Projet LSINF1121 - Algorithmique et structures de données

-

Rapport final Mission 6

Groupe 26

Laurian Detiffe (6380-12-00)

Sundeep Dhillon (6401-11-00)

Alexis Macq (5910-12-00)

Xavier Pérignon (8025-11-00)

Thibaut Piquard (4634-13-00)

Thomas Wyckmans (3601-12-00)

Année académique 2015-2016

Introduction

Dans le cadre du cours "Algorithmique et structures de données", il nous a été demandé de concevoir et d'implémenter une application permettant de minimiser le coût d'exploitation de réseaux aériens en utilisant des graphes où les nœuds représentent les destinations et les poids associés aux arêtes représentes le coût des trajets aériens entre l'origine et la destination de ces arêtes.

1 Choix d'implémentation

Tout d'abord, nous avons choisi d'implémenter la classe `Controller` qui est la seule classe appelée par la `Main` classe. `Controller` appelle la classe `CitiesParser` qui permet de parser l'argument texte qui contient les villes ainsi que les coûts du vol entre deux villes afin de créer un graphe de type `AdjacencyMapUndirectedGraph`.
La classe `AdjacencyMapUndirectedGraph` permet de représenter et manipuler notre graphe représentant notre plan aérien.

Celui-ci est ensuite transformé grâce à la classe `MSTGraph`, qui s'appuie sur l'algorithme de *Prim*, en graphe `AdjacencyMapUndirectedGraph` minimisé.

2 Diagramme UML de classe

Voir Annexe.

3 Questions liées au problème posé

Question 1

Implémenter des tests unitaires vérifiant qu'une solution donnée au problème final est correcte, c'est-à-dire que le graphe correspondant contient tous les noeuds, qu'il n'y a pas de cycle (= il s'agit d'un arbre), qu'il s'agit bien d'un graphe connecté, que la solution est bien optimale, etc.

Tâche INGInious 3.

Question 2

Finalement, implémenter un algorithme de résolution du problème posé et produire une liste de connections retenues pour chacun des 2 réseaux fournis comme exemples. Les solutions proposées sont-elles uniques ? Ces solutions sont-elles optimales ? Pourquoi ? Vérifier vos solutions avec vos tests unitaires.

Puisque rien n'empêche que le coût de plusieurs connexions aériennes soit le même, plusieurs solutions valides de MST peuvent être obtenues d'un même graphe. La forme finale du MST dépendra des choix de priorité faits par la `Priority Queue` pour ces mêmes coûts. Ces solutions sont optimales vu que lorsqu'on applique l'algorithme de `Prim`, on conserve uniquement les arrêtes qui ont un faible coût et ce, sans avoir de cycle dans le graphe. En output, on a donc un arbre de recouvrement de poids minimal ; il s'agit d'une solution optimale.

Question 3

Analyser la complexité temporelle de votre algorithme d'un point de vue pratique et vérifier si celle-ci est conforme avec les bornes théoriques de l'algorithme retenu. Vous disposez pour cela de 2 instances particulières du problème, respectivement avec 50 et 500 villes. Au besoin, vous pouvez créer des instances complémentaires de taille variée pour affiner votre analyse. Posez-vous la question de savoir si le nombre de villes du réseau potentiel est le seul paramètre caractérisant la taille du problème. Pourquoi ?

Nous avons implémenté de *Prim* en se basant sur ce qui est décrit dans le livre DSAJ-5, dans la section 13.6.2.

fichier	nombre de vols	nombre de villes	temps
cities_small.txt	75	50	? ? ?sec
cities.txt	1250	500	? ? ?sec

TABLE 1 – Analyse temporelle

En théorie, la complexité temporelle de l'algorithme de `Prim` est en $O(log(n))$. Le nombre de villes dans le réseau est potentiellement le seul argument si l'on part du fait qu'il n'existe que maximum $\frac{n*(n-1)}{2}$ liens possibles entre les villes. Rapportant donc ainsi la complexité temporelle dans le pire cas à $O(n^2 \times logn)$.

