Institut National
Universitaire
Champollion

TP 3 - Applications des parcours

Théorie et Algorithmique des Graphes
L3 INFO - Semestre 6

Le but de ce TP est de mettre en œuvre les principales applications des parcours en largeur ou profondeur dans le traitement des graphes orientés ou non orientés.

1. Graphes non orientés

Le parcours d'un graphe non orienté permet, à l'aide de modifications mineures, de fournir de nombreux renseignements sur le graphe.

1. **Tester la connexité d'un graphe à l'aide d'un parcours en profondeur**
   Un graphe non orienté est connexe si et seulement si le parcours en profondeur à partir du sommet 1 visite tous les sommets.

   $\rightsquigarrow$ Écrire une fonction booléenne `isConnexe(G)` déterminant si un graphe G est connexe.

2. **Tester l'existence de cycles à l'aide d'un parcours en profondeur généralisé**
   Un graphe non orienté G ne contient pas de cycle si et seulement si le parcours (généralisé à tout le graphe car un cycle peut apparaître dans n'importe quelle composante connexe) ne provoque pas de revisite (autre que celle du père). On doit donc introduire un deuxième paramètre à la fonction pour se rappeler du numéro du père. On utilise également une variable booléenne `cycle` qui sera vrai si et seulement si on a détecté un cycle.

   $\rightsquigarrow$ Écrire une fonction `cyclicRec(G,i,pere,Visite,cycle)` : qui effectue un parcours en profondeur du sommet i (dont le père est `pere`) en mettant à jour les variables `Visite` et `cycle`. Dès qu'un cycle est détecté, il faut arrêter tous les parcours en cours.

   $\rightsquigarrow$ Écrire une fonction `isCyclic(G)` qui initialise les variables et gère le parcours en profondeur généralisé à tout le graphe.

3. `Test d'arbre`
   Un graphe non orienté est un arbre si et seulement si il est connexe et sans cycle.

   $\rightsquigarrow$ Écrire une fonction booléenne `isArbre(G)` qui renvoie `true` si et seulement si G est un arbre.

4. **Recherche des plus courts chemins à l'aide d'un parcours en largeur**
   Comme vu en cours, on utilise un vecteur `Dist` tel que `Dist[y]` est la longueur du plus court chemin de x vers y (ou la valeur -1 si y n'est pas accessible à partir de x). Afin de retrouver le chemin, on utilise également un tableau des pères tel que `Pere[y]` est le prédécesseur de y dans le plus court chemin de x vers y.

   $\rightsquigarrow$ Écrire une fonction `plusCourtChemin(G,i)` qui retourne les vecteurs `Dist` et `Pere` donnant les plus courts chemins issus du sommet i.

5. **Test de bipartisme à l'aide d'un parcours en largeur généralisé**
   On a vu en cours qu'un graphe est biparti si et seulement si il n'admet pas de cycle de longueur impaire. L'algorithme de recherche de cycle peut donc être adapté en test de bipartisme. Il suffit pour cela de numéroter alternativement à l'aide de 1 et 2 les sommets visités. Un cycle de longueur impaire est détecté lorsque x revisite un sommet y de même numéro que lui.
   On utilise pour cela le vecteur de visite qui vaudra 0 pour un sommet non visité, et alternativement 1 ou 2 pour les sommets visités.
   L'ensemble X est alors l'ensemble des sommets à 1 et Y l'ensemble des sommets à 2.

   ⇝ Écrire une fonction booléenne `biparti(G)` qui renvoie `true` si et seulement si le graphe est biparti et les ensembles X et Y (sous forme de listes).

2. Graphes orientés

Si la recherche des plus courts chemins ne diffère pas du cas non orienté, en revanche la recherche de cycle est très différente.
En effet dans le cas d'un graphe orienté, les cycles sont caractérisés par les arcs arrières mais il ne faut pas tenir compte des revisites par arcs transverses ou en avant. Un arc arrière est caractérisé par le fait qu'on revisite un sommet dont le parcours en profondeur n'est pas encore terminé.
On utilise pour ça le vecteur de visite qui vaut 0 avant la visite, 1 quand la visite commence et 2 lorsque le parcours est terminé.

1. **Tester l'existence de cycles à l'aide d'un parcours en profondeur**
   ⇝ Écrire une fonction `cyclicRec(G,i,Visite,cycle)` qui effectue un parcours en profondeur du sommet i en mettant à jour les variables `Visite` et `cycle`. Dès qu'un cycle est détecté, il faut arrêter tous les parcours en cours.

   ⇝ Écrire une fonction `isCyclic(G)` qui initialise les variables et gère le parcours en profondeur généralisé à tout le graphe.