Rapport Gogol Car

ENNAJI Samia
ESPASA Kévin

30 November 2016

Sommaire

1 Introduction

L'objectif de ce projet est de modéliser et d'implémenter un programme visant à utiliser les notions de graphes vu en cours . Nous commencerons par modéliser le problème et présenter les différentes variantes à implémenter, avec les solutions trouvées pour chaque problème. Enfin, Nous développerons nos algorithmes en pseudo-code et analyserons la complexité globale de notre code.

2 Manuel utilisateur

Ligne de commande pour compiler depuis le dossier src: **javac *.java**
Ligne de commande pour exécuter depuis le dossier src: **java Main**
Ligne de commande pour générer la Javadoc dans le dossier "doc" depuis le dossier racine:
javadoc -d doc -encoding UTF8 -docencoding UTF8 -charset UTF8 src/*.java

Fonctionnement du logiciel:

1. Entrer le nom du fichier de données. L'extension doit être en .txt. Le fichier doit être de la forme:
   Place 1.
   Rue;Place 1;Place 2.
   Il ne doit pas y avoir d'espaces avant ou après les séparateurs (";" ou "."). Si le fichier est corrompu ou n'existe pas le programme s'arrête.

2. Entrer la place de départ. Tant que la place n'est pas correct le logiciel redemande le nom de la place de départ.

3. Entrer 1 pour un parcours GogolS, 2 pour un GogolL et 3 pour un GogolXL.

4. Le résultat s'affiche dans le terminal. Il est également présent dans le fichier *GogolCarResult.txt* à la racine de l'application.

3 Modélisation du problème

3.1 Structures des données

Pour répondre à l'objectif du projet, nous avons identifié le problème pour chaque variante . Nous avons choisi de modéliser sous forme de graphe, les sommets sont représentés par des Places et les arêtes le sont par des Rues. Une place contient une liste de rues, tandis qu'une rue contient les places présentes à ses extrémités. Cette structure contient un inconvénient, lors de la constructure à partir du fichier, l'ordre des places dans la rue dépend de la façon dont est écrite la ligne du fichier. Par exemple, si la phrase est:
Rue;Place A;Place B.

L'objet rue contiendra dans l'attribut place1 la Place A et dans place2 la place B. Ce qui implique qui si l'on est dans la Place A et que l'on veut faire un parcours récursif, on doit fait *parcours(Rue.place2)*, tandis que si l'on est dans la Place B on doit faire *parcours(Rue.place1)*. On se retrouve donc à tester à chaque fois que l'on est dans une place et que l'on veut aller vers une autre place si la place actuelle est présente sur l'attribut place1 ou l'attribut place2 de la rue. Pour simplifier le pseudo code et ne pas le surcharger, on fait le choix de considérer que la place courante est positionnée sur l'attribut place1 de rue.

4 Variantes

4.1 Variante 1: Algorithme GogolS

Dans cette variante, la Gogol car ne peut photographier que du côté droit de la voiture.Ce programme permet à la Gogol Car de prendre le chemin le plus court possible de la ville et de photographier les deux côtés de la rue. Pour cette première variante , on a choisi de résoudre ce probléme en utilisant un parcours en profondeur sur la classe ville qui représente notre graphe .
Le parcours en profondeur est un algorithme de recherche qui progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S. La classe Rue représente nos arêtes et la classe place représente nos sommets.

Algorithm 1 Procédure Traverse de la classe GogolCarS

```
0: procedure TRAVERSE(p : Place)
0:   visite[p] ← Vraie;
0:   for all Rue r de p do
0:     if ruevisite[r] = Faux then
0:       resultat ← resultat + r.place1.Nom + r.place2.Nom;
0:       ruevisite[r] ← Vraie;
0:       TRAVERSE(r.place2);
0:       resultat ← resultat + r.place2.Nom + r.place1.Nom;
0:     end if
0:   end for
0: end procedure=0
```

On traverse toutes les rues grâce à un parcours en profondeur, On ne regarde pas si une place été visitée mais si une rue l'a été. Si ce n'est pas le cas on sauvegarde ce passage dans le résultat puis on la marque comme visité, on appelle la fonction sur la seconde place. Lors de la "fermeture" des appels récursifs, on ajoute au résultat le chemin inverse. La ville étant obligatoirement connexe, en partant de n'importe quel sommet on arrive à visiter toutes les rues.

