AMS Algorithmique Avancée et Graphes

CERI - Licence 2 Informatique
S. Gueye

Semestre 3

1 Données

Vous diposez de deux fichiers csv contenant des données relatives à une partie du réseau routier réel d'Avignon et alentours : nodes.csv et arcs.csv. Ces fichiers sont une représentation sous forme de graphe du réseau.

1.1 Noeuds

"nodes.csv" contient un ensemble de noeuds du réseau correspondant à des localisations géographiques précises. A chaque noeud est associé les attributs suivants.

- Un identifiant(id).
- Une latitude (y).
- Une longitude (x).
- Le nombre de routes passant par ce noeud (*street_count*).

Le fichier des noeuds contient d'autres attributs. Nous ne retiendrons dans cette activité que ceux ci-dessus.

1.2 Routes

Chaque ligne du fichier des arcs est un couple de noeuds représentant un segment de route. Les attributs ci-dessous sont associés à un arc :

- osmid : un identifiant de l'arc.
- oneway : un booléen indiquant si la route est à sens unique ou non.
- lanes : le nombre de voies de la route.
- name : le nom de la route.
- highway : le type de route (autoroute, résidentiel, nationale (primary), départementale (secondary), etc.)
- reversed : dans le cas où la route est à sens unique (oneway = true), il indique quel est ce sens. Si "oneway = true" et "reversed = false" alors le sens est donné en lisant le couple de noeuds de gauche à droite (sinon de droite à gauche). Si "oneway = false" alors "reversed" n'a pas d'importance.
- length : longueur en métres de la route.
- geometry : suite de points constituant la route.
- maxspeed : vitesse maximale sur la route (en km/h)
- *speed_kph* : si la vitesse maximale n'a pas été indiquée, cette valeur est une moyenne des vitesses maximales autorisées sur les routes de ce type, ou la moyenne des vitesses de tous les arcs du réseau.

— *travel_time* : temps de trajet sur la route en supposant un trajet à la vitesse maximale autorisée, ou à la vitesse *speed_kph* .

Notez que les attributs décrits ci-dessus ne sont pas tous renseignés pour tous les noeuds, et tous les arcs. Aussi, un couple (a, b) d'une route à sens unique correspond à un seul arc allant du sommet origine a vers le sommet destination b. Une route à double sens (oneway = false) correspondra par contre à deux arcs (a, b) et (b, a) indiquant qu'on peut aller de a vers b, et de b vers a.

2 Objectifs

L'objectif de cette Activité de Mise en Situation (AMS) sera de stocker les données de ces fichiers dans des structures de données appropriées sur lesquelles des algorithmes les exploitant devront être développés. Les indications données vous donnent une architecture **partielle** de ces structures. C'est à vous de les compléter en fonction des questions posées.

3 Structures de données

3.1 Tâche 1

Un graphe est par définition un couple d'ensembles (V, E) où V est l'ensemble des noeuds, et E l'ensemble des arcs. On peut implémenter en C++ un arc (o, d) par la classe ci-dessous.

- class arc
  {
  string id ;
  string o ;
  string d ;
  (autres attributs de l'arc)
  ... ;
  public :
  arc(string id) ;
  ∼arc() ;
  void affiche() ;
  ....
  } ;

"id " sera le nom de la route. Si un arc dans le fichier n'a pas de nom, vous lui donnerez comme nom la chaîne de caractères correspondant au couple de noeuds constituant l'arc. Pour tous les autres attributs non connus, vous choisirez une valeur par défaut indiquant que le champ est vide. Si plusieurs routes ont même nom, disons "name", le premier arc associé aura comme identifiant id = "*name*", le second sera "*name*2", le troisième "*name*3", etc.

Avec cette classe, nous pouvons alors définir un ensemble E d'arcs. Il vous faut décider comment représenter cet ensemble. S'agira-t-il d'une liste chaînée d'arcs, d'un arbre binaire de recherche d'arcs, d'une table de hachage d'arcs ? Vous pouvez au choix :

1. Utiliser les classes adéquates de la librairie STL (voir sur e-uapv "Introduction à la librairie STL").
2. Coder votre propre classe E implémentant un ensemble d'arcs.

Choisir une structure de données pour E. Dans E, on doit pouvoir : *ajouter des arcs, rechercher un arc à partir de son identifiant, afficher le contenu de l'ensemble.* Votre choix doit être guidé, et justifié,

par le fait que l'on doit pouvoir retrouver "rapidement" les informations de n'importe quelle route à partir de son identifiant.

3.2 Tâche 2

Un noeud sera représenté par la classe suivante :

- class noeud
  {
  string id ;
  (attributs du noeuds)
  (Type à proposer) *arcs_sortants* ;
  (Type à proposer) *arcs_entrants* ;
  ... ;
  public :
  noeud(string id) ;
  ∼noeud() ;
  void affiche() ;
  ....
  } ;

Cette classe contient deux attributs spéciaux : *arcs_entrants*, *arcs_sortants*. Il s'agit de la liste des arcs dont le noeud en question est l'origine (*arcs_sortants*), et la liste dont c'est la destination (*arcs_entrants*). Proposer une structure de données pour ces attributs.

