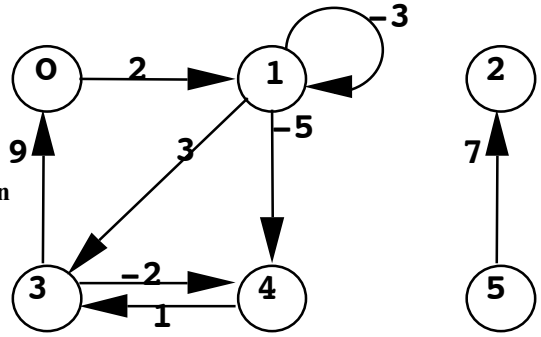
PROJET D'INFORMATIQUE 1A PET : PLUS COURT CHEMIN

Ce projet sera réalisé en binôme, sur les 4 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.

L'objectif du projet est de calculer le meilleur itinéraire, i.e. le plus court chemin entre 2 villes ou 2 stations de métro parisien.
Le réseau routier ou le métro se représente facilement par un graphe, chaque sommet étant une intersection de routes ou une station de métro, les arcs représentant les routes et la distance, ou une ligne de métro et la distance entre 2 stations.

1. GRAPHES : DEFINITION & 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.



1.1. Représentation des sommets et des arcs

On peut représenter un sommet par une structure contenant un sommet, c'est à dire son nom, sa latitude, sa longitude (pour les dessiner graphiquement, la ligne de métro à laquelle ce sommet appartient si besoin et la liste de ses successeurs possibles, c'est à dire la liste des intersections de routes pour le réseau routier ou les stations suivantes pour le métro.

Les types de données utiles ressembleront à :
Les sommets : T_SOMMET est une structure

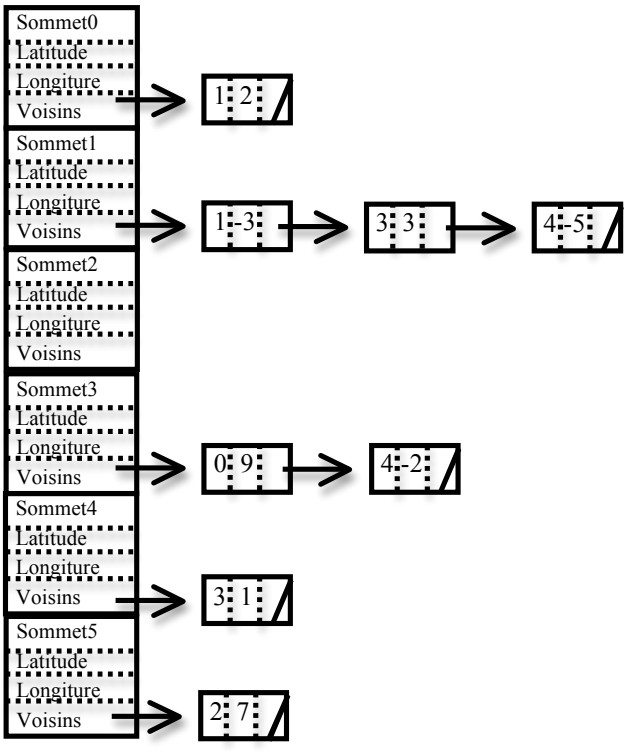
```
typedef struct {
       char* nom;
       double x,y ;
       L_ARC voisins;} T_SOMMET ;
```

Les arcs: un arc est un structure

```
typedef struct {
       int arrivee;
       double cout } T_ARC ;
```

Les listes de successeurs : une liste classique avec un suivant, la valeur est un arc.

```
typedef struct lsucc {
   T_ARC val;
   struct lsucc* suiv ; }* L_ARC;
```



1.2. Représentation du graphe

Le graphe sera représenté par un tableau de sommets.et le nombre de sommets.

2. PLUS COURT CHEMIN DANS UN GRAPHE

Dans un problème de plus court chemin, on considère un graphe orienté **G=(X, A). X** est l'ensemble des sommets, **A** celui des arcs. Chaque arc $\mathbf{a_i}$ est muni d'un coût $\mathbf{p_i}$. Un chemin $\mathbf{C=< a_1, a_2, ..., a_n >}$ 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**. Il existe plusieurs algorithmes de calcul du plus court chemin :

- L'algorithme de parcours en largeur d'abord permet de trouver le plus court chemin dans un graphe.
- L'algorithme de Dijkstra, dans le cas d'arcs à valuation positive, est celui utilisé dans le routage des réseaux IP/OSPF.
- L'algorithme de Bellman-Ford est le seul à s'appliquer dans le cas d'arc à valeur négative. Attention cependant aux circuits de valeur négative, car il n'existe pas de solutions dans ce cas.
- L'algorithme A* ou Astar qui utilise une heuristique pour trouver rapidement une solution. Il ne s'applique qu'aux graphes dont les arcs sont à valuation positive. De plus, il nécessite qu'il soit possible d'estimer la « distance » entre deux nœuds par une fonction heuristique.

2.1. Dijkstra

L'idée de l'algorithme de Dijkstra est que lorsqu'on a trouvé un plus court chemin entre le sommet de départ **d** et un sommet intermédiaire **k**, il faut mettre à jour les successeurs possibles de **k** que l'on peut atteindre. Lorsque l'on trouve le plus court chemin entre **d** et **k**, c'est aussi le plus court chemin entre le départ et chaque sommet sur le chemin entre **d** et **k**.

