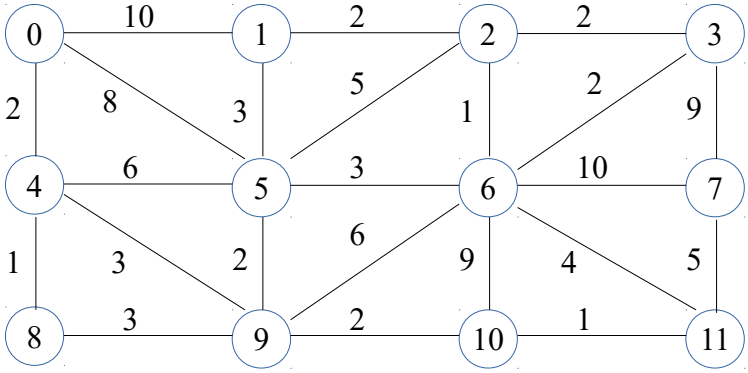
Université de Reims Champagne-Ardenne
Département de Mathématiques et Informatique

Pierre Delisle
Info0501
2018/2019

Travaux Pratiques 6
Graphes
Arbres couvrants de poids minimal

PARTIE A – IMPLÉMENTATIONS DE BASE PAR TABLEAU

L'objectif de ce TP est de programmer les principaux algorithmes permettant de déterminer un arbre couvrant de poids minimal dans un graphe. Pour ce faire, nous utiliserons ce graphe d'exemple :



Vous pouvez télécharger le fichier de données correspondant à ce graphe (**graphe2.txt**) sur le site Web du cours (http://cosy.univ-reims.fr/~pdelisle/enseignement.php). Ce fichier a la même structure que celui du TP précédent (graphe1.txt), à la différence qu'une troisième valeur a été ajoutée à la définition de chaque arête : son poids.

1) Modifiez vos codes d'importation de graphes du TP précédent pour intégrer les poids des arêtes dans vos listes d'adjacences et votre matrice d'adjacences.

Les deux algorithmes ci-dessous construisent un arbre couvrant de poids minimal en ajoutant progressivement des arêtes à un ensemble où chaque arête est représentée par son sommet d'origine et son sommet d'extrémité.

2) Définissez une/des structure(s) de données appropriée(s) pour représenter les arêtes et l'ensemble d'arêtes définissant l'arbre couvrant de poids minimal.

Algorithme de Kruskal

L'algorithme de Kruskal permet de déterminer un arbre couvrant de poids minimal d'un graphe en gérant l'évolution des composantes connexes des sommets durant la construction de l'arbre. Une implémentation de l'algorithme de Kruskal devrait comporter les principales étapes suivantes :

- construction d'un tableau de toutes les arêtes du graphe à partir des listes d'adjacence ou de la matrice d'adjacences ;
- tri du tableau d'arêtes construit ;
- création d'une structure de données permettant de gérer les composantes connexes ;

- construction de l'arbre couvrant de poids minimal (ensemble des arêtes retenues) en choisissant les arêtes par ordre croissant de longueur selon l'appartenance ou non de ses sommets à la même composante connexe.

3) Écrivez une fonction **genererAcpmKruskal** permettant de générer un arbre couvrant de poids minimal pour le graphe d'exemple par la méthode de Kruskal implémentée en gérant les composantes connexes avec un tableau, puis une fonction **afficherAcpm** permettant d'afficher l'ensemble d'arêtes résultant. Testez vos fonctions dans le main et vérifiez que vous obtenez bien un arbre couvrant de poids minimal de poids 24.

Algorithme de Prim

L'algorithme de Prim permet de déterminer un arbre couvrant de poids minimal d'un graphe en démarrant d'un sommet arbitraire et en étendant un arbre jusqu'à couvrir tous les sommes du graphe. Pour implémenter l'algorithme de Prim, il faut gérer l'ensemble des sommets qui n'appartiennent pas à l'arbre en cours de construction à l'aide d'une structure de données permettant la sélection de l'arête de poids minimal reliant chacun de ces sommets à l'arbre.

4) Écrivez une fonction **genererAcpmPrim** permettant de générer un arbre couvrant de poids minimal pour le graphe d'exemple par la méthode de Prim implémentée par tableau, puis utilisez la fonction **afficherAcpm** développée précédemment pour afficher le résultat. Testez vos fonctions dans le main et vérifiez que vous obtenez bien un arbre couvrant de poids minimal de poids 24.

PARTIE B – IMPLÉMENTATIONS AVANCÉES

Pour implémenter efficacement l'algorithme de Kruskal, on peut utiliser une structure d'ensembles disjoints pour gérer l'évolution des composantes connexes du graphe.

5) Définissez les fichiers **ensemble.c** et **ensemble.h** permettant d'implémenter une structure d'ensembles disjoints par listes chaînées en implémentant les opérations suivantes :

- **créerEnsemble(x)** : créée un ensemble à partir d'un sommet *x* du graphe ;
- **trouverEnsemble(x)** : détermine l'ensemble auquel appartient le sommet *x* ;
- **union(x,y)** : fusionne les ensembles dont font partie les sommets *x* et *y*.

Modifiez ensuite votre fonction **genererAcpmKruskal** pour qu'elle utilise cette structure plutôt qu'un tableau pour gérer l'évolution des composantes connexes.

Pour implémenter efficacement l'algorithme de Prim, on peut faciliter la sélection de la nouvelle arête à ajouter à l'arbre en cours de construction. Pour ce faire, il faut gérer l'ensemble des sommets qui n'appartiennent pas à l'arbre à l'aide d'une file de priorités min basée sur le poids minimal de l'arête reliant chaque sommet à l'arbre.

6) Définissez les fichiers **filePrioritesMin.c** et **filePrioritesMin.h** permettant d'implémenter une file de priorités min pour l'algorithme de Prim. Une bonne idée est d'adapter la file de priorités max définie au TP 1.

Modifiez ensuite votre fonction **genererAcpmPrim** pour qu'elle utilise cette structure plutôt qu'un tableau pour gérer les sommets qui n'appartiennent pas à l'arbre en cours de construction.