C.U.F.R. J-F. Champollion — L2 Informatique S5
Albi — 2015

Théorie et Algorithmique des Graphes

TP2 (Deux séances)
Parcours en largeur et profondeur

Le but de ce TP est de mettre en œuvre les méthodes de parcours en largeur et en profondeur d'un graphe. Ce TP sera complété par le TP3 portant sur les applications de ces parcours.
Rappelons que ces deux algorithmes s'appliquent indifféremment à un graphe orienté ou non orienté et devront donc être testés dans les deux cas.

On pourra tester les parcours sur les quatre graphes suivants (dont les illustrations sont fournies sur Moodle dans les archives *largeur.zip* et *profondeur.zip :*

3 graphes orientés :
G1={{5},{1,4},{2},{3},{2,4}}
G2={{5},{1,4,5},{2,4},{},{4}}
G3={{3,5,6},{1},{2,4},{},{},{4}}

et le graphe de Petersen :
Petersen={{2,5,6},{1,3,7},{2,4,8},{3,5,9},{1,4,10},{1,8,9},{2,9,10},{3,6,10},{4,6,7},{5,7,8}}

Vous trouverez enfin dans l'archive *TP2Parcours.zip* un corrigé du TP1 et les entêtes des méthodes à écrire lors du TP2.

1 Parcours en largeur

- On utilise un vecteur *Visite* tel que Visite[i]=0 si le sommet i n'a pas encore été visité et Visite[i]=1 sinon.
- On utilise une liste *File* pour gérer la file d'attente des sommets à visiter prochainement.
- Le résultat de l'appel de la méthode sera la liste des sommets visités dans l'ordre de ce parcours représenté par une liste *ordreVisite*.

Compléter les méthodes suivantes :

- *public Liste largeur(int i)* effectuant le parcours en largeur du graphe à partir du sommet i.
- *public Liste[] largeurG()* effectuant le parcours en largeur généralisé à tout le graphe.

2 Parcours en profondeur

- On utilise toujours le vecteur *Visite* défini dans le parcours en largeur.
- Le résultat de l'appel de la fonction sera la liste des sommets visités dans l'ordre de ce parcours représenté par une liste *ordreVisite*. On fonction de l'endroit où on positionne la mise à jour de la liste *ordreVisite* la fonction donnera l'ordre de première visite ou l'ordre de dernière visite. On pourra passer d'une version à l'autre en commentant une des lignes de mise à jour ou en ajoutant un paramètre à la fonction.

Comme souvent lorsqu'on écrit un algorithme récursif, on distinguera une fonction auxiliaire (récursive) et une fonction d'appel dont le rôle est d'initialiser les variables et d'enclencher le premier appel récursif.

Compléter les méthodes suivantes :

- *public void profRec(int i)* méthode auxiliaire récursive qui provoque un parcours en profondeur du graphe à partir du sommet i. Cette fonction ne retourne aucun résultat et se contente de mettre à jour les attributs *Visite* et *ordreVisite.*
- *public Liste profond(int i)* effectuant le parcours en profondeur du graphe à partir du sommet i.
- *public Liste[] profondG()* effectuant le parcours en profondeur généralisé à tout le graphe.