[i]$ et retourne le nombre k de ces arêtes (c'eat-à-dire le nombre de points accessibles depuis la racine moins un).

Utiliser ensuite la fonction `afficheArbre(n, k, point, arbre)` pour créer le fichier Arbre.ps.

Exercice 5- *Pour aller plus loin*

- Lorsque dmax est très grand, que constatez-vous ?
- Les arêtes de l'arbre de Disjktra peuvent-elles se couper ?
- Essayer des métriques différentes (sup, manhattan).

Quelques méthodes de la classe vector :

```
- vector<int> vect // déclare la variable vect comme vecteur d'entiers
- vect.assign(i,j) // initialise vect avec i copies de j
- vect.push_back(i) // empile la valeur i sur vect
- vect.pop_back() // dépile vect
- vect.back() // retourne la valeur en haut de vect
- vect.size() // retourne le nombre d'élement de vect
- vect.empty() // retourne VRAI lorsque vect est vide
```

Parcours d'un vector :

```
vector <int> v;
vector <int>::iterator Iter;
vect.push_back(10);
vect.push_back(20);
for(Iter=v.begin(); Iter<v.end(); Iter++)
  cout << *Iter << " ";
```