Université de Caen-Normandie

DM de Graphe et Recherche Arborescente

Auteurs :
Guillaume Drouart
Benjamin Lemaitre

Correcteur :
Patrice Boizumault

Diplôme préparé : M1 Informatique

Introduction

Vous trouverez dans l'archive le code source dans le dossier "Graphe" et vous trouverez aussi le fichier "DM_Graphe_Drouart_Lemaitre.pdf" qui contient les réponses aux questions.

Afin d'implémenter l'algorithme de Kruskal et celui de 2-approximation, nous avons choisi d'utiliser le langage Java.

Nous avons fait ce choix, car java est un langage avec lequel nous sommes à l'aise et qui permet d'utiliser des ArrayList qui sont des structures de données efficaces et qui permettent d'utiliser l'algorithme de trie par défaut des collections de Java, cet algorithme est le mergesort et a une complexité au pire assurée en $n\log(n)$ ce qui est très efficace pour répondre aux problèmes de tri de l'algorithme de Kruskal.

Dans le dossier "Graphe" vous avez les dossiers "src", "data" et le fichier "build.xml" permettant de compiler le code avec ant.

- Pour compiler, vous utilisez la commande "ant compile"
- Pour créer l'exécutable "Graphe.jar" vous utilisé la commande "ant jar"
- Pour tester exécuter vous utilisé la commande "ant run1" qui permet d'exécuter sur le fichier "dm.tsp" qui contient l'exemple dans le sujet
- Vous pouvez aussi tester les fichiers en utilisant la commande "java -jar jar/Graphe.jar data/dm.tsp"
- Enfin vous pouvez tester d'autre fichier du dossier "data" avec "ant run2", "ant run3" et "ant run4"
- Pour compiler et lancer, vous utilisez la commande "ant all"

Table des matières

Chapitre 1

Partie 1 Kruskal

1.1 Question 1.1

1.1.1 Question 1.1 a

— Création d'un ensemble qui contiendra les arêtes de l'ARPM.
— Création d'un ensemble pour chaque sommet, chaque sommet est donc seul dans son ensemble.
— Trier les arêtes pour pouvoir les traiter par ordre croissant de poids, car les arêtes de poids faible sont plus intéressantes.
— Dans l'ordre des poids des arêtes, on va regarder si l'arête est à ajouter à l'ensemble.
— On vérifie que l'arête ne relie pas deux sommets d'un même ensemble, car le résultat attendu est un ARPM et une telle arête créerait un cycle : ce ne serait donc plus un arbre.
— Si les deux sommets sont déjà dans le même ensemble, ils le sont grâce à des arrête de poids plus faible, car on les traite par ordre croissant donc il est inutile de chercher à remplacer une autre arrête par celle-là.
— L'arrête est à ajouter à l'ARPM donc on l'ajoute à l'ensemble E
— On a gardé l'arête entre u et v donc on fusionne les ensembles pour représenter ce lien.
— La boucle s'arrête après avoir parcouru toutes les arêtes.
— On récupère notre ARPM.

1.1.2 Question 1.1 b

Il n'est pas toujours indispensable d'envisager toutes les arêtes, car le nombre d'arêtes d'un arbre recouvrant de poids minimum (ARPM) est toujours égal au nombre de sommets moins 1. Donc la boucle peut s'arrêter lorsque le nombre d'arêtes de l'ensemble atteint ce nombre, sinon ce serait un cycle.

1.2 Question 1.2

Par commodité nous utilisons des ArrayList dans notre code Java mais ici notre pseudo-code utilise des tableaux.

