Université de Strasbourg
UFR Mathématique et Informatique

L2 Informatique - S4 Printemps
2012/2013

Structures de données, algorithmes 2

Projet : Arbres couvrant de poids minimal

Introduction

Soit $G = (S, A, w)$ un graphe non-orienté et connexe, où S représente l'ensemble des sommets et A l'ensemble des arêtes. Le graphe G est de plus pondéré, c'est à dire que pour chaque arête $a \in A$, un poids (une valeur réelle) $W(a)$ y est attribuée avec la fonction $w : A \longrightarrow \mathbb{R}$.

On appel un arbre couvrant minimal de G tout arbre $G' = (S', A')$ tel que :

- $S' = S$; tous les sommets de G appartiennent à G'.
- $A' \subseteq A$ où chaque sommet de S' est touché par une arête de A' (on dit que A' couvre S),
- (S', A') est connexe et sans cycle (c'est à dire un arbre),
- Le poids total de l'arbre couvrant $P = \sum_{a \in A'} W(a)$ est minimal.

Il existe deux principaux algorithmes pour construire un tel arbre couvrant de poids minimal, dus respectivement à `Kruskal` et `Prim`. Assez proches dans leurs principes et leur performances, ils reposent tous les deux sur la sélection successives des arêtes qui composent l'arbre. Nous allons nous intéresser à l'algorithme de `Prim`.

Description de l'algorithme de Prim

L'algorithme de `Prim` part d'un sommet quelconque puis sélectionne successivement les arêtes de poids minimal qui composeront l'arbre couvrant, comme suit :

1. On choisit un sommet de départ d.
2. On choisit l'arête de poids minimal parmi celles incidentes à d, et on l'ajoute à l'arbre en construction (ainsi que l'autre sommet de l'arête). Si plusieurs arêtes ont le même poids minimal, on choisit n'importe laquelle.
3. On itère en choisissant à chaque fois l'arête de poids minimal parmi celles qui sont incidentes à l'arbre, et ne créent pas de cycle (donc ne sont incidentes qu'à un seul sommet de l'arbre).

La figure 1 présente un graphe pondéré, ainsi que toute les étapes de construction de l'arbre couvrant de poids minimal en partant du sommet 3.

Remarque : Remarquons qu'il y a deux conditions d'arrêt équivalentes : on a marqué le bon nombre d'arêtes d'un arbre (|S|-1), ou bien il n'y a plus d'arêtes incidentes à un sommet de l'arbre. Pour implémenter cet algorithme, on peut utiliser un marquage simple des sommets : faire partie de l'arbre déjà construit (Vu) ou pas (NonVu).

Pour trouver rapidement la prochaine arête à considérer, on utilise une file de priorité (donc triée) dans laquelle on range les arêtes incidentes aux sommets de la partie de l'arbre déjà construite.

Choix d'implémentation

Le programme doit comporter au moins les fonctions suivantes :

- **Charger un graphe :** elle prend en argument un nom de fichier contenant un graphe sous la représentation suivante : chaque ligne contient les informations d'une arête de la forme
  `Sommet de départ` `Poids de l'arête` `Sommet d'arrivé`
- **Sauvegarder un graphe :** elle sauvegarde le graphe manipulé dans un fichier texte ayant le format précédent.

