Projet d'informatique : 1A PET

Ce projet sera réalisé en binôme, sur les 5 dernières séances d'informatique ET en dehors des séances d'informatique. Pour la première séance, vous devez avoir lu le sujet et formé les binômes.
Attention : La réussite de ce projet exige du travail en dehors des séances.

Première séance : analyse du problème.
Cette séance permet de répondre à vos questions sur les algorithmes et les structures de données. A la fin de cette séance, vous devez avoir une vision claire des grandes étapes de votre programme et vous ferez un document décrivant :

- les types et structures de données utilisées, en explicitant le rôle de chaque élément des structures
- les prototypes de fonctions, en explicitant :
  - le rôle exact de la fonction
  - le rôle de chaque paramètre et son mode de passage (par valeur ou par adresse)
  - l'algorithme ou les grandes étapes permettant de réaliser la fonction. Précisez uniquement les points qui peuvent être délicats à comprendre et/ou programmer.
- les tests prévus
  - tests unitaires : tests des fonctions précédentes individuellement. Par exemple, il faut tester la fonction de lecture des données du graphe avant même d'essayer de trouver le plus court chemin.
  - Tests d'intégration : quels sont les tests que vous allez faire pour prouver que l'application fonctionne, sur quels exemples.
- les modules (couples de fichiers .c et .h), les fonctions contenues dans chaque fichier
- la répartition du travail entre les 2 membres du binôme : quelles sont les fonctions qui seront réalisées par chaque membre du binôme et pour quelle séance.
- Le planning de réalisation du projet

Ce document est essentiel et doit permettre ensuite de coder rapidement votre application en vous répartissant les tâches.

Séances suivantes :
Ces séances servent à débugger et à poser des questions aux enseignants. Si la répartition est bien faite, chaque membre du binôme peut avancer sa partie de travail.

Dernière séance :
Il faut finir de réaliser les tests. Vous copierez l'ensemble des fichiers sources dans le répertoire /users/phelma/phelma2015/tdinfo/mon-login/seance14.
Vous ferez un **rapport** contenant l'analyse initiale, les tests prévus, la répartition et le planning effectif. Il indiquera l'état final du programme réalisé, ce qui fonctionne, ce qui ne fonctionne pas..
Le rapport fera au plus 20 pages, non compris le code, dont voici un plan indicatif:

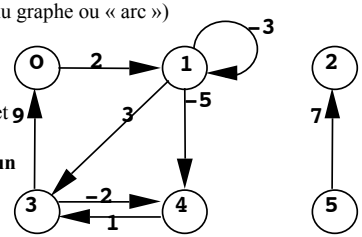
1- Intro
2- Spécifications
  2.1 Données : description des structures de données
  2.2 Fonctions : Entete et rôle des fonctions **essentielles**
  2.3 Tests : quels sont les tests prévus
  2.4 Répartition du travail et planning prévu : qui fait quoi et quand ?
3- Implémentation
  3.1 Etat du logiciel : ce qui fonctionne, ce qui ne fonctionne pas
  3.2 Tests effectués
  3.3 Exemple d'exécution
  3.4 Les optimisations et les extensions réalisées
4- Suivi
  4.1 Problèmes rencontrés
  4.2 Planning effectif
  4.3 Qu'avons nous appris et que faudrait il de plus?
  4.4 Suggestion d'améliorations du projet
5- Conclusion

Projet 2016 : Plus court chemin dans un graphe

L'objectif du projet est de calculer le meilleur itinéraire, i.e. le plus court chemin entre 2 stations de métro parisien. Un fichier vous donne l'ensemble des stations et des relations entre 2 stations. Le métro se représente facilement par un graphe.

Définition & Terminologie
Un Graphe est défini par un couple G[X,A] où X est un ensemble de nœuds ou sommets et A est l'ensemble des paires de sommets reliés entre eux (arêtes du graphe ou « arc »)

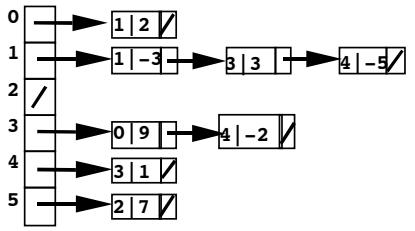
- Arc = arête orientée
- **chemin = séquence d'arcs menant d'un sommet i à un sommet j**
- circuit = chemin dont les sommets de départ et d'arrivée sont identiques
- **valuation, coût = valeur numérique associée à un arc ou à un sommet**
- degré d'un sommet = nombre d'arêtes ayant ce sommet pour extrémité
- voisins : les voisins des sommets sont ceux qui sont reliés à ce sommet par un arc.



Représentation
On peut représenter un graphe en numérotant les sommets et les arcs par des entiers. Un arc est un triplet "sommet de départ","sommet d'arrivée","coût de l'arc".

