Graphe

JOYMANGUL Jensen

Licence Informatique
UFR Sciences et Techniques
Université de Bourgogne

13 mai 2015

Table des matières

1. Introdution

Dans le cadre de ce projet de graph, nous devions implémenter l'algorithme pour calculer l'ensemble stables maximum dans les graphes de disques. Cet algorithme a été découvert par *J.M. ROBSON* en 1986. Cet algorithm est NP-complet et grâce à l'algorithme de Robson, on peut espérer un temps d'exécution de $O(2^{0.276n})$.

Pour l'implémenter j'ai choisis de le faire en C++, car on pourra programmer en orienter objets ce qui est impossible de faire en C. Je n'ai pas pris du JAVA car pour certaines parties surtout la partie pour générer et afficher le graphe j'ai repri le code du TP2 et le code était déjà en C.

2. Représentation des graphes

Il existe deux structures pour représenter des graphes :

1. Matrice d'adjacence
2. Tableau de listes d'incidence

J'ai pris la deuxième option car c'est plus facile d'enlever des points dans le graph mais aussi on peut l'implementer en C++ avec un *vector*(une liste) ce qui rend la taille modulable et rend la gestion de la mémoire plus atomiser.

3. Structure du porgramme

Il y a deux classes : une classe ***Point*** et une classe ***Graphe***.

3.1 La classe Point

Cette classe contient seulement les coordonnées(x et y) d'un point. Il y a aussi une fontion *bool egale(Point* p)* qui test si deux points sont équivalents.

3.2 La classe Graphe

C'est la classe la plus importante du programme. L'algorithme de *l'ensemble stable maximum* est implémenté dans cette classe.

Les attributs

int sommet : Le nombre de sommets dans le graphe

int dmax : La taille de chaque disque

int height : La hauteur du graph

int length : La largeur du graph

vector<vector<int» voisin : La liste de voisin. Pour chaque sommet on stock les indice de ses voisins.

vector<int>* degre : La liste des degrés pour chaque sommet.

vector<Point*> stable : La liste des points stable

Les méthodes

int distanceCarre(Point* p1, Point* p2) : Retour la distance au carré entre deu points.

void initvoisin() : initialise la liste des voisins.

void initdegre() : Initialise la liste de degrés.

int degremax() : Retour l'indice du point avec de degré maximal.

Graph* enlevePoint(int indice) : Retour un nouveau graph avec un point en moin(l'indice de ce point est passé en paramètre).

Graph* enleveVoisin(int indice) : Retour un nouveau graph avec un point et tous ses voisins en moin(l'indice de ce point est passé en paramètre).

void affichegraphe(char* s) : Crée un fichier *postscript* représentant le graphe. Le nim de ce fichier est passé en paramètre.

int ESM(Graph* g,vector<Point*>* s) : Retourne L'enmseble maximum stable

int maxi(int a, int b, int inddegre, vector<Point*>* s, Graph *g) : Retourne la valeur la plus grande entre a et b.

Méthode pour enlever un point et ses voisins dans un graphe

```
1  Graph* Graph::enleveVoisin(int indice)
2  {
3      Graph* g = new Graph();
4      g->liste = this->liste;
5      vector<int> row = this->voisin[indice];
6      g->sommet = this->sommet - row.size()-1;
7      g->dmax = this->dmax;
8      int cmpt=0;
9      for(int i=0;i<row.size();i++)
10     {
11             if(row[i]< indice)
12                 cmpt++;
13             g->liste.erase(g->liste.begin() + (row[i] - (i)));
14     }
15
16     if((indice - cmpt )<=0)
17         g->liste.erase(g->liste.begin()+0);
18     else
19         g->liste.erase(g->liste.begin() + (indice-cmpt));
20
21     g->voisin.resize(g->sommet);
22     g->initvoisin();
23     g->degre = new vector<int>;
24     g->initdegre();
25     return g;
26 }
```

Listing 3.1 – Méthode pour enlever un point et ses voisins

La variable *row* contient la liste des indices des voisins du point que l'on veut supprimer. On parcour cette liste et on test le contenu de cette liste.

A la *ligne 11* si le contenu de la liste à l'indice i est plus petit que l'indice du point que l'on souahaite supprimer alors on incrémente le compteur *cmpt* qui va être utilisé plus bas dans la méthode.Ensuite, on supprime directement le voisin dans la liste des points cependant comme a chaque fois la taille de cette liste de point change il faut faire *row[i] - (i)* pour supprimer le point au bon indice. Ce réajustement peut seulement se faire si la liste *row* est ordonné en ordre ascendant.

A la *ligne 16* on traite le cas où il faute éliminer le premier voisin qui se trouve dans la liste *row*.

Après on réinitialise la liste des voisins et la liste des degrés du nouveau graphe.

4. Résultats

Pour faire des tests j'ai fixé la hauteur et la largeur du graphe. On fait varier le nombre de sommet et la taille des disques et on fait plusieurs test avec ses valeurs puis calculer une moyenne.

4.1 Test avec 5 Sommets

Taille des disques	Test 1	Test 2	Test 3	Test 4	Moyenne
d=30	5	4	5	4	**4,50**
d=40	4	3	5	4	**4,00**
d=50	2	4	3	2	**2,75**
d=60	2	3	3	3	**2,75**

4.2 Test avec 10 Sommets

Taille des disques	Test 1	Test 2	Test 3	Test 4	Moyenne
d=30	5	6	7	6	**6,00**
d=40	5	5	4	4	**4,50**
d=50	4	3	3	3	**3,25**
d=60	4	3	3	4	**3,50**

4.3 Test avec 15 Sommets

Taille des disques	Test 1	Test 2	Test 3	Test 4	Moyenne
d=30	7	9	7	7	**7,50**
d=40	5	5	7	6	**5,75**
d=50	4	4	4	5	**4,25**
d=60	4	3	4	5	**4,00**

5. Limitations de la version actuelle et possibilités d'améliorations

Malheureusement faute de temps je n'ai pas pu implémenter les deux améliorations précisées dans le sujet car j'ai été tout seul à réaliser ce projet. Pour trouver les composantes connexes on pourra réutiliser le programme du *TP1* et pour enlever les composantes connexes dans le graph on pourra utiliser la méthode *enlevePoint*. Une autre amélioration possible est s'il existe deux sommets adjacent alors on supprime un des deux sommets. J'ai commencé à implémenter cette amélioration mais j'avais une erreur et par manque de temps encore je n'ai pas pu le corriger mais pour le réaliser j'ai reutiliser la méthode *enleveVoisin* en le modifiant un peu.