Question 4

Décrire comment adapter la résolution du problème posé si l'on veut prendre en compte un coût additionel (fixé à 50) pour chaque ville qui sert d'étape intermédiaire pour connecter d'autres villes. Autrement dit, si dans la liste des connections retenues, l'aéroport de la ville B sert de transit pour un ou plusieurs vols reliant d'autres villes, on ajoute 50 au coût du réseau déployé pour cet aéroport de transit. Le problème devient alors de connecter l'ensemble des villes du réseau avec un coût total minimal, ce-inclus le coût des aéroports de transit. Votre algorithme de résolution de ce problème adapté a-t-il la garantie de produire une solution optimale ? Si oui, pourquoi ? Sinon, donnez un contre-exemple sur un réseau de quelques villes.

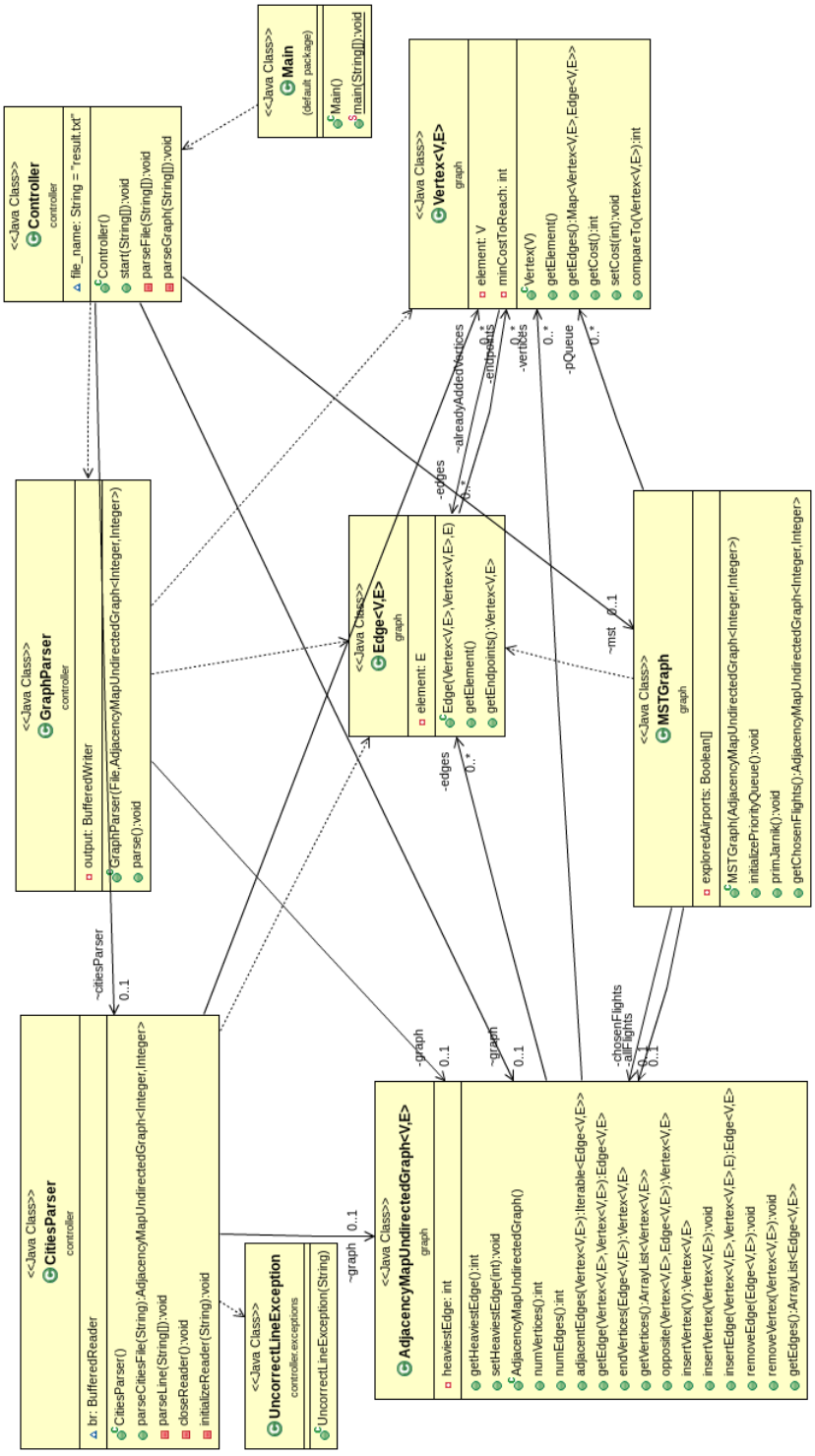
Il suffit de regarder pour chaque ville si celle-ci est transitaire. Si c'est le cas, on devra rajouter un poids de 50 à l'arrête adjacente. Ensuite, il faudra simplement vérifier pour chaque sommet (ville) qu'il est bel et bien impossible d'en faire une ville de transit en rajoutant un vol au réseau.
Cette condition sera vérifiée seulement si la nouvelle arrête ajoutée a un coût plus petit que (50 + le coût de l'ancienne arrête).