A partir d'un noeud, on définit un ensemble V de noeuds. Comme pour les arcs, il vous faut décider de la façon de représenter cet ensemble : liste chaînée de noeuds, arbre binaire de recherche de noeuds, table de hachage ? Dans V on doit pouvoir : *ajouter des noeuds, rechercher un noeud à partir de son identifiant, et afficher le contenu de l'ensemble.*

Vous avez les mêmes choix d'implémentation de cet ensemble que dans la tâche précédente. Le choix doit être justifié par le fait que l'on doit pouvoir retrouver "rapidement" les informations de n'importe quel noeud à partir de son identifiant.

3.3 Tâche 3

Un graphe étant défini par V et E, on peut l'implémenter maintenant par la classe.

- class graphe
  {
  V lesnoeuds ;
  E lesarcs ;
  int nombredenoeuds ;
  int nombredarcs ;
  public :
  graphe(fichierarcs,fichiernoeuds) ;
  ∼graphe() ;
  void affiche() ;
  ...
  } ;

— Ajoutez à la classe *graphe* la méthode : "*int lecture_arcs*($nomfichier$)". Elle lit le fichier ("arcs.csv") contenant les arcs, ligne par ligne, et insére dans "lesarcs" les arcs rencontrés. La méthode renvoie le nombre d'arcs.
— Ajoutez à la classe *graphe* la méthode : "*int lecture_noeuds*($nomfichier$)". Elle lit le fichier ("noeuds.csv") contenant les arcs, ligne par ligne, et insére dans "lesnoeuds" les sommets rencontrés. A la création d'un noeud, les attributs *arcs_sortants* et *arcs_entrants* seront initialement à NULL. La méthode renvoie le nombre de noeuds.
— Ajoutez à la classe *graphe* la méthode "*void liste_incidence*()". Elle met à jour, pour chaque noeud de l'ensemble "lesnoeuds", les attributs *arcs_sortants* et *arcs_entrants* en parcourant "lesarcs".

Le constructeur "graphe(fichierarcs,fichiernoeuds)" construit les ensembles "lesnoeuds" et "lesarcs" en appelant les méthodes "*int lecture_arcs*($nomfichier$)" et "*int lecture_noeuds*($nomfichier$)".

4 Algorithmes de parcours

4.1 Tâche 4

Le degré d'un noeud est le nombre d'arcs contenant ce noeud. Ajoutez dans la classe "graphe" une méthode "void degre(n)" affichant les n premiers noeuds de plus fort degré (où $1 \leq n \leq nombredenoeuds$). Vous donnerez, et justifierez la complexité de l'algorithme proposé.

4.2 Tâche 5

On appelle "chemin" une suite d'arcs e_1,e_2,...,e_q dans laquelle le sommet destination de e_i correspond au sommet origine de e_{i+1} pour tout $i = 1, 2, ..., q-1$. On souhaite ajouter une méthode "bool chemin(o,d)" permettant de savoir s'il existe ou non un chemin permettant d'aller du noeud d'identifiant o au noeud d'identifiant d. Pour ce faire, il faut être capable de parcourir le graphe à partir d'un noeud donné (se réfèrer cours graphes L1). Un parcours en profondeur d'un graphe, à partir d'un noeud s, peut se faire de la manière suivante.

```
Algorithme : Parcours(s)
pile : une pile de noeuds intialement vide ;
pour chaque noeud v de V faire
    v.visite = faux ;
fin
pile.empiler(s);
tant que (pile.estVide() == faux) faire
    s = pile.depiler();
    si s.visite == faux alors
        s.visite = vrai;
        pour i allant de 1 à n faire
            si il existe un sommet u tel que l'arc (s, u) et (u.visite == faux) alors
                pile.empiler(u);
            fin
        fin
    fin
fin
```

L'algorithme suppose l'existence d'un attribut (à ajouter) "visite" à la classe noeud.

En vous basant sur ce pseudo-code, codez une méthode "int chemin(string o,string d)". Elle renverra le nombre d'arcs du chemin s'il existe, ou 0 sinon. Elle affichera aussi les noms des routes utilisées dans le chemin.

4.3 Tâche 6

Toujours en vous basant sur le pseudo-code, mais en remplaçant la pile par une autre structure, codez une méthode "int pluscourtchemin(o,d)" cherchant le plus court chemin **en nombre d'arcs** du graphe de o à d. La méthode renverra le nombre d'arcs de ce chemin si il existe, ou 0 sinon. De même que précédemment, les noms de routes empruntées devront être affichés.

4.4 Tâche 7

Un utilisateur lambda de votre code n'a aucune connaissance de l'identifiant des noeuds, par contre il connait les noms des routes. Ajoutez une méthode "int itineraire(string origine,string destination)" donnant l'itinéraire le plus court, en nombre de rues, pour se rendre de la rue "origine" à la rue "destination".

5 Modalités d'évaluation

- L'AMS fait l'objet d'une note à part, distincte de la note TP.
- Travail réalisable en groupes de 2 maximum.
- L'évaluation se fera à l'oral après le rendu du code sur le site.
- Pour un groupe deux personnes, les questions sur n'importe quelle partie du code pourront être posées à une personne quelconque. Chacun devra donc maîtriser le contenu du rendu. Selon les réponses apportées, les notes pourront être différentes.
- Tous les codes devront être commentés avec justification des choix effectués.