INFO0027 : Techniques de programmation

Troisième projet : « Arbre couvrant minimal dans un graphe »

Lundi 2 mai 2011

Contenu du document (7 pages) :

1. Énoncé du troisième projet ;
2. Annexe A : Algorithme de Kruskal.
3. Annexe B : Prototypes des fonctions à implémenter.

1 Énoncé du Projet 3

Objectifs :
– Implémenter des algorithmes de graphes ;
– Rédiger un rapport de qualité.

Votre tâche :

1. Écrire (en langage C) un fonction `genererGraphe(g, nbS, nbA, c)` générant de manière aléatoire un graphe à `m` sommets et `n` arcs, connexe si la valeur de `c` est différente de zéro [1] ;
2. Écrire (en langage C) un fonction `estAcyclique(g)` permettant de tester si un graphe est acyclique ou non.
3. En déduire une implémentation de l'algorithme de Kruskal (cf. A.2) ;
4. Rédiger un rapport comprenant une présentation du fonctionnement des fonctions implémentées ainsi qu'une brève discussion de la complexité (opérations coûteuses, améliorations envisageables, etc.).

NB : Les prototypes des fonctions à implémenter est fourni en Annexe B.

Évaluation :

Le travail se verra attribuer la mention

TB = "Très Bon", B = "Bon", I = "Insuffisant" ou TI = "Très Insuffisant"

sur base des critères suivants :
– clarté et pertinence du rapport ;
– respect du délai et de l'ensemble des consignes.

Remise du rapport :

Lundi 30 mai 2011 (au début de l'examen)

Remarque importante :

En plus du rapport, les implémentations réalisées (codes de test compris) sont à envoyer, pour le lundi 30 mai à 23h59, à l'adresse `Thomas.Leuther@ulg.ac.be` sous forme d'une archive `NOM-prénom-projet3.zip`.

1. Si `c` est nul, il n'y a aucune contraite sur la connexité du graphe à générer.

A Algorithme de Kruskal

A.1 Recherche de l'arbre couvrant de poids minimal

On considère un graphe simple [2] non orienté et pondéré

$$G = (S, A, w : A \to \mathbb{R}).$$

Un *arbre couvrant minimal* de G est un sous-graphe $G' = (S, A', w|_{A'})$ de G tel que :
- chaque sommet $s \in S$ de G est atteint par au moins un arc de A' (i.e. A' recouvre l'ensemble S des sommets de G) ;
- le graphe (S, A') est connexe et sans cycle (i.e. c'est un arbre) ;
- le poids total de l'arbre $\sum_{a \in A'} w(a)$ est minimal.

A.2 Algorithme (Kruskal)

Donnée d'entrée : graphe non orienté $G = (S, A)$, connexe et pondéré.

1. Initialiser $A' := \{\}$.
2. Énumérer les arcs de G par ordre croissant de poids ;
3. Si l'arc courant ne crée pas de cycle dans (S, A'), l'ajouter à A' ;
4. Itérer jusqu'à avoir un arbre couvrant (i.e. $\#A' = \#S - 1$).

Le graphe $(G, A', w|_{A'})$ est un arbre couvrant de G, de poids minimal.

2. Si deux sommets étaient connectés par plusieurs arcs, il suffirait de conserver l'arête de plus petit poids.

B Prototypes des fonctions à implémenter

B.1 Dans le fichier "graphesAleatoires.h" :

```
/*
- Génère un graphe aléatoire comportant un nombre de sommets et d'arcs fixés,
toujours connexe si la valeur de connexe est non nulle.
- Retourne la valeur -1 si les valeurs des arguments passés à la fonction
sont incompatibles , une valeur nulle sinon.
*/
int genererGraphe(graphe *g,
                  unsigned int nbSommets, unsigned int nbArcs, int connexe);
```

B.2 Dans le fichier "algoKruskal.h" :

```
/*
- Teste la présence de cycles dans le graphe.
- Retourne une valeur nulle si le graphe est acyclique, non nulle sinon.
*/
int estAcyclique(graphe *g);

/*
- Retourne un arbre couvrant de poids minimal (déterminé au moyen de l'agorithme
de Kruskal) pour un graphe connexe passé en argument.
*/
void algoKruskal(graphe *g, graphe *h);
```

C Implémentation fournies de listes d'adjacence

C.1 Dans le fichier "graphes.h" :

```
/*
structure pour les éléments de la liste des adjacents à un sommet du graphe
*/
struct eltadj_t{
int dest;                   // label du sommet adjacent
int poids;                  // poids de l'arc vers le sommet adjacent
struct eltadj_t *suivant;
};

/*
structure pour représenter un sommet du graphe
*/
struct sommet_t{
int label;                // label du sommet
int info;                 // information supplémentaire associée au sommet
struct sommet *suivant;
struct eltadj_t *adj;
};

/*
structure pour représenter un graphe
*/
struct graphe_t{
int nbS;                  // nombre de sommets
int nbA;                  // nombre d'arcs
int maxS;                 // label max des sommets du graphe
struct sommet_t *premierSommet;
struct sommet_t *dernierSommet;
};

typedef struct graphe_t graphe;
typedef struct sommet_t sommet;
typedef struct eltadj_t eltadj;
```

D Prototypes des fonctions fournies

D.1 Dans le fichier "graphes.h" :

```
/*
- Recoit l'adresse d'une variable de type graphe.
- La variable correspondante sera initialisée au graphe vide.
*/
void initialiserGraphe(graphe *);

/*
- Recoit l'adresse d'une variable de type graphe et un entier.
- Le champ maxS est incrémenté d'une unité, un sommet de label maxS (après incrémenta
est ajouté au graphe en fin de liste et ce sommet porte l'information "info" fournie
- Retourne le code d'erreur -1 si l'allocation de mémoire n'est pas possible
*/
int ajouterSommet(graphe *, int info);

/*
- Crée un nouvel arc du sommet de label "a" vers le sommet de label "b" avec le poids
- Les arcs sont rangés dans la liste d'adjacence par indice croissant.
- Si un arc entre "a" et "b" existe déjà, le poids est mis à jour sans créer de nouve
- Retourne le code d'erreur -1 si "a" n'existe pas, -2 si "b" n'existe pas, -3 si l'a
*/
int ajouterArc(graphe *, int a, int b, int poids);

/*
- Supprime un sommet et les arcs correspondants (i.e., les arcs ayant leur origine ou
et libère la mémoire correspondante.
- Retourne le code d'erreur -1 si le sommet n'existe pas.
*/
int supprimerSommet(graphe *, int label);

/*
- Supprime l'arc joignant "a" à "b". Cette fonction libère la mémoire correspondante.
- Retourne le code d'erreur -1 si l'arc n'existe pas
*/
int supprimerArc(graphe *, int a, int b);
```

```
/*
- Supprime entièrement un graphe en libérant la mémoire correspondante.
- Réinitialise le graphe au moyen de la fonction "initialiserGraphe".
*/
void supprimerGraphe(graphe *);

/*
- Affiche à l'écran la valeur des champs nbS, nbA et maxS.
- Affiche également l'ensemble des sommets et pour chacun, la liste des arcs vers les
*/
void afficherGraphe(graphe *);
```