Pour implémenter cela, il suffit de recourir à l'algorithme de recherche `Brute-force`. Ensuite, on va essayer toutes les combinaisons possible et sélectionner la meilleure.

4 Difficultés rencontrées

En pratique, notre implémention du problème posé n'est pas optimale. En effet, sur la plate-forme INGInious, 2 tests sur 5 échouent.

5 Annexe


<<Java Class>>
Main
(default package)
Main()
main(String[]):void
<<Java Class>>
Controller
controller
file_name: String = "result.txt"
Controller()
start(String[]):void
parseFile(String[]):void
parseGraph(String[]):void
<<Java Class>>
Vertex<V,E>
graph
element: V
minCostToReach: int
Vertex(V)
getElement()
getEdges():Map<Vertex<V,E>,Edge<V,E>>
getCost():int
setCost(int):void
compareTo(Vertex<V,E>):int
<<Java Class>>
GraphParser
controller
output: BufferedWriter
GraphParser(File,AdjacencyMapUndirectedGraph<Integer,Integer>)
parse():void
<<Java Class>>
Edge<V,E>
graph
element: E
Edge(Vertex<V,E>,Vertex<V,E>,E)
getElement()
getEndpoints():Vertex<V,E>
<<Java Class>>
MSTGraph
graph
exploredAirports: Boolean[]
MSTGraph(AdjacencyMapUndirectedGraph<Integer,Integer>)
initializePriorityQueue():void
primJarnik():void
getChosenFlights():AdjacencyMapUndirectedGraph<Integer,Integer>
<<Java Class>>
CitiesParser
controller
br: BufferedReader
CitiesParser()
parseCitiesFile(String):AdjacencyMapUndirectedGraph<Integer,Integer>
parseLine(String[]):void
closeReader():void
initializeReader(String):void
<<Java Class>>
UncorrectLineException
controller.exceptions
UncorrectLineException(String)
<<Java Class>>
AdjacencyMapUndirectedGraph<V,E>
graph
heaviestEdge: int
getHeaviestEdge():int
setHeaviestEdge(int):void
AdjacencyMapUndirectedGraph()
numVertices():int
numEdges():int
adjacentEdges(Vertex<V,E>):Iterable<Edge<V,E>>
getEdge(Vertex<V,E>,Vertex<V,E>):Edge<V,E>
endVertices(Edge<V,E>):Vertex<V,E>
getVertices():ArrayList<Vertex<V,E>>
opposite(Vertex<V,E>,Edge<V,E>):Vertex<V,E>
insertVertex(V):Vertex<V,E>
insertVertex(Vertex<V,E>):void
insertEdge(Vertex<V,E>,Vertex<V,E>,E):Edge<V,E>
removeEdge(Edge<V,E>):void
removeVertex(Vertex<V,E>):void
getEdges():ArrayList<Edge<V,E>>
-alreadyAddedVertices
0..*
-endPoints
0..*
-vertices
0..*
-pQueue
0..*
-edges
0..*
-edges
0..*
~mst
0..1
~citiesParser
0..1
-graph
0..1
-graph
0..1
-graph
0..1
-chosenFlights
-allFlights
0..1
0..1