1.2.1 Question 1.2 a

chaque noeud est racine et n'a donc pas de noeud parent

```
CRÉER-ENSEMBLE(x : Elément)
debut
T[x]=null
fin
```

1.2.2 Question 1.2 b

méthode récursive

```
TROUVER-ENSEMBLE(x : Elément) ->Elément
debut
si T[x]==null
alors retourner x
sinon retourner TROUVER-ENSEMBLE(T[x])
fin si
fin
```

méthode non récursive

```
TROUVER-ENSEMBLE(x : Elément) ->Elément
debut
element racine=x
tant que T[racine] different de null faire
racine=T[racine]
fin tant que
retourner racine
fin
```

1.2.3 Question 1.2 c

```
UNION(x,y : Elément)
debut
element parentX= TROUVER-ENSEMBLE(x)
element parentY= TROUVER-ENSEMBLE(Y)
T[parentX]=parentY
fin
```

1.3 Question 1.3

Dans le pire cas si on ne prend aucune précaution particulière lors de la procédure UNION(x, y : Elément),la hauteur maximale de l'arbre obtenu sera le nombre de ses noeuds (ou éléments) -1 c'est à dire n-1 car l'arbre obtenu est une chaine.

1.4 Question 1.4

Si l'on fait le choix de l'équilibrage, la taille d'un arbre n'augmente que si les unions sont fait entre des arbres de même hauteur, sinon le plus grand sert de racine donc la taille n'augmente pas.

Pour le cas n=0 on a k=1 et h=0 donc $0 \leq \log(1)$ Donc l'équation est vérifier pour le cas 0.

Dans le cas n on a k=n et on suppose que $h \leq \log(k_n)$ est vrai donc deux cas possible :

- Le cas avec l'union de deux arbres de mêmes taille donc les deux arbres ont eu au pire $\log(k)$ unions, et la hauteur augmente $h_{n+1} = h_n + 1$

On a donc : $\log(k_{n+1}) = \log(k_n + k_n + 1) = \log(2k_n + 1) = \log(2) + \log(2k_{n+\frac{1}{2}}) = 1 + \log(2k_{n+\frac{1}{2}})$

La fonction logarithme est strictement croissante donc : $1 + 1 + \log(2k_{n+\frac{1}{2}}) > 1 + \log(2k_n)$
on a donc : $h_{n+1} \leq 1 + h_n \leq 1 + \log(2k_n \leq \log(2k_{n+1})$
alors $h_{n+1} \leq \log(k_{n+1})$.

- Le cas avec l'union de deux arbres de tailles différentes donc la hauteur n'augmente pas : $h_{n+1} = h_n$.

La fonction logarithme est strictement croissante donc : $\log(k_{n+1}) > \log(k_n)$

on a donc : $h_{n+1} \leq \log(k_n) \leq \log(k_{n+1})$ alors $h_{n+1} \leq \log(k_{n+1})$

On vois bien que dans les deux cas on a : $h_{n+1} \leq \log(k_{n+1})$

L'hypothèse est vérifiée pour le cas 0, on a supposé que dans le cas n elle était valide, et on en a déduit que pour le cas n+1 elle est valide.

Donc l'hypothèse est toujours valide.

1.5 Question 1.5

On pose n = le nombre de nœuds du graphe.

le nombre d'arête d'un graphe complet est : $m = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

la complexité du parcours d'un arbre est au maximum de : $log(n)$

dans le pire cas on dois effectuer 4 parcours d'arbre par arête car il en faut deux lors du test (TROUVER-ENSEMBLE(u) et TROUVER-ENSEMBLE(v)) et deux lors du union (Union(u,v), et on fera m-1 union donc $2(n-1)\log(n)$ parcours dans les unions

la complexité des autres éléments de l'algorithme est négligeable, donc la complexité sans le tri au pire est de :

$$n(n-1)\log(n) + 2(n-1)\log(n) = (n^2 + n - 2)\log(n) \tag{1.1}$$

la complexité du tri est de : $m\log(m) = \frac{n(n-1)}{2}\log(\frac{n(n-1)}{2})$

donc la complexité du tri est d'ordre $\mathcal{O}(n^2 \log n^2)$

vu que la complexité sans le tri est d'ordre $\mathcal{O}(n^2 \log n)$

cela est négligeable par rapport au tri donc la complexité total au pire de l'algorithme est d'ordre $\mathcal{O}(m \log m)$ qui est la complexité du tri.

1.6 Question 1.6

La compression de chemin permet de parcourir moins de nœuds lors des prochaines exécution de la fonction TROUVER-ENSEMBLE(x) sur l'un des nœuds parcouru.

Vu que la complexité au pire est d'ordre $\mathcal{O}(m \log m)$, la compression du chemin est négligeable.

1.7 Question 1.7

On utilise R[x] un tableau marquant la profondeur a partir d'un nœud.

```
CRÉER-ENSEMBLE(x : Elément)
Debut
T[x]=null
R[x]=0
Fin

TROUVER-ENSEMBLE(x : Elément) ->Elément
Debut
Element racine=x
Tant que T[racine] different de null faire
racine=T[racine]
Fin tant que
Elément n = x
Tant que n different de racine faire
Element temp=T[n]
T[n]=racine
n=temp
Fin tant que
Retourner racine
Fin
```

```
UNION(x,y : Elément)
Debut
Element parentX= TROUVER-ENSEMBLE(x)
Element parentY= TROUVER-ENSEMBLE(Y)
T[parentX]=parentY
Si R[x]<R[y] alors
T[x]=y
Sinon si R[x]>R[y] alors
T[y]=x
Sinon
T[y]=x
R[x]=R[x]+1
Fin Si
Fin
```

Chapitre 2

Partie 2 2-Approximation

2.1 Question 2.1

On sait que $w(C^*) = z^*$, or z^* est le cout du cycle C^*.
C^* est un cycle hamiltonien donc si on lui retire une arête on obtient un arbre recouvrant, or M est un ARPM donc le poids de l'ARPM est inférieur au poids du cycle : $w(M) < z^*$ car C^* est un arbre avec une arête en plus.

Le poids de C est deux fois le poids de M car on a doubler les arêtes, $2 \times w(M) < 2 \times z^*$ donc : $w(C) < 2 \times z^*$

2.2 Question 2.2

On peut supprimer les sommets de C visité plus d'une fois car on a l'inégalité triangulaire, donc : $w(C') < w(C)$ et vu que $w(C) < 2z^*$ alors $w(C') < 2z^*$

2.3 Question 2.3

On avait démontré dans l'algorithme de Kruskal que la complexité au pire pour générer un ARPM est de l'ordre $\mathcal{O}(n^2 \log n^2)$ avec n le nombre de sommet.
C'est donc la complexité de la première étape du 2-approximation.

La deuxième étape du 2-approximation est d'ordre $\mathcal{O}(n)$ car il suffit d'ajouter une arête pour chaque arête de l'ARPM et il y a n-1 arête dans un ARPM de n sommets.

La troisième étape du 2-approximation est d'ordre $\mathcal{O}(2 \times n)$ car on parcourt un cycle avec $2n - 2$ arêtes.

Donc la complexité temporelle au pire de l'algorithme de 2-approximation est d'ordre : $\mathcal{O}(n^2 \log n^2)$ c'est a dire la première étape, les autres étapes étant négligeables.

Chapitre 3

Partie 3 Christofides

3.1 Question 3.1

Le nombre de sommet de degré impaire d'un multi-graphe sans arrête est égale à 0, il est donc pair.

pour chaque nouvelle arrête, il y a trois possibilités :

- Les deux sommets étaient de degré pair alors les deux sommets deviennent de degré impaire car le degré augmente de 1 pour chacun et le nombre de sommets de degré impaire augmente de 2.
- Les deux sommets étaient de degré impair alors les deux sommets deviennent de degré pair car le degré augmente de 1 pour chacun et le nombre de sommets de degré impaire diminue de 2.
- Un sommet était de degré pair et l'autre sommet de degré impair alors celui de degré impair devient de degré pair et inversement car le degré augmente de 1 pour chacun et le nombre de sommet impair reste inchangé.

Donc un multi-graphe sans arrête possède un nombre pair de sommets de degré impair et chaque nouvelle arrête ne peut modifier ce nombre que de 2 donc ce nombre est toujours pair.

3.2 Question 3.2

On sait que tous multi-graphe contient un nombre pair de sommet de degré impair.

G' est constitué des sommets de degré impair de M il a donc un nombre pair de sommet.