Matrice d'incidence sommets-arcs par liste chaînées
On utilise un tableau de listes chaînées pour représenter les arcs entre sommets. L'indice du tableau est le numéro du sommet et donne accès à la liste des arcs qui partent de ce sommet et qui mènent à un successeur. Chaque élément de la liste contient le numéro du successeur (sommet d'arrivée de l'arc) et sa valeur (son coût).

Exemple : tableau de listes représentant le graphe ci-dessus. Les sommets sont numérotés de 0 à 5. Le tableau représentant le graphe contient les listes des arcs, chaque arc contenant le sommet d'arrivée et le coût de cet arc.



Plus court chemin par Bellman

On considère un graphe orienté **G=(X, A).** Chaque arc $\mathbf{a_i}$ est muni d'un coût $\mathbf{p_i}$. Un chemin **C=< a_1, a_2, ..., a_n >** possède un coût qui est la somme des coûts des arcs qui constituent le chemin. Le plus court chemin d'un sommet **d** à un sommet **a** est le chemin de coût minimum qui va de **d** à **a**. L'algorithme de Bellman-Ford est le seul à s'appliquer dans le cas d'arcs à valeur négative. Attention cependant aux circuits de valeur négative, car il n'existe pas de solutions dans ce cas.

Le principe de l'algorithme de Bellman est le suivant : si C est un plus court chemin entre les sommets **s** et **t** et si **u** appartient à ce plus court chemin, alors les sous-chemins de **s** à **u** et de **u** à **t** sont également des plus courts chemins La version basique de cet algorithme est la suivante :

Algorithme

Recherche du plus court chemin dans le graphe G à partir du sommet s
Mettre les poids de tous les sommets à +infini
Mettre le poids du sommet initial à 0 ;
pour i=1 à Nombre de sommets -1 ou stabilité faire
/* Itération i*/
| pour chaque arc (u, v) du graphe faire
| | si poids(u) + poids(arc(u, v)) < poids(v) alors
| | /*le chemin passant par l'arc u,v est plus court pour rejoindre v que les précédents chemins trouvés, mettre à jour le poids du sommet v car un chemin plus court est trouvé */
| | poids(v)= poids(u) + poids(arc(u, v))
pour chaque arc (u, v) du graphe faire
| si poids(u) + poids(arc(u, v)) <poids(v) alors
| il y a un circuit négatif dans le graphe, solution impossible
Ici, On a trouvé la solution

Remarque

- Pour retrouver le chemin, il faut aussi stocker les pères des nœuds lorsque la mise à jour du poids du sommet est réalisée. Une solution consiste alors à ajouter un champ « pere » à la structure représentant un sommet, qui indique quel est le sommet précédent sur le plus court chemin. Pour trouver le chemin final, Il faut partir du point d'arrivée pour remonter de père en père au sommet de départ.

Optimisation

On peut optimiser en ne traitant à l'itération i que les arcs dont le coût du sommet de départ a été modifié à l'itération précédente, plutôt que de traiter tous les arcs à chaque étape. On utilise alors une file de sommets dans laquelle on enfile les sommets dont on modifie les couts à chaque itération. Cette version optimisée ne fonctionne cependant que pour les graphes ne comportant pas de cycles négatifs (ce qui est notre cas). L'algorithme s'écrit alors :

```
Enfiler le sommet de depart
nbiteration=0
   Tant que la file n'est pas vide
        /* Test valide pour les graphes sans cycles négatifs*/
   Faire
      Iteration++
      Defiler le sommet u
      Pour tous les arcs(u,v) partant de ce sommet u
      Faire
        si poids(u) + poids(arc(u, v)) < poids(v) alors
                /*le chemin passant par l'arc u,v est plus court pour
                rejoindre v que les précédents chemins trouvés mettre à
                jour le poids du sommet v */
            poids(v)= poids(u) + poids(arc(u, v))
```



```
si le sommet v n'est pas déjà dans la file
alors
 enfiler ce sommet
```

Remarque : pour accélérer la recherche dans la file, on peut utiliser un champ spécifique dans la structure sommet qui indiquera si le sommet est déjà ou non dans la file, ce qui évite de parcourir cette file.

Format des fichiers de données

Pour tester votre algorithme, vous disposez de trois fichiers de données avec des graphes de différente taille : un petit (graphe1_2012.csv), un très grand (graphe3_2012.csv), et un troisième qui encode le réseau métro/rer/tram parisien (metro.csv),. Dans ces fichiers, les coordonnées des sommets sont des coordonnées GPS comprises dans le rectangle (2.2 ; 48.2) (3.0 ; 49.1).

Le format de ces fichiers est le suivant :

Première ligne : deux entiers ; nombre de sommets (X) nombre d'arcs (Y)
Deuxième ligne : la chaîne de caractère "Sommets du graphe"
X lignes dont le format est le suivant :

- un entier : numéro de la station ;
- deux réels indiquant la latitude et la longitude
- une chaine de caractères (sans séparateurs) contenant le « nom de la ligne » (par exemple M1, M3bis, T3, A1 pour le fichier métro parisien)
- et une chaîne de caractères contenant le nom du nœud (qui peut contenir des séparateurs, par exemple des espaces).