La complexité est en O(Places+Rues).

4.2 Variante 2:Algorithmes GogolL et GogolXL

Dans cette variante, la Gogol car peut photographier les deux côtés de la rue en un seul passage. On parle ici d'un graphe non-orienté , le but est de faire cela en minimisant la longueur du trajet, définie comme le nombre d'arêtes parcourues entre le départ de la Gogol de sa position initiale et son retour à cette même position. Dans cette Variante , on a développé deux différents algorithmes :GogolL et GogolXL.

4.2.1 Algorithme GogolL

Cet algorithme fonctionne dans le cas particulier où le graphe défini par la ville contient un cycle eulerien (un cycle qui passe exactement une fois par chaque rue). On a commencé par trouver une anti arborscence (un grahe avec les arcs inversés) en utilisant un parcours en profondeur. On a numéroté nos rues (arêtes) en fonction du fait qu'elles soient couvrantes ou non , puis nous avons trié les rues en fonction de leur numéro assigné avant. Lors de la traversé de la ville on a donc plus qu'à parcourir la liste des rues d'une place dans l'ordre.

Algorithm 2 Procédure Anti Arborescence de la classe Ville

```
0: procedure antiArbo(p : Place)
0:   visite[p] ← Vraie;
0:   for all Rue r de p do
0:     if visite[r.place2] = Faux then
0:       visite[r.place2] ← Vraie;
0:       couvrante[r] ← Vraie;
0:       antiArbo(r.place2);
0:     end if
0:   end for
0: end procedure=0
```

Complexité: O(Places + Rues)

Algorithm 3 Procédure Numérote et Trie de la classe Ville

```
0: procedure numEtTri
0:   for all Place p de Ville do
0:     max;
0:     min ← 0;
0:     tailleMax ← p.nbRues;
0:     if tailleMax − 1 = min then
0:       max ← tailleMax;
0:     else
0:       max ← tailleMax − 1;
0:     end if
0:     for all Rue r de p do
0:       if couvrante[r] = Vraie then
0:         numero[r] ← tailleMax;
0:       else if couvrante[r.inverse] = Vraie then
0:         numero[r] ← max;
0:         max ← max − 1;
0:       else
0:         numero[r] ← min;
0:         min ← min − +1;
0:       end if
0:     end for
0:     Trier(numero)
0:   end for
0: end procedure=0
```

```
Complexité: O(Places * Rues)
```

Algorithm 4 Procédure Traverse de la classe GogolCarL

```
0: procedure traverse(p : Place)
0:   for all Rue r de p do
0:     if ruevisite[r] = Faux then
0:       resultat ← resultat + r.place1.Nom + r.place2.Nom;
0:       ruevisite[r] ← Vraie;
0:       ruevisite[r.inverse] ← Vraie;
0:       traverse(r.place2);
0:     end if
0:   end for
0: end procedure=0
```

```
Complexité: O(Places + Rues)
Complexité Global de GogolS: O(Places * Rues)
```

