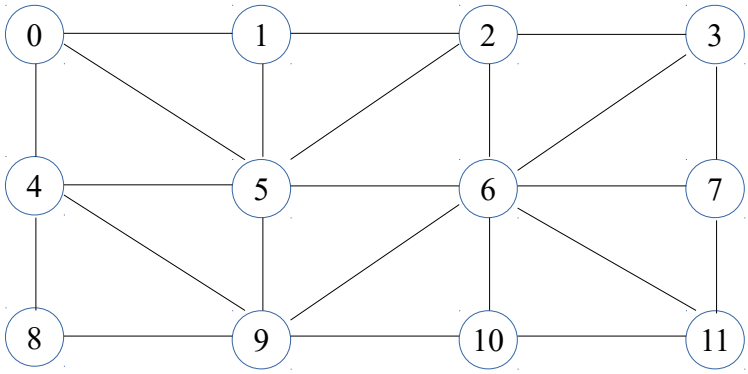
Université de Reims Champagne-Ardenne
Département de Mathématiques et Informatique

Pierre Delisle
Info0501
2018/2019

Travaux Pratiques 5
Graphes
Représentation et parcours

L'objectif de ce TP est de programmer les principaux algorithmes de parcours de graphes. Pour ce faire, nous utiliserons le graphe suivant comme exemple :



Vous pouvez télécharger le fichier de données correspondant à ce graphe (**graphe1.txt**) sur le site Web du cours (http://cosy.univ-reims.fr/~pdelisle/enseignement.php).

Sur la première ligne du fichier, on retrouve le nombre de sommets du graphe (dans la suite, les n sommets sont identifiés par un entier entre 0 et $n - 1$). Sur les 3 lignes suivantes, on retrouve les caractéristiques du graphe sous forme de booléens. Les valeurs indiquent donc que ce graphe est non orienté, non valué et non complet. Dans la section délimitée par « debutDefAretes » et « finDefAretes », chaque arête est représentée par 2 nombres : le sommet d'origine et le sommet d'extrémité. La première étape est de représenter ce graphe en mémoire.

Représentation mémoire

On peut représenter ce graphe sous forme de listes d'adjacences et de matrice d'adjacences.

1) Créez les fichiers **graphe.c** et **graphe.h** et implémentez-y une structure de graphe ainsi que les procédures suivantes :

- **creerListesAdjacences** : lit les données du fichier **graphe1.txt** et construit le tableau de listes d'adjacences de ce graphe (une bonne idée est de réutiliser votre implémentation de tableau de listes chaînées du Tp3).
- **afficherListesAdjacences** : affiche les listes d'adjacences d'un graphe ;
- **creerMatriceAdjacences** : lit les données du fichier **graphe1.txt** et construit la matrice d'adjacences de ce graphe ;
- **afficherMatriceAdjacences** : affiche la matrice d'adjacences d'un graphe.
- **creerGraphe** : crée un graphe en allouant la mémoire nécessaire et en utilisant les fonctions définies précédemment pour associer les représentations mémoire à la structure de graphe ;
- **détruireGraphe** : détruit un graphe en libérant ses ressources mémoire.

Avant d'aller plus loin, n'oubliez pas d'écrire un main pour tester vos fonctions !

Parcours en largeur

Afin de réaliser le parcours en largeur, il faut tout d'abord créer une structure permettant d'enregistrer le parcours.

2) Créez un tableau de sommets permettant, pour chaque sommet du graphe, de conserver sa couleur, sa distance du sommet d'origine du parcours et son père dans l'arbre de parcours.

Il faut ensuite créer une structure de file afin de gérer l'ensemble des nœuds durant le parcours.

3) Créez les fichiers **file.c** et **file.h** et implémentez-y une structure de file de sommets par tableau ainsi que les procédures de file suivantes :

- **créerFile** : crée une file vide de capacité maximale fixe en allouant la mémoire nécessaire ;
- **détruireFile** : détruit une file en libérant ses ressources mémoire ;
- **fileVide** : retourne vrai si la file est vide, faux sinon ;
- **enfiler** : ajoute un élément en queue de file ;
- **défiler** : enlève l'élément en tête de file s'il en existe un et retourne sa valeur.

Avant d'aller plus loin, n'oubliez pas d'écrire un main pour tester votre file !

4) Définissez la fonction **parcoursLargeur** qui construit l'arbre de parcours en largeur à partir d'un sommet spécifique.

5) Définissez la fonction **afficherChemin** qui, à partir d'une arborescence de parcours en largeur, donne le chemin le plus court entre l'origine du parcours et un sommet du graphe. Affichez ensuite le plus court chemin entre l'origine et chaque sommet.

Parcours en profondeur

6) Définissez la fonction **parcoursProfondeurRecursif** qui construit une forêt de parcours en profondeur tout en calculant les dates de découverte et de fin de traitement de chaque sommet.

7) Définissez la procédure **afficherParcoursProfondeur** qui, à partir d'une arborescence de parcours en profondeur, affiche les dates de découverte et de fin de traitement de chaque sommet.

<u>Questions supplémentaires pour les rapides et les motivés</u>

- Définissez la fonction **parcoursProfondeurIteratif** qui effectue le parcours en profondeur d'un graphe sans s'appeler récursivement ;
- Récrivez les fonctions **parcoursLargeur** et **parcoursProfondeur** de sorte à utiliser la matrice d'adjacences au lieu des listes d'adjacences ;
- Définissez la fonction **triTopologique** qui effectue le tri topologique d'un graphe orienté acyclique.