Calcul de plus courts chemins dans des graphes

1 Introduction

Le second TP d'Algorithmique 2 a pour but de manipuler des graphes, et notamment de coder et de tester divers algorithmes de calcul de plus courts chemins : l'application étudiée est celle d'un réseau de transports en commun.

La notion de graphe a été vue en Recherche Opérationnelle ; elle sera revue en Algorithmique 2 à la fin du semestre. Nous étudierons plus spécifiquement les représentations algorithmiques des graphes et diverses implémentations d'algorithmes vus en R.O. L'objectif de ce TP est triple :

1. manipuler une première représentation des graphes ;
2. programmer en Ada divers algorithmes très classiques dont on reparlera en CTD d'Algorithmique 2 ;
3. tester et comparer ces algorithmes.

Ce TP vous permettra également de manipuler des files de priorité (notion vue en CTD).

Comparativement au premier TP, celui-ci vous laisse plus de liberté dans le choix de vos implémentations. Corollaire : très peu de code vous est fourni. Les sections 3 à 5 détaillent les différents modules que vous avez à programmer en Ada. Vous devez rendre, outre ces modules, un court (2 pages maximum) compte-rendu.

2 Documents fournis

Les documents fournis sont les suivants :

— un fichier `reseauMetroRERTramParis2009.graph` (*courtoisie* Michel Desvignes), décrivant le graphe du réseau parisien de métro, RER et tramway ;
— un fichier `reseauTramGrenoble2010.graph`, décrivant le graphe du réseau grenoblois de tramway ;
— un fichier `GenAleaGraphe.adb`, comprenant une procédure permettant de générer un fichier de manière aléatoire `alea.graph`. Ce fichier décrit un graphe "aléatoire" comportant n nœuds et m arcs (n et m doivent être fournis par l'utilisateur).

Description du format `.graph`

Les graphes fournis ou générés sont stockés dans des fichiers au format `.graph`. Un fichier `.graph` comporte, dans l'ordre :

— une ligne avec le nombre n de nœuds et le nombre m d'arcs du graphe ;
— n lignes avec, sur chacune, un entier identifiant de manière unique un nœud et une chaîne de caractères indiquant le nom de la station correspondant à ce nœud ;
— m lignes avec, sur chacune, deux entiers indiquant les identifiants des nœuds source et destination d'un arc, un réel indiquant le coût associé à cet arc, et un nombre, un caractère ou une chaîne de caractères indiquant la ligne de métro/bus/tramway associée (le nombre 0 est indiqué dans le cas d'une correspondance).

Pour information, dans `reseauMetroRERTramParis2009.graph` le coût d'un arc correspond à la distance euclidienne entre les stations, sauf quelques cas particuliers :

— le coût d'une correspondance (même nom de station, mais changement de ligne) est de 120.0 ;

— le coût d'un trajet à pied (changement de ligne avec changement de nom de station) est de 240.0 ;
— le coût d'un trajet en tramway est de 40.0.

3 Représentation des graphes

Nous ne rappelons pas dans cet énoncé les définitions de base relatives aux graphes ; elles sont disponibles dans le polycopié de Recherche Opérationnelle que vous devez tous posséder. Ce polycopié est également disponible sur le Kiosk : `https://intranet.ensimag.fr/KIOSK/Matieres/3MMRO/POLY2009.pdf`

Nous allons représenter les lignes d'un service de transports en commun, par exemple la RATP ou la TAG, sous forme de graphe. Les nœuds du graphe correspondront aux stations de métro/tramway/bus, tandis que les arcs connecteront des stations voisines (c'est-à-dire joignables, sans arrêt intermédiaire, par un service de transport en commun). Pour une station donnée, il y aura **autant de nœuds dans le graphe que de lignes passant par cette station** : par exemple, trois nœuds correspondront à la station "Place d'Italie" pour le graphe du métro parisien. Les nœuds correspondant à la même station seront reliés par des arcs, pour modéliser les correspondances entre lignes.

Les graphes étudiés dans ce TP sont des graphes *orientés* (voir section 8 du polycopié de Recherche Opérationnelle) : en effet, bien que de nombreuses connexions entre stations voisines soient possibles dans les deux sens, certaines se font à sens unique (voir notamment la liaison Pré Saint Gervais-Danube de la ligne 7bis du métro parisien). Les nœuds des graphes étudiés seront munis des informations suivantes :
— un entier `Id` identifiant de manière unique le nœud étudié ;
— une chaîne de caractères `Station` indiquant le nom de la station.
La chaîne de caractères `Station` n'identifie pas de manière unique le nœud, car tous les nœuds correspondant à la même station partagent la même chaîne de caractères. Les arcs des graphes étudiés, outre un pointeur vers un nœud `Source` et un pointeur vers un nœud `Destination`, seront eux munis de deux types d'information :
— une chaîne de caractères `Ligne` représentant le numéro de la ligne (0 si correspondance) ;
— un réel **positif ou nul** `Cout` représentant le coût de transport du nœud source au nœud destination.
Ce coût a une interprétation qui varie selon le contexte : il peut correspondre à la durée du trajet ou au coût tarifaire, par exemple. Il peut être modulé en fonction du mode de transport : métro, bus, pieds (pour les stations comportant des correspondances).

