Projet sur les graphes

Master Estel - 2023-2024

Valérie Roy, Mines Paris PSL, valerie.roy@minesparis.psl.eu

Introduction

Un graphe orienté est composé de sommets et d'arêtes étiquetées orientées connectant ces sommets.

Les graphes apparaissent très fréquemment en informatique (pour représenter des réseaux aériens, routiers, ferrés...). Sur ces représentations, nous sommes amenés à nous poser des questions très générales:

- Quels sont toutes les villes atteignables à partir d'une ville donnée quelque soit le nombre de changements ?
- Quel est le chemin *le plus court* entre deux villes quelque soit le nombre de changements; si on a le graphe des temps ça sera le temps le plus court, si on a le graphe des prix ça sera le prix le plus bas.
- Quels sont tous les chemins les plus courts entre toutes les villes ?

L'étude des graphes a donné lieu à une théorie informatique et des algorithmes ont été proposés pour résoudre ces problèmes: des parcours de graphes (*depth first search* et *breadth first search*), des calculs de plus courts chemins (*Dijkstra*, *Floyd-Warshall*)...

Ce projet vous propose d'implémenter en `c++` quelques algorithmes classiques de graphes sachant que, dans la *vraie vie*, vous utiliseriez une librairie existante éprouvée.

Mise en garde

Le sujet des graphes est très classique et il est très discuté sur Internet. Vous pouvez y trouver des tas de codes qui sont pour la plupart très maladroits. Ne vous laissez pas influencer par ces codes. Implémentez vos idées: c'est comme cela que vous progresserez.

Les exigences

- réalisez le sujet seul
- implémentez des `class`, faites attention à ce que vous mettez en `public` et en `private`
- utilisez la librairie standard dès que vous avez besoin d'un conteneur (`std::vector`, `std::unordered_map` = dictionnaire) ou d'une fonctionnalité particulière (sauf algorithme de graphes)
- découpez votre code en fichiers d'entête et fichiers d'implémentation; choisissez des noms de fichiers expressifs
- écrivez du code lisible, correctement indenté et intelligemment commenté
- choisissez avec soin vos noms de classes, attributs, méthodes, variables, fonctions (ils doivent être expressifs)
- proposer des fonctions pour tester vos algorithmes
- rendez votre code sous `github`

Contraintes sur vos graphes

Les graphes sont dirigés, leurs sommets sont étiquetés par des chaînes de caractères, leurs arêtes sont valuées (étiquetées) par des nombre réels positifs.

Terminologie sur les graphes

Un sommet s_2 sera dans la *liste* (mais il ne faut pas utiliser de `std::list` `c++`) d'adjacence du sommet s_1 si il existe une arête qui va du sommet s_1 au sommet s_2.

Il y a un `chemin` entre le sommet s_1 et le sommet s_n si il existe une suite (non nulle) d'arêtes pour aller de s_1 à s_n $s_1 \rightarrow s_2 \rightarrow s_3 ... \rightarrow s_n$. La valeur du chemin est la somme des valeurs des arêtes du chemin.

Représentations des graphes en mémoire

Il y a deux manières pour représenter un graphe dans la mémoire d'un programme:

par listes d'adjacences:

- un graphe est représenté par ses sommets (réfléchissez à la structure de données pour stocker les sommets: vecteur, ensemble, dictionnaire ?)
- chaque sommet connaît ses sommets adjacents i.e. ceux vers qui il peut aller (réfléchissez à la structure de données pour stocker les sommets adjacents: vecteur, ensemble, dictionnaire ?)

par matrice d'adjacence:

- un graphe est représenté par une matrice de taille $(n \times n)$ avec n est le nombre de sommets
  L'élément à la position $[s_i, s_j]$ dans cette matrice est la valeur de l'arête qui va du sommet s_i au sommet s_j.
  Si il n'y a pas d'arête entre deux sommets: ***pour les débutants*** mettez 0 et ***pour les avancés*** trouvez une manière adéquate pour représenter cette absence (e.g. il existe une valeur `nan` réelle `std::isnan("")` de la librairie `<cmath>`).

Quelle structure de graphe choisir ?
En fait, suivant la représentation choisie, la place mémoire occupée par le graphe, les algorithmes et leur complexité seront différents. Donc ça dépend.

Si votre graphe n'a que très peu d'arêtes la matrice d'adjacence sera toute vide (creuse). Si votre graphe a une arête entre chacun de ses sommets (le graphe est complet) les deux représentations vont prendre la même place.

