Algorithmique Avancé
Compte rendu TP3

Nombre_arcs et Nombre_sommets :

Ces fonctions comptent le nombre d'arcs et de sommets d'un graphe. Elles parcourent la liste chaînée donnée et comptent. Dans Nombre_sommets on parcourt juste grâce à un while sur les sommets.

Dans Nombre_arcs on fait un double while : un sur les sommets et puis un pour chaque arc de chaque sommet.

Fonctions de calcul de degré :

Les premières fonctions degre_entrant_noeud et degre_sortant_noeud utilisent un algorithme de double while. Dans degre_sortant_noeud on part du noeud et on compte tous les arcs de ce noeud. On vérifie quand même que le noeud appartient au graphe.

Dans degre_entrant_noeud on parcourt tous les noeuds du graphe et on regarde pour chacun de ces noeuds si il y a arcs qui va vers le noeud recherché.

Ensuite on se sert de ces 2 fonctions pour calculer les degrés maximal et minimal d'un graphe. Les fonctions suivent le même algorithme. On compte le degré sortant pour chaque noeud et on prend le minimum/maximum.

Indépendant :

On regarde chaque noeud du graphe avec un while. Pour chacun de ces noeuds, on compte les degrés entrant et sortant de chaque noeud. Si la somme est supérieure à 3, alors le graphe n'est pas indépendant. Sinon, il faut vérifier dans quel cas on se trouve. Le but est de n'avoir que des groupes de 2 noeuds maximum.

Complet :

On regarde pour chaque noeud du graphe si il est relié par des arcs à tous les autres. Pour ce faire on prend chaque noeud et on compte le nombre d'arcs, et si il est plus petit que le nombre de noeuds du graphe moins 1, alors le graphe n'est pas complet.

Régulier :

On stocke la valeur du degré du premier noeud. On regarde pour chaque noeud du graphe si leur degré est le même que celui trouvé au début. Si a un moment ce n'est pas le cas, on renvoie 0 (faux), sinon le graphe est régulier.

Afficher_graphe_profondeur :

Pour cette fonction, on a modifié la structure du noeud du graphe en ajoutant un entier "visite". On veut parcourir juste une fois le graphe, donc quand on sait que l'on est déjà passé par cette noeud (visite==1). C'est une fonction récursive. A chaque fois, on commence par afficher le label du noeud racine et et ceux des noeuds qui sont liés à ce

noeud par ses arcs. On fait ça récursivement jusqu'à qu'on retombe sur le noeud qui est déjà visité (visite==1).

Colorier_graphe :

Dans cette fonction, on ajoute dans la structure du noeud une variable couleur qui indique la couleur du noeud (un entier). Au début, on met tous les noeud à la même couleur (la 1er couleur donnée dans le tableau de couleurs donnée en paramètre). On parcourt chaque noeud et ses arcs. Si un noeud lié par un arc a la même couleur que le noeud actuel, on modifie la couleur du noeud actuel en mettant une couleur à l'indice suivante dans le tableau des couleurs.

Afficher_graphe_largeur :

Dans cette fonction, on utilise la variable "visite" ainsi qu'une file pour stocker les noeuds. On parcourt chaque noeud. Au début, on dépose un noeud dans la file. Tant que la file n'est pas vide, on retire un noeud dans la file, si ce noeud n'est pas visité, on affiche son label et met sont "visite" à 1. Puis on parcours les arcs de ce noeud en ajoutant dans la file des noeuds non visités liés à ces arcs.

Plus_court_chemin :

On utilise l'algorithme de Dijkstra pour calculer le plus court chemin. On rajoute un champ poid dans les noeuds pour avoir la distance de chaque noeud de la source. On initialise chacun de ces noeuds à 99 sauf l'origine à 0. Une fois que c'est fait on trouve un noeud de distance minimum avec l'origine, que l'on supprime du graphe (on passe le champ visite à 1, et du coup on ne le comtpe pas quand on cherche le minimum). On mets à jour les distances de tous les noeuds voisin à celui que l'on vient d'enlever, et on répète jusqu'à qu'il n'y est plus de noeuds dans le graphe.

Une fois qu'on a les chemins, on part de la destination et on remonte le chemin jusqu'à la source grâce au champ précédent chemin des noeuds. On stocke ce chemin dans la variable chemin (en l'inversant pour qu'il soit dans le bon sens), et on stock le nombre de noeud dans la variable nb_noeuds.

Élémentaire et Simple :

On se sert du champ visite dans les noeuds. On l'initialise à 0 pour chaque noeud, puis on regarde chaque noeud du chemin. Si pour un noeud du chemin donné, le champ visite est à 1, on renvoie 0 car le chemin n'est pas élémentaire. Sinon, on passe le champ visite à 1 et on continue de visiter le chemin.

L'algorithme de simple est le même, mais pour les arcs.

Hamiltonien et Eulérien :

On sert encore une fois du champ visite des noeuds, que l'on initialise à 0. Ensuite, on parcourt tout le chemin comme dans élémentaire, et on mets à 1 chaque noeud que l'on visite. Une fois que c'est fait, on refait un while sur tous les noeuds du graphe. Si un noeud n'est pas à 1 alors le chemin n'est pas hamiltonien et on renvoie 0.

L'algorithme de Eulérien est le même, mais pour les arcs.

Graphe_hamiltonien :

Pour la fonction graphe_hamiltonien, on utilise le théorème de Alain Ghouila-Houri qui dit qu'un graphe orienté simple fortement connexe à n sommets dont chaque sommet est au moins de degré entrant n/2 et de degré sortant n/2 est hamiltonien. On regarde donc pour chaque noeud du graphe le degré entrant et sortant. Si pour un des noeuds la propriété n'est pas vérifié alors on renvoie 0.

Graphe_eulerien :

On utilise cette fois ci le théorème d'Euler. Le théorème pour les graphes orientés est "G est fortement connexe et chacun de ses sommets est l'extrémité initiale et terminale du même nombre d'arêtes". On vérifie donc d'abord que G est bien fortement connexe, puis pour chaque noeud du graphe on regarde si le degré entrant et sortant est le même. Si on a visité tous les noeuds et que l'on ne trouve pas de problèmes, alors le graphe est eulérien.

Distance :

La fonction distance fonctionne de la même manière que la fonction plus_court_chemin : on utilise l'algorithme de Dijkstra. La différence est qu'après l'algorithme, on renvoie le distance contenu dans le champ poid de la variable y, vu qu'on a pris x en tant qu'origine.

Excentricité et Diamètre :

Pour l'excentricité, on calcul pour tous les noeuds du graphe leur distance avec le noeud n. On renvoie la valeur de distance maximale.

Pour le diamètre, on calcul l'excentricité pour chaque noeud du graphe. On renvoie la valeur maximale.