1 ligne : la chaîne de caractère "arc du graphe : départ arrivée valeur "
Y lignes dont le format est le suivant :

- un entier : numéro du sommet de départ
- un entier : numéro du sommet d'arrivée
- un réel : valeur ou cout de l'arc

Remarque importante pour la lecture de ces fichiers en C:

La lecture des lignes contenant les sommets du graphe (les X lignes) doit se faire en lisant d'abord un entier, deux réels et une chaine, puis une chaine de caractères qui peut contenir des espaces. Le plus simple est de lire les entiers et les réels ainsi que la ligne de métro avec la fonction fscanf puis le nom de station (chaine avec séparateurs) avec la fonction fgets :

```
fscanf(f,"%d %lf %lf %s", &(numero), &(lat), &(longi), line);
fgets(mot,511,f);
if (mot[strlen(mot)-1]<32) mot[strlen(mot)-1]=0;
```

numéro contient alors l'entier, lat et longi la position, line le nom de la ligne et mot le nom du sommet.

Fichiers metro.csv: le métro, le tramway et le RER de Paris

Cet exemple est un graphe avec environ 700 sommets et 1900 arcs. Chaque station est un sommet du graphe. Attention, il y a une station différente par ligne, même si deux lignes ont la même station. La notion de correspondance est prise en compte : si deux lignes 1 et 2 se croisent avec correspondance, deux sommets différents existent pour cette station de correspondance, un sur la ligne 1 et un autre sur la ligne 2. La correspondance est modélisée un arc de coût pré-établi, de valeur 360 entre ces deux sommets. Le fichier inclut aussi les correspondances à pied (exemple : entre Gare du Nord et La chapelle). Dans ce cas, le coût pré-établi est de 600. Le coût des autres arcs dépend du moyen de transport (métro, tramway et RER). Pour tous les arcs, on est assuré que le coût est toujours supérieur à :

*cout_min(s, a) = 40 + distance_euclidienne(s,a)*3900.*

(où 40 estime un temps d'arrêt en station et 1/3900 minore la vitesse du moyen de transport le plus rapide, c'est à dire du RER).

Cette remarque vaut incidemment pour les trois fichiers de donnée.

Remarques :

- Chaque arc est orienté, donc il y a par exemple un arc entre le sommet 1 et 12 et un arc entre le sommet 12 et 1, et ces deux arcs ne sont pas identiques.
- Plusieurs stations portent le même nom : lorsqu'une station se trouve sur 2 lignes différentes, elle est alors considérée comme 2 sommets distincts du graphe, mais ces 2 sommets portent le même nom. Par exemple, *Montparnasse Bienvenue* est une correspondance pour 4 lignes : les lignes 4, 6, 12 et 13. On trouve donc 4 sommets pour *Montparnasse bienvenue*, les sommets 98, 138, 300, 361 correspondant à ces lignes, les 16 arcs (*Saint Placide-Montparnasse-Vavin*, *Pasteur-Montparnasse-Quinet*, *Notre Dame des Champs-Montparnasse-Falguière*, *Duroc-Montparnasse-Gaîté*), ainsi que des arcs modélisant la correspondance et reliant les différents nœuds *Montparnasse bienvenue.*

Travail demandé

Faire une application qui demande les numéros des stations de départ et d'arrivée et qui calcule et affiche le plus court chemin pour ce voyage.

Prévoyez un développement incrémental en testant toutes vos fonctions au fur et à mesure.

Ayez recours aux 3 fichiers de données pour tester votre logiciel.

Dans un premier temps, vous désignerez les stations en utilisant leur numéro (un entier). Vous validerez le programme réalisé sur les graphes simples avant de le tester sur le métro. Ensuite, vous réaliserez une (ou plusieurs) fonctions permettant de rentrer le nom (et non le numéro) de la station. Il faut faire alors correspondre le nom de la station (la chaine de caractères) à **tous** les sommets de même nom.

Remarque : pour les stations de départ et d'arrivée recherchée par leur nom, une station de même nom sur des lignes différentes est un sommet différent. Pour partir de Montparnasse bienvenue, on peut utiliser les sommets 98, 138, 300, 361. De même pour l'arrivée. Et vous ne savez pas à priori quelle est la ligne que vous allez prendre au départ et celle par laquelle vous arrivez. Il faut donc trouver une solution simple pour considérer tous les sommets de départ correspondant au nom demandé.

Ne vous perdez pas et Bon voyage !!!

Quelques éléments pris en compte dans la notation

Rapport : 4 points
Lecture du graphe : 3 points
Calcul du plus court chemin: 4 points
Affichage du chemin sommet par sommet : 3 points
Gestion des noms des stations : 3 points
Tests : 2 points
Code commenté : 1 point
Extensions : graphisme, optimisation, etc : bonus
Plagiat interne ou externe à phelma : 0/20