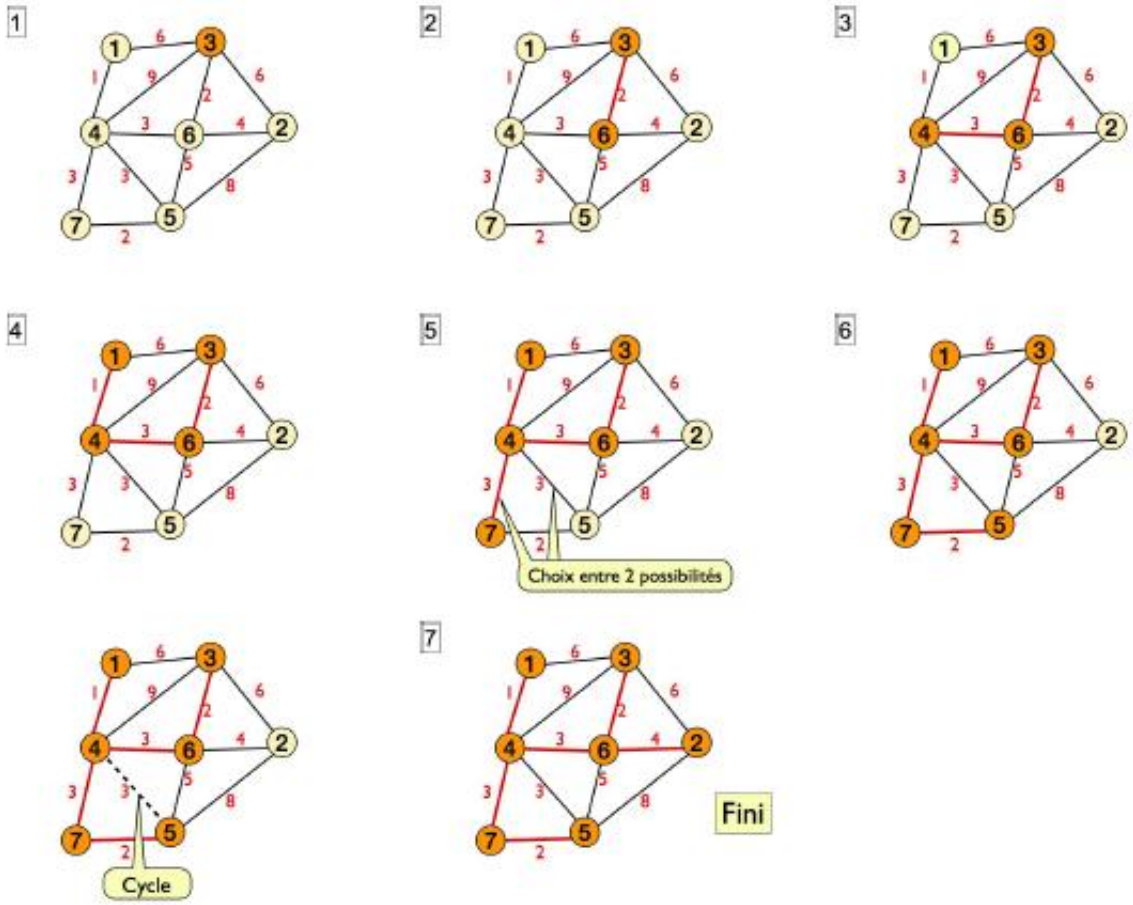

FIGURE 1 – Exemple de construction d'arbre couvrant de poids minimal avec l'algorithme de Prim

- **Modifier une arête :** elle permet de modifier le poids d'une arête du graphe.
- **Ajouter une arête :** elle permet d'ajouter une arête dans le graphe. Pendant l'ajout d'une arête, un nouveau sommet peut aussi être introduit, mais pas deux sommets sinon le graphe devient non-connexe.
- **Supprimer une arête :** elle permet de supprimer une arête du graphe seulement si le graphe restera connexe.
- **Supprimer un sommet :** elle permet de supprimer un sommet en supprimant aussi toutes les arêtes incidentes à ce sommet. Cette action risque de rendre le graphe non-connexe. Assurez-vous de supprimer le sommet seulement si le graphe reste connexe.
- **Calculer l'arbre :** elle permet de calculer l'arbre couvrant minimal avec l'algorithme de `Prim` en partant d'un sommet de départ passé en argument.
- **Afficher l'arbre :** elle permet de générer en sortie au format `pdf` le graphe manipulé ainsi que l'arbre couvrant généré. Pour cette fonction, utilisez l'outil `Latex` ainsi que le package `Tikz`.

D'autres fonctions peuvent être mises en place aussi telle que l'affichage des arêtes incidentes à un ou plusieurs sommets.

Spécification

Chaque sommet est identifié par une chaînes de caractères, alors que le poids des arêtes est une valeur entière signée.

Vous devez spécifier algébriquement toutes les structures de données utilisées pour stocker le graphe et l'arbre manipulés : c'est à dire, la sorte sommet, arête, graphe, ainsi que les opération de base sur ces structures. Les principales fonctions du programme doivent utiliser le maximum des fonctions de base sur les structures que vous définissez. Elle doivent également être spécifier clairement dans le rapport.

Puisque le graphe et l'arbre manipulés sont de taille quelconque, préférez des structures de données dynamiques pour les stocker.

Organisation

- Vous vous organiserez par groupe de deux.
- Votre spécification, conception, les choix d'implémentation et les tests s'y rapportant feront l'objet d'un rapport ($\sim$ 7-10 pages). Le rapport doit comporter au moins les sections suivantes : Introduction, Spécification, Discussion de l'implémentation, Tests et Conclusion. Un organigramme d'appels de fonctions doit être précisé dans la section Discussion d'implémentation.
- Votre code (bien évidemment correctement commenté et documenté avec doxygen) et votre rapport devront être envoyés par courriel à l'adresse **mekhaldi@unistra.fr** sous la forme d'une archive dont le nom respectera ce format **nom1_nom2-ProjetSDA2P.tar.gz**. L'objet du mail doit commencer par **[S4PSDA2]**.
- Date limite de retour : 01/05/2013.