Tout au long du calcul, on va donc maintenir deux ensembles :

- **C**, l'ensemble des sommets du graphe qui restent à visiter ; initialement **C=X**
- **S**, l'ensemble des sommets du graphe pour lesquels on connaît déjà leur plus court chemin au point de départ; initialement, **S={}.**
- Le tableau pcc[i] contient la valeur du plus court chemin actuellement trouvé entre le sommet de départ d et le sommet i.
- Le tableau pere[i] contient l'indice du sommet précédent i dans le plus court chemin actuellement trouvé entre le sommet de départ d et le sommet i.

L'algorithme se termine bien évidemment lorsqu'on atteint le sommet voulu. Pour chaque sommet **s** dans **S**, on conservera le coût actuel du chemin entre **s** et **d** dans la variable **PCC.** On conservera aussi le sommet **pere** qui le précède dans ce plus court chemin. **Pere[i]** est l'indice du sommet qui est le prédécesseur de **i** dans le plus court chemin entre **d** et **a**. Ainsi, pour retrouver le plus court chemin, il suffira, en partant de **a**, de remonter de prédécesseur en prédécesseur jusqu'à **d** grâce au tableau pere.

2.2. Algorithme : trouver le plus court chemin entre d et a

```
1.  début
2.      pour tous les sommets i de G[X,A] faire  pcc[i] ← +∞;  pere[i]=-1 ;
3.      pcc[d] ← 0
4.      S ← {}
5.      C ← X
6.       faire
7.              Sélectionner le sommet j de C de plus petite valeur pcc[j]
8.              C ← C \ j                              // supprimer j de l'ensemble C
9.              S ← S ∪ j                              // ajouter j à l'ensemble S
10.             pour tous les sommets k  adjacents à j faire // les successeurs de j
11.                     si  pcc[k] > pcc[j] + c[j][k]          // c(j,k) est le coût pour aller de j à k
12.                     alors
13.                             pcc[k] = pcc[j] + c[j][k]; // Passer par j est plus court
14.                             pere[k]=j ;                //  pour aller de a en k
15.                     fin si
16.             fin pour
17.     tant que a∉ S   et pcc[j]!=+∞
18. fin
```

3. FORMAT DES FICHIERS DE DONNEES

Pour tester votre algorithme, vous disposez de plusieurs fichiers dont le format 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 :

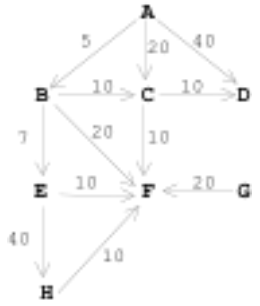
- un entier : numéro de la station ;
- deux réels indiquant la latitude et la longitude
- une chaine de caractères (sans séparateurs) contenant le « nom de la ligne » (par exemple M1, M3bis, T3, A1 pour le fichier métro parisien)
- une chaîne de caractères contenant le nom du nœud (qui peut contenir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

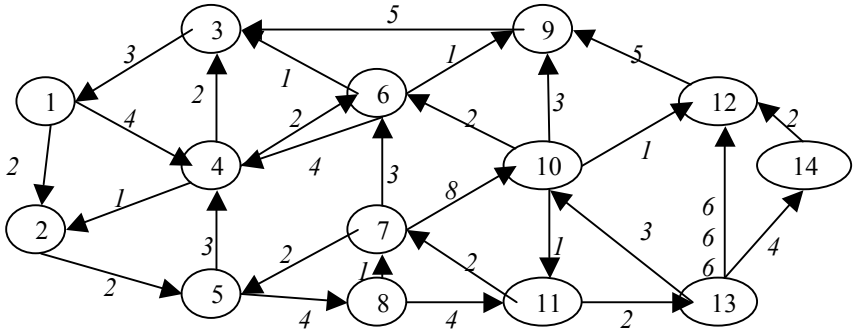
- un entier : numéro du sommet de départ
- un entier : numéro du sommet d'arrivée
- un réel : valeur ou cout de l'arc

Les fichiers contenant les graphes sont dans le répertoire `/users/prog1a/C/librairie/projets22018`..

- graphe1.txt et graphe2.txt contiennent des petits graphes ci dessous
- metroetu.csv contient le graphe représentant les lignes de metro et RER de Paris
- Les fichiers grapheNewYork.csv, grapheColorado.csv, grapheFloride.csv, grapheGrandLacs.csv, grapheUSAOuest.csv, grapheUSACentral.csv contiennent le réseau routier américain. **Ne pas recopier ces fichiers dans vos répertoires** car il sont très volumineux et comportent de 250000 à 14000000 sommets.



Graphe1.txt



Graphe2.txt

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` :

```
int numero ;
double lat,longi ;
char line[128] ;
char mot[512] ;

fscanf(f,"%d %lf %lf %s", &(numero), &(lat), &(longi), line);
/* numéro contient alors l'entier, lat et longi la position, line le nom de la ligne */
fgets(mot,511,f);
if (mot[strlen(mot)-1]<32) mot[strlen(mot)-1]=0;
/* mot contient le nom du sommet. */
```

Pour sauter les lignes de commentaires, on peut simplement utiliser la fonction fgets et ne pas utiliser la chaine de caractères