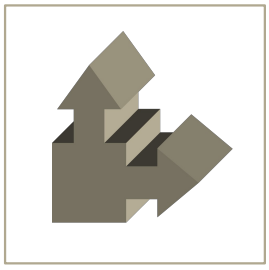
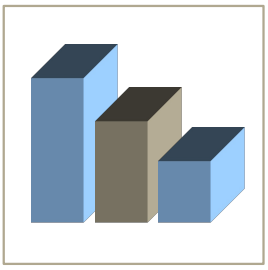




# Devoir n°2 – Arbre Couvrant Minimal

**Second devoir d'algorithmique**

Julian Bruyat
Julien Estaillard
2016-2017

## Introduction:

Les objectifs définis dans le projet sont

- d'implémenter les algorithme permettant de construire un ACM (Arbre Couvrant Minimal) (Algorithmes de Kruskal, Prim et leurs améliorations) et afficher les arbres trouvés ainsi que les graphes d'origine.
- Trouver l'ultramétrique entre deux points dans un graphe.

Nous avons utilisé Java pour l'implémentation du code, notre binôme ayant plus travaillé dans ce langage ce choix nous a paru évident.

## Structures:

Les structures ont été codées dans l'optique Java qui favorise les abstrations.

Ainsi, nos graphes continennent une liste d'arêtes. Chaque arête contient les deux sommets qu'elle affecte et son poids. De plus, c'est une manière qui nous semblait plus simple car plus intuitive qu'une matrice.

Notre programme est constitué de quatre classes :

Classe Arete: Une arête est composé de deux sommet et d'un poids.

Classe Sommet: Possède un nom pour être différencier et les coordonnées (x,y) du sommet sur le graphe

Classe Graph: Correspond à une liste de sommet et d'arêtes.

Classe Tas: Nous avons implémenter à la main la structure de tas min sur les arêtes afin d'en contrôler précisément l'implémentation.

## Implémentation des algorithmes :

Prim:

Pour cette algorithme en partant d'un point pris arbitrairement (le premier qu'on a ajouté dans le graphe) nous recherchons l'arête traversante de poids le plus faible. Pour éviter les cycle nous utilisons une fonction qui regarde si l'arête traversante selectionnée fait parti des sommets de l'ACM.

La fonction ‘‘nombreDeSommetsAppartenantA(Liste<Sommet>)" renvoie le nombre de sommet appartenant à l'objet "Arete" parmis la Liste mise en argument. L'arête est traversante si cette fonction renvoie 1 (on passe en argument la liste des sommets de l'ACM)

Kruskal:

Pour trier les arêtes du graph nous utilisons une fonction existante de Java.

Il suffit ensuite d'ajouter toute les arêtes à l'arbre qui ne créent pas de cycle, jusqu'à en avoir suffisament.

Pour ne pas faire de cycle, nous n'avons pas utilisé l'Union Find de base mais nous avons crée une fonction qui parcours l'arbre déjà crée. Si à partir d'un sommet de l'arête elle arrive à atteindre l'autre sommet en utilisant les arêtes déjà dans l'ACM, nous n'ajoutons pas l'arête candidate.

Prim par Tas:

En utilisant la structure de tas codée à la main, nous nous assurons que le fonctionnement du tas est optimal pour notre utilisation.

Dans notre cas, au lieu de chercher à chaque fois quelles arêtes sont traversantes dans le graphe d'origine et de selectionner la plus petite, on ajoute chaque arête au tas qui concerne le sommet que l'on vient d'ajouter. Il faut tout de même vérifier lors de l'extraction que l'arête extraite est (encore) traversante.

Kruskal Union Find via les forêts :

Cette méthode présentée en cours utilise des arbres (forêts) pour déterminer si l'on va faire un cycle ou pas. Il suffit de regarder la racine des deux sommet de l'arête sélectionnée, si c'est la même on va faire un cycle, et l'arête candidate n'est pas ajoutée

Lors de l'implémentation des algorithme de Kruskal nous avons buté sur une erreur de test de la boucle principal des méthodes. Le nombres d'arêtes +1 doit être différents du nombre de sommet et non le nombre d'arêtes -1.