Si le graphe est représenté par une matrice d'adjacence, pour parcourir tous les sommets adjacents à un sommet donné, on va parcourir n éléments (une ligne de la matrice) alors que si le graphe est représenté par des listes d'adjacences on parcourra uniquement les arêtes partant de ce sommet.

Première question

- implémentez la représentation par listes d'adjacences d'un graphe (vous aurez besoin de définir plusieurs classes);

- ***pour les avancé***: implémentez aussi la représentation par matrice d'adjacence d'un graphe (faites très simple, représentez les sommets par les indices de la matrice, faites une classe `Matrix` avec un `std::vector`);

Format textuel d'un graphe

Dans un fichier, un graphe est donné (par exemple) par la liste de ses arêtes: avec une arête par ligne représentée par le nom de son sommet de départ, le nom de son sommet d'arrivée et la valeur de l'arête (i.e. son étiquette).

Deuxième question

- implémenter le code pour lire un graphe contenu dans un fichier et en construire la représentation sous forme de liste d'adjacence;
- ***pour les avancé***: donnez la possibilité de passer le nom d'un fichier contenant un graphe en argument à votre exécutable; les fichiers contenant des graphes doivent être postfixés par `.graph` (vérifiez que le nom du fichier est bien formé);

Parcours de graphes

Parcourir un graphe consiste à *passer* par tous ses sommets et toutes ses arêtes.

Le parcours en profondeur depuis le sommet s_i d'un graphe, s'exprime très simplement d'une manière récursive. Vous lancez la visite du sommet s_i, dans celle-ci vous commencez par marquer le sommet s_i comme ayant été visité. Ensuite vous considérez chacun de ses sommets adjacents s_j et, si ce sommet s_j n'a pas encore été visité, vous lancez sa visite. Quand un sommet n'a plus de sommets non visités dans sa liste d'adjacence, sa visite s'arête. Quand tous les sommets atteignables à partir du sommet s_i ont été visités, le parcours en profondeur depuis le sommet s_i du graphe termine. Pour parcourir tous les sommets du graphe, vous devrez lancer la visite de tous les sommets qui n'ont pas encore été visités.

Troisième question

- implémentez le parcours en profondeur de manière récursive dans le graphe sous la forme de liste d'adjacence;
- ***pour les avancé***: faites le aussi pour le graphe sous la forme de matrice d'adjacence.
  La récursivité c'est quand une fonction ou une méthode s'appelle elle-même (sur un objet *plus petit* sinon vous bouclez à l'infini).
- affichez les sommets et les arêtes par lesquels vous passez

Si on réfléchit, pendant le parcours récursif, les sommets en cours de visite, sont *conservés* grâce à la pile d'exécution du programme (pile: premier entré = dernier sorti) et c'est cela qui réalise un parcours en profondeur. Mais alors si au lieu de s'appuyer sur la pile d'exécution du programme, on utilise un conteneur avec un comportement de pile, on peut implémenter un parcours en profondeur qui soit itératif et plus récursif.

Quatrième question

- implémentez le parcours en profondeur de manière itérative dans vos représentations de graphes;
- affichez les sommets et les arêtes par lesquels vous passez;

Maintenant, si on remplace ce mécanisme de pile par un mécanisme de file (file: premier entré = premier sorti), l'algorithme va changer et que devient-il ? Un parcours en largeur d'abord.

Cinquième question

- implémentez le parcours en largeur dans vos représentations de graphes;
- affichez les sommets et les arêtes par lesquels vous passez

Problèmes de plus courts chemins

Sixième question

- implémentez l'Algorithme de Dijkstra qui calcule le plus court chemin entre deux sommets d'un graphe;
- Nous n'allons pas donner ici une explication de cet algorithme, vous devez vous référerez à des explications sur Internet, par exemple, : https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra;

Vous le ferez sur la représentation du graphe que vous voulez.

- affichez les chemins trouvés pour plusieurs paires de sommets

Septième question

- implémentez l'Algorithme de Floyd-Warshall *https://fr.wikipedia.org/wiki/Algorithme_de_Floyd-Warshall* qui donne le plus court chemin entre tous les sommets d'un graphe (qui est représenté par une matrice d'adjacence);

- affichez tous les chemins trouvés pour un graphe;

(optionnelle) Représentation graphique des graphes

Les graphes sont des modèles intrinsèquement visuels aussi est-il intéressant de les dessiner.

Huitième question

- (optionnelle) Utilisez une librairie, par exemple `graphviz`, pour proposer une représentation graphique de vos graphes.