Représentation Avancée de Données

05/04/2023

1 Questions techniques

Ce projet est à rendre au plus tard le **mercredi 17 mai 2023 à 23h59** sur l'espace prévu à cet effet sur Arche. Il vous est possible de modifier votre projet autant de fois que vous le souhaitez sur Arche, aussi je vous conseille de le déposer r égulièrement afin d'avoir une copie de sauvegarde en cas de problème. **Il vous sera nécessaire de valider le dépôt pour que le projet soit définitif et que je le considère comme rendu**. Je retirerai 2 points sur la note pour chaque jour de retard. Ce projet comptera pour la moitié de la note du cours.

chaque question sera indiqué par un ★.

Il s'agira de déposer un fichier java **très fortement commenté** (au moins 20% du code devrait être du commentaire informatif.Vous êtes donc invités à expliquer ce que font toutes vos méthodes, ce que sont vos attributs ainsi que la fonction de lignes de code qui ne seraient pas évidentes) pour le code et un fichier texte (txt, pdf, docx, odt,...) pour les questions.

Lorsqu'il s'agit d'implémentation, les consignes indiquent des éléments nécessaires. Vous avez toute liberté pour **ajouter des attributs et fonctions annexes au besoin**. Il est en revanche d'autant plus indispensable de les commenter.

Ce projet est à faire individuellement. Vous pouvez bien sur vous entraider, mais **le plagiat est formellement interdit**.

n'hésitez surtout pas à me contacter sur clement.beysson@univ-lorraine.fr si vous avez le moindre soucis.

2 Introduction

Avec l'ouverture à la concurrence du réseau ferré, une entreprise souhaite investir dans le transport ferroviaire. Cependant, elle n'a pas les moyens d'ouvrir des lignes sur toutes les voies existantes. Elle cherche donc un moyen de créer un réseau permettant d'aller de n'importe quelle ville à n'importe quelle autre, mais sans ligne superflue. Elle souhaite également arriver à cet objectif avec un coût minimal.

Nous allons donc commencer par modéliser un réseau ferré à l'aide d'une nouvelle structure de donné : le graphe. Ensuite, nous utiliserons l'algorithme de Kruskal pour trouver les meilleurs lignes à ouvrir.

3 Graphe

Commençons par définir ce qu'est un graphe.

Il s'agit d'un ensemble de noeuds reliés par des arcs. Par défaut, ces arcs ne sont pas orientés (Si à est relié à B alors B est relié à A).

Dans le cas de notre problème, les noeuds correspondront aux villes et les arcs au voies ferrées existantes. Nous aurons de plus besoin d'étoffer un peu cette structure de base en ajoutant un poids strictement positif sur chaque arc afin de représenter le coût d'exploitation de cette ligne.

Pour une telle structure, les fonctions usuelles sont :

add_edge ajoute un arc entre deux noeuds du graphe, en en précisant le poids. Si cet arc est déjà présent, on se contente de mettre à jour son poids

add_node ajoute un noeuds donné au graphe. Ce nouveau noeuds n'est à priori relié à aucun autre

++are_connected indique s'il existe un chemin permettant de relier deux noeuds en suivant les arcs du graphe

del_edge supprime un arc du graphe

del_node supprime un noeuds donné du graphe. Ceci entraine la suppression de tous les arcs y étant connectés

++get_degree retourne le nombre d'arcs connectés à un noeud donné

++get_lightest retourne l'arc de poids le plus faible

++get_weight retourne le poids d'un arc

graph génère un nouveau graphe vide

++num_edge retourne le nombre d'arcs total

4 Spécification algébrique

★ Vous renseignerez la spécification algébrique des graphes. Pour rappel, il sera nécessaire de donner la signature des fonctions, puis les préconditions et enfin les axiomes.

★ Vous implémenterez une classe abstraite *Graph* correspondant à cette spécification.

5 Matrice de relation

Nous allons maintenant proposer une implémentation possible des graphes : la matrice de relation.

Cette implémentation a pour particularité de stocker à des endroits différents l'arc e_ij et l'arc e_ji. **Il faudra bien faire attention de toujours les modifier en même temps**.

Nous utiliserons deux attributs :

nodes est une liste de *char* contenant le nom des noeuds. Ainsi, on pourra par la suite utiliser l'index du noeuds dans cette liste pour en parler, et récupérer son nom si besoin.

edges est une matrice (un tableau à double entrée, que l'on peut implémenter comme un tableau de tableau) indiquant si un arc relie deux noeuds. Ainsi, si Paris (d'index 0 dans nodes) et Nancy (d'index 1 dans nodes) sont reliés par un arc de poids 350, alors edges[0][1] sera égal à 350. Si ces deux villes ne sont pas relier, on indiquera 0 par convention.

★ Vous implémenterez une classe *Graph_matrix*, héritant de *Graph*, possédants ces attributs et les fonctions décrites dans votre spécification algébrique.

6 Algorithme de Kruskal

Nous disposons à présent d'une classe gérant les graphes, ce qui nous permet de modéliser notre réseau ferré. Il nous faut encore disposer d'un algorithme pour sélectionner les voies à conserver pour notre entreprise. Pour cela, il existe la notion d'arbre couvrant minimal qui correspond exactement à ce que nous recherchons, et un algorithme (celui de Kruskal) qui nous le trouve.

Voici donc cet algorithme en pseudocode :

```
G1 est le graph contenant tous les arcs encore à explorer
G2 est le graph correspondant à l'arbre couvrant en cours
    de construction
N le nombre de noeuds dans le graphe
--
G1 = graph
G2 = empty_graph
tant que {G2 a moins de N-1 arcs et G1 a au moins un arc} faire
    choisir parmis les arcs de G1 celui de moindre poids : E_ij
    supprimer E_ij dans G1
    si i n'est pas dans G2
    ajouter i à G2
fin si
si j n'est pas dans G2
    ajouter j à G2
fin si
    si il n'existe pas de chemin de i à j dans G2
        ajouter E_ij à G2
    fin si
fin tant que
rendre G2
```

G2 correspond alors au réseau que notre entreprise devrait développer !
On va donc créer une classe *Entreprise* avec les attributs suivants :

villes contient une liste des villes du réseau

reseau_nationnal contient un graphe du réseau ferré global

reseau_entreprise contient le graphe correspondant au réseau que devrait maintenir l'entreprise

et les fonctions suivantes :

add_city ajoute une ville au réseau, sans voie

add_rail ajoute une voie ferré entre deux villes

del_rail supprime une voie ferré entre deux villes

optimal met à jour reseau_entreprise pour qu'il corresponde bien au réseau optimal pour l'entreprise, étant donné l'état actuel du réseau global

★ Vous implémenterez la classe *Entreprise*.

7 Alternative(s)

Nous disposons maintenant d'un programme utilisable. Cependant, il est très loin d'être optimal car l'implémentation choisie pour les graphes s'adapte assez mal à son utilisation dans l'algorithme de Kruskal.

★ On suppose disposer d'une classe *Edge* qui contient les deux extrémités d'un arc ainsi que sont poids. Quelle(s) structure(s) de donnée vu en cours jugez vous efficace pour stocker les arcs et représenter un graphe, dans l'optique de l'utiliser dans l'algorithme de Kruskal ? Justifier.

Bonne chance ☺