DUFFAUT Julien
PHILIPPE Joshua

# Devoir n°2 – Ultramétrique sur un nuage de points

## Notes d'utilisation :

**Compilation :**

- Utiliser la commande "*make all*" dans le dossier décompressé pour compiler le programme.
- Il est possible d'utiliser "*make clean*" pour nettoyer les fichiers .o générés.

**Utilisation du programme :**

Une fois compilé, le lancement du programme affiche une interface graphique affichant le graphe généré aléatoirement à partir de l'image *image.pgm* dans le dossier.
Si le dossier ne contient pas de fichier *image.pgm*, le programme en générera un automatiquement.
Le poids de l'arbre est affiché à tout moment dans le coin en haut à droite du programme.

En **cliquant** sur un noeud, il est mémorisé, si un autre est mémorisé et que l'ultramétrique a été calculée, une arête spécifique pour ces deux noeud est affichée en vert.

Plusieurs touches du clavier servent à utiliser le logiciel. Celles-ci sont listées dans le coin en bas à gauche du programme :

- ***O* :** Calculer l'ACM(Arbre Couvrant Minimal) en utilisant l'algorithme de Prim (version normale) et l'afficher.
- ***P* :** Calculer l'ACM en utilisant l'algorithme de Prim (version Tas) et l'afficher.
- ***J* :** Calculer l'ACM en utilisant l'algorithme de Kruskal (version normale) et l'afficher.
- ***K* :** Calculer l'ACM en utilisant l'algorithme de Kruskal (version avec forêts) et l'afficher.
- ***U* :** Calculer les distances ultramétriques du graphe et écris le tableau dans un fichier texte (*ultrametrics.txt*). Dans le mode ultramétrique. Un clic souris sur deux sommets permet d'afficher la distance ultramétrique entre eux.
- ***R* :** Revenir à l'arbre de départ (Reset).
- ***E* :** Enlever la sélection actuelle de sommet dans le mode ultramétrique.
- ***Q* :** Quitter le logiciel.

## Structuration du programme :

Le programme a été découpé en plusieurs modules.

- ***Devoir2*** **:** Contient le programme principal, la création de l'interface, du nuage de points.
- ***Node*** **:** Contient la création de la classe Node, correspondant à un noeud d'un graphe et de la structure Edge utilisée pour les arêtes associées.
- ***Graph*** **:** Contient la classe Graph, composée de d'un ensemble de Node. Ses méthodes incluent les algorithmes de Prim et de Kruskal, avec leurs 2 versions respectives.
- ***Prim*** **:** Contient les implémentations (basique et avec tas) de l'algorithme de Prim de la classe Graph.
- ***Kruskal*** **:** Contient les implémentations (basique, avec forêts et une version pour l'ultramétrique) de l'algorithme de Kruskal de la classe Graph.
- ***EdgeHeap*** **:** Contient la classe EdgeHeap, utilisée lors de la version de l'algorithme de Prim utilisant les tas d'arêtes.
- ***Image*** **:** Contient la classe GrayImage, qui stocke des informations sur des images en niveaux de gris et qui permet de lire ces images au format PGM (Portable Graymap Format).
- ***Ez-draw*** **et** ***Ez-draw++*** **:** Contiennent le code des bibliothèques ez-draw (prévue à la base pour le langage C) et son adaptation en C++ ez-draw++. EZ-Draw permet de faciliter l'affichage graphique des formes.

## Choix :

**C++ :** Le langage de programmation que nous connaissons le mieux. Orienté objet et assez complet pour nous faciliter l'implémentation des algorithmes.

**Ez-draw++ :** Pour l'IHM, nous avons choisi d'utiliser la librairie ez-draw++ pour le C++. Il s'agit d'un outil permettant de développer des interfaces graphiques simplement, mais suffisant à nos besoins pour ce projet.

**Format d'image PGM :** Le PGM (Portable GrayMap Format) est un format simple d'image en niveaux de gris. Ayant travaillés avec ce format à l'IUT, nous avons pu ré-utiliser certaines fonctions pour lire ce format que nous avions créées à l'époque. Nous utilisons donc une image en pgm pour créer nos nuages de points.

## Difficultés :

**Structure des données et conception générale :** Nous avons du passer beaucoup de temps au début du projet à concevoir nos structures de données de manière cohérente et utile pour les algorithmes à implémenter.

**Cycles dans Kruskal (version basique) :** Un de nos principaux problèmes d'implémentation fut la détection de cycles dans la version basique de Kruskal en particulier. En effet nous effectuions des recherches en profondeurs de l'arbre pour trouver si il existait déjà un chemin entre les deux sommets de l'arête avant de l'ajouter. Cela conduisait à une boucle infinie dans le cas ou le noeud n'était pas trouvé.