Ultramétrique:

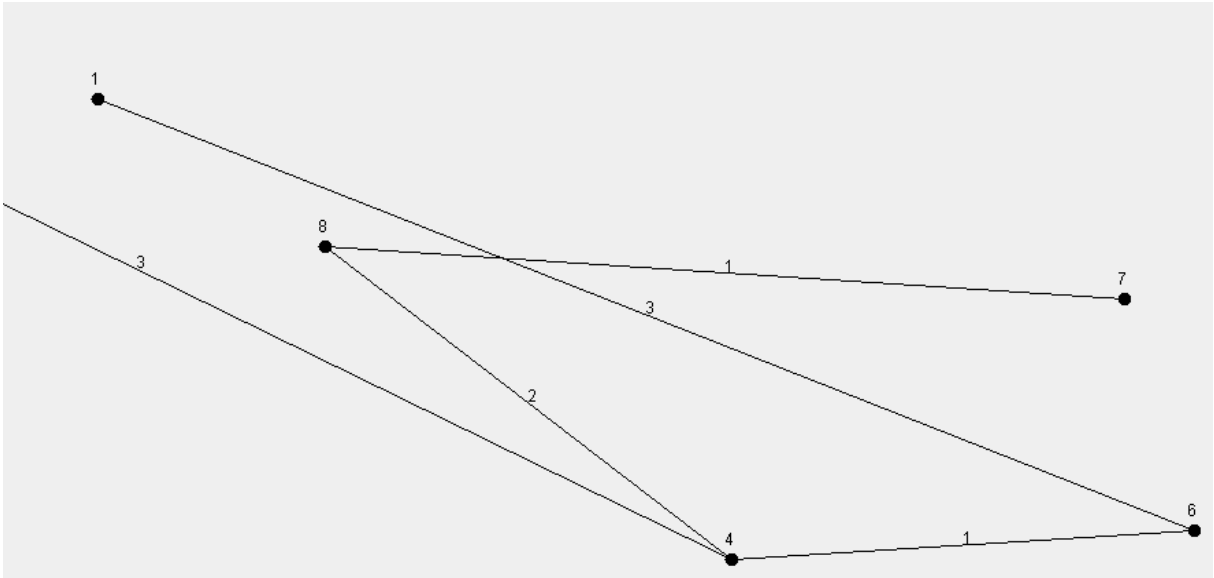
L'ultramétrique étant l'arête de poids la plus grande dans la liste des arêtes qui vont d'un sommet A à un sommet B dans l'ACM, nous avons réutilisé la fonction qui permet de parcourir un graphe, crée pour vérifier si il y avait un cycle dans Kruskal.

Cette fonction recursive nous permet d'avoir la liste des arêtes permettant d'aller du sommet A au sommet B dans l'ACM (dans la pratique, les arêtes sont ajoutées pour aller du sommet B au sommet A).

On prend ensuite le poids maximum de cette liste d'arête pour avoir l'ultramétrique.

## Graphe:

Nous avons utilisé la bibliothèque Swing de Java vue en jours de Programmation Conception Orientée Objet pour afficher les graphs réalisés.
On utilise les fonctions fillOval et drawString pour dessiner les SommetS.
Pour dessiner les arêtes, on utilise drawLine et drawString.




Dans cet arbre, l'ultramétrique entre les points 1 et 8 est de 3.
Pour aller de 1 à 8, il faut utiliser les arêtes (1,6) (6,4) et (4,8) qui ont pour poids respectivement 3, 1 et 2. L'ultramétrique est donc donneé par (1,6). C'est ce que le programme nous répond :
Points choisis : 1 et 8
Ultramétrique : 3

## Tests de rapidité sur Prim et Kruskal

Nous avons testé les différents algorithmes de constructions d'acm sur 1000 nuages de points de 200 points, cela nous a permis de consulter la différence de temps d'exécution entre les algorithmes.

Nous avons effectué ce test sur deux configurations différentes :

*Configuration 1 (ancienne configuration : environ 5 ans) :*
Pour Prism: Non optimisé : 73188 ms / Optimisé : 31256 ms
Pour Kruskal: Non optimisé : 75523 ms / Optimisé : 12683 ms

*Configuration 2 (configuration récente : quelques mois) :*
Prim : Non optimisé : 22311 ms / Optimisé : 10846 ms
Kruskal : Non optimisé : 23637 ms / Optimisé : 10411 ms

On remarque une différence significative entre les algorithmes optimisés et non optimisés.
On voit également que la configuration ancienne est largement plus efficace sur la version optimisée de Kruskal, se rapprochant sensiblement de la configuration récente.

Nous n'avons pas poussé les tests plus loin pour voir à partir de combien de points quand tel ou tel algorihtme est plus performant. En effet le devoir précédent nous avait montré qu'en terme de comparaison de temps d'exécution et d'efficacité précise d'algorithmes, Java était un langage peu adapté.

## Remarque sur la valeur de l'ultramétrique

Notre nuage de point s'étend sur 1400 x 700 pixels et les points sont séparés d'un cercle de rayon de 50 pixels.
Nous avons remarqué que plus nous mettions de points, plus l'ultramétrique a tendance à se rapprocher de 50. Cela se comprend car plus il y a de points, plus les points seront rapprochés et donc il y aura de plus en plus de chance de trouver des points éloignés d'un peu plus de 50 pixels.