Questions

1. Définir en Ada, dans un paquetage `Graphe`, un type `Noeud` et un type `Arc`, correspondant aux spécifications indiquées ci-dessus.
2. Ajouter au paquetage `Graphe` un type `Graphe`. Un graphe sera représenté par une matrice de `Noeud`, la case `[i,j]` de la matrice correspondant à l'arc allant du i-ème au j-ème `Noeud`, s'il existe (voir cours de Recherche Opérationnelle).

4 Algorithme de Bellman-Ford

L'algorithme de Bellman et Ford calcule le plus court chemin à partir d'un nœud source vers tous les autres nœuds du graphe. Voici une version, en pseudo-langage, qui fonctionne également pour les

graphes orientés avec circuits :

```
Distance(1..n) := (others => Float'Last);
Distance(Source) := 0;
pour i = 2 jusqu'à n faire
  pour chaque arc (u,v) du graphe faire
    DistTemp := Distance(u) + Cout(u,v);
    si DistTemp < Cout(v) alors
      Distance(v) := DistTemp;
    fin si
  fin pour
fin pour
```

Questions

1. Implémenter en Ada, dans un paquetage `BellmanFord`, la procédure `BellmanFord(G : Graphe ; Source : Noeud)`, qui calcule le coût du plus court chemin du nœud `Source` vers tous les autres nœuds du graphe `G` selon l'algorithme de Bellman-Ford. Vous êtes libres d'ajouter autant de champs que vous le désirez aux structures `Noeud`, `Arc` et `Graphe`.
2. Implémenter, dans le même paquetage, une procédure `AffichageBellmanFord(G : Graphe ; Source : Noeud ; Destination : Noeud)` qui affiche le détail du plus court chemin allant du nœud `Source` au nœud `Destination`. Ne seront affichées que les stations en correspondance (c'est-à-dire celles où l'on change de ligne). Exemple d'affichage possible :

   ```
   Plus court chemin de : Gare d'Austerlitz (295) à : Denfert-Rochereau (141)
   Départ de : Gare d'Austerlitz (295)
   Correspondance à : Gare d'Austerlitz (295) vers : Gare d'Austerlitz (123)
   3 stations
   Correspondance à : Place d'Italie (126) vers : Place d'Italie (145)
   4 stations
   Arrivée à : Denfert-Rochereau (141)
   ```

   Vous pouvez modifier la procédure `BellmanFord(G : Graphe ; Source : Noeud)` pour répondre à cette question.

5 Algorithme de Dijkstra

L'algorithme de Dijkstra calcule également le plus court chemin d'un nœud source vers tous les nœuds du graphe. Il utilise une **file de priorité** `Attente`, et peut s'écrire de la façon suivante :

```
Distance(1..n) := (others => Float'Last);
Distance(Source) := 0;
Attente := { Source };
tant que Attente non vide faire
  Noeud u := ElementDeDistanceMinimum(Attente);
  Supprimer(u,Attente);
  pour chaque arc (u,v) du graphe faire
    si Distance(u) + Cout(u,v) < Distance(v) alors
      Distance(v) := Distance(u) + Cout(u,v);
      MettreAJourOuInserer(v,Attente);
    fin si
  fin pour
fin tant que
```

Vous n'êtes pas obligés de re-programmer vous même une file de priorité mais pouvez utiliser les files disponibles avec Ada.

Questions

1. Implémenter en Ada, dans un paquetage `Dijkstra`, la procédure `Dijkstra(G : Graphe ; Source : Noeud)`, qui calcule le coût du plus court chemin du nœud `Source` vers tous les autres nœuds du graphe `G` selon l'algorithme de Dijkstra.
2. Implémenter, dans le même paquetage, une procédure `AffichageDijkstra(G : Graphe ; Source : Noeud ; Destination : Noeud)` qui affiche le détail du plus court chemin allant du nœud `Source` au nœud `Destination`. Ne seront affichées que les stations en correspondance (c'est-à-dire celles où l'on change de ligne).

6 Documents à rendre

Vous devez rendre via Teide, pour le vendredi 16 mai 2014 à 23h59 dernier délai, une archive *.tgz* comportant :

1. votre implémentation en Ada des structures de données permettant de représenter des graphes orientés à coûts positifs ou nuls (voir section 3) ;
2. vos implémentations en Ada des différents algorithmes demandés (voir sections 4 à 5) ;
3. un fichier *main.adb* permettant un test des différentes fonctions requises.
4. un rapport **au format PDF**, **de 2 pages maximum** (soit 1 feuille recto-verso) expliquant les choix d'implémentation réalisés.