Or vu qu'il a un nombre pair de sommet il est possible de les lier deux à deux.

Donc il existe au moins un couplage parfait de G'.

3.3 Question 3.3

On est sur que H possède un cycle eulérien car tout graphe connexe ayant tous ses sommets de degré pair possède un cycle eulérien (théorème des cycles eulérien) et H est connexe car il a au moins toutes les arêtes de M qui est un Arbre Recouvrant et tous ses sommets sont de degré pair car pour chaque couple de sommets de degré impair de M on a ajouté une arête de M'.

3.4 Question 3.4

3.4.1 Question 3.4-a

Le nombre de sommets de G' est inférieur ou égale au nombre de sommets de M car il est composé de ses sommets impairs.

z^* est la longueur d'un cycle hamiltonien de longueur minimal.

M^* est un couplage parfait de G' de coût minimal.

Le nombre d'arête d'un couplage est égale à la moitié du nombre de sommet : $\frac{n}{2}$.

Pour obtenir un cycle on ajoute a ce couplage un autre couplage du même graphe, si le couplage est de poids minimal alors sont poids est inférieur ou égale à la moitié du poids du cycle car sinon le deuxième couplage serait de poids inférieur, or le couplage est de poids minimal car il n'existe pas de couplage de poids inférieur.

Comme le nombre de sommets de G' est inférieur ou égale au nombre de sommets de M, la longueur du cycle hamiltonien de longueur minimale de G' est inférieur a la longueur du cycle hamiltonien de longueur minimale de G.

Donc on sait que la longueur du couplage parfait de poids minimal $w(M^*)$ est inférieur ou égale à $\frac{1}{2}$ de la longueur du cycle hamiltonien de G' de longueur minimal et que cette longueur est elle même inférieur ou égale à z^*.

Donc on a bien : $w(M^*) \leq \frac{1}{2}z^*$

3.4.2 Question 3.4-b

On sait que $w(M) < z^*$ comme démontré dans la question 2.1 et on sait que $w(M^*) \leq \frac{1}{2}z^*$ comme démontré dans la question précédente et que la longueur de H est égale à la longueur de M plus la longueur de M^* : $w(H) = w(M) + w(M^*)$.

Donc $w(H) \leq z^* + \frac{1}{2}z^* \equiv w(H) \leq \frac{3}{2}z*$

De la même manière que pour la question 2.2 on sait que $w(C') < w(H)$.

Donc $w(C') < \frac{3}{2}z^*$

3.5 Question 3.5

1. Création du sous graphe G' avec les sommets de degré impair : A,M,E,C
2. Recherche d'un couplage parfait minimal M^* dans G' => (A,M),(E,C)
3. Création du multi-graphe $H = M \cup M^*$
4. Création du cycle eulérien C à partir de H => A,M,S,E,C,B,S,A
5. Création du cycle hamiltonien C' à partir du cycle C => A,M,S,E,C,B,A
6. Retourner le cycle et sa longueur

3.6 Question 3.6

On avait démontré dans l'algorithme de Kruskal que la complexité au pire pour générer un ARPM est de l'ordre $\mathcal{O}(n^2 \log n^2)$ avec n le nombre de sommet.

C'est donc la complexité de la première étape du Christofides.

La deuxième étape du Christofides est d'ordre $\mathcal{O}((\frac{n}{2})^3) = \mathcal{O}(n^3)$ comme donné dans l'énoncé.

La troisième étape du Christofides est d'ordre $\mathcal{O}(\frac{n}{2})$ car on parcourt le couplage pour l'ajouter au graphe.

La quatrième étape du Christofides est d'ordre $\mathcal{O}(n)$ car on parcourt le cycle eulérien pour l'ajouter au cycle hamiltonien.

Donc la complexité temporelle au pire de l'algorithme de Christofides est d'ordre : $\mathcal{O}(n^3)$ car $n > \log(n^2)$ soit $n^3 > n^2 \log(n^2)$ c'est a dire la deuxième étape, les autres étapes étant négligeables.