4.2.2 Algorithme GogolXL

Cet algorithme doit fonctionner même dans le cas où le graphe défini par les rues de la ville n'est pas eulérien.Afin de pouvoir proposer une solution à ce problème on s'est basés sur l'algorithme du postier chinois. On a commencé par définir une méthode qui trouve tous les sommets de degré impair à partir du graphe G et de former un nouveau graphe G' contenant juste les sommets de degré impair trouvés et les rues les reliant entre eux. Si le graphe possède un nombre impair de sommets impair,dans ce cas l'algorithme du postier chinois n'est applicable. Ensuite, Entre deux sommets du graphe G', on ajoute une arête dont la longueur représente le plus court chemin entre les deux sommets dans le graphe G. Pour le calcul du plus court chemin , on a choisi d'utiliser l'algorithme de Floyd Warshall , cet algorithme prend en entrée un graphe (ville pour notre programme), décrit par une matrice de distances.Pour les indices enregistrer des positions dans la matrice de distances on a ajouté une classe indice qui contient 2 attributs: l'indice en x et celui en y de la case qui nous intéresse. Après cette étape on a definis une méthode pour trouver un couplage parfait de poids minimum dans G'. Cette algorithme est glouton, on cherche les valeurs les plus petites et on les sauvegarde. Ensuite, pour chaque paire d'indices on crée une nouvelle rue dans le graphe G. Notre ville est donc eulérienne , on peut appliquer notre algorithme de GogolL. Dans le cas , ou le graphe contient un cycle eulérien , on applique directement l'algorithme de GogolL.

Algorithm 5 Fonction qui créer une ville Impair à partir des places de degré impair d'une ville

```
0: function IMPAIRVILLE
0:   for all Place p de Ville do
0:     if p.nbRues mod 2 != 0 then
0:       nouvelleVille ← nouvelleVille + p;
0:     end if
0:   end for
0:   for all Place p de nouvelleVille do
0:     for all Rue r de p do
0:       if r.place2 ∉ nouvelleVille then
0:         listeRues[p] ← listeRues[p] − r;
0:       end if
0:     end for
0:   end forreturn nouvelleVille
0: end function=0
```

```
Complexité O(Places*Rues)
```

Algorithm 6 Procédure Floyd Warshall de la classe Ville

0: **procedure** FLOYDWARSHALL($matriceDist[], MatriceSucceseur[], listePlace$)
0: INITMATRICE(matriceDist[],MatriceSucceseur[],listePlace)
0: **for** k de 0 à listePlace.taille()-1 **do**
0: **for** i de 0 à listePlace.taille()-1 **do**
0: **for** j de 0 à listePlace.taille()-1 **do**
0: **if** matriceDist[i,k]+matriceDist[k,j]¡matriceDist[i,j] **then**
0: $matriceDist[i,j] \leftarrow matriceDist[i,k] + matriceDist[k,j]$;
0: $matriceSuccessuer[i,j] \leftarrow listePlace[k]$;
0: **end if**
0: **end for**
0: **end for**
0: **end for**
0: **end procedure**=0

```
Complexité O(Places^3)
```

Algorithm 7 Fonction qui créer un couplage entre 2 Places

0: **function** COUPLAGE($nouvelleVille$: Ville, $mDistance[], listePlace$)
0: $couple \leftarrow \emptyset$;
0: **for** i de 0 à listePlace.taille()-1 **do**
0: $mDistance[i,i] \leftarrow Max_Value$;
0: **end for**
0: **for** i de 0 à listePlace.taille()-1 **do**
0: $distanceMin \leftarrow Max_Value$;
0: **for** j de 0 à listePlace.taille()-1 **do**
0: **if** $(listePlace[i], listePlace[j]) \notin couple$**and**$(listePlace[i], listePlace[j]) \in nouvelleVille$**and**$mDistance[i,j] < distanceMin$ **then**
0: $couple \leftarrow (listePlace[i], listePlace[j])$
0: **end if**
0: **end for**
0: **end forreturn** $nouvelleVille$
0: **end function**=0

```
Complexité O(Places^2)
Complexité de GogolXL: O(Places^3)
```

5 Complexité

Variante	complexité
GogolS	O(Places+Rues)
GogolL	O(Places*Rues)
GogolXL	O(Places^3)
Global	O(Places^3)

6 Conclusion

La réalisation de ce projet nous a permis de mieux comprendre l'algorithme de Floyd Warshall,et de maitriser les algorithmes de cours (comme les algorithmes de parcours dans un graphe). Il nous a également permis de connaître le problème du postier chinois.

7 Annexe

Figure 1: Résultat de GogolS sur euler2.txt

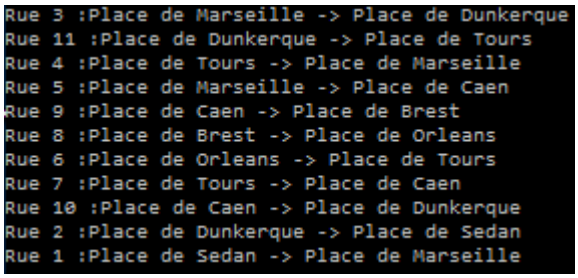

Figure 2: Résultat de GogolL sur euler2.txt

Résultat de GogolXL sur Noneuler.txt: GogolCarResult.txt

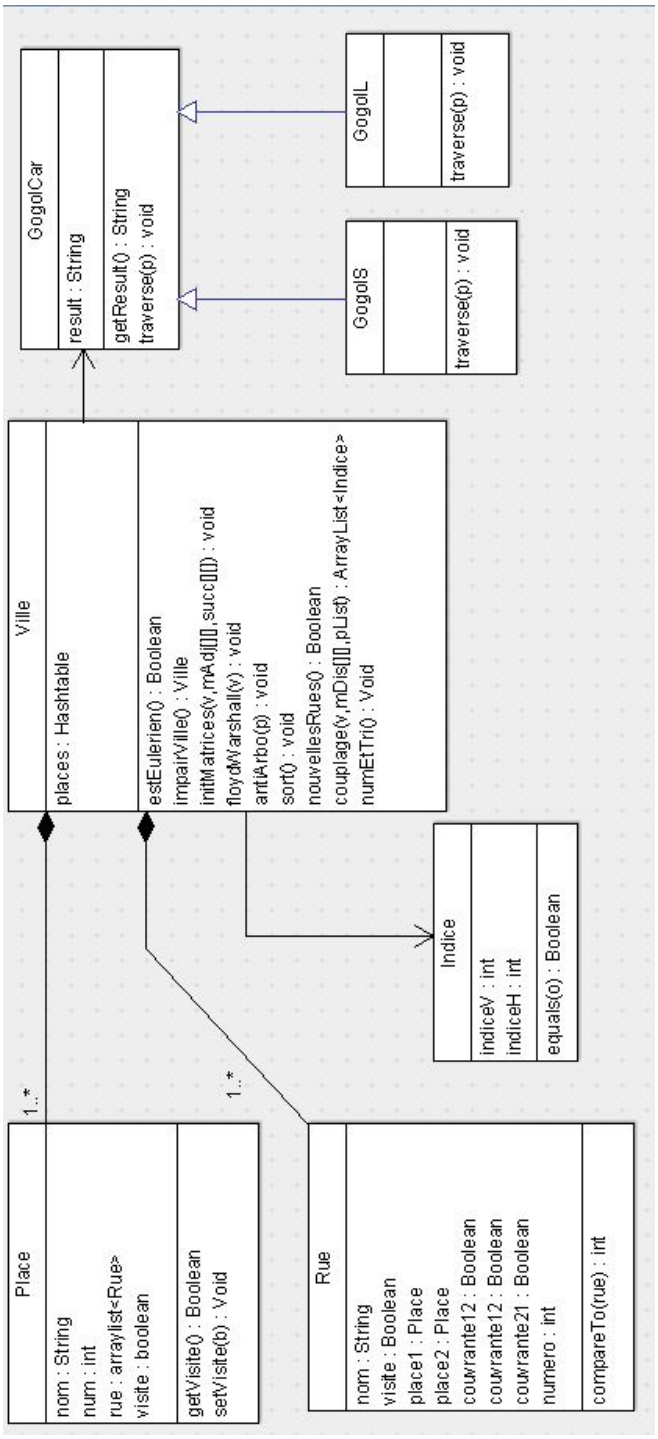

Figure 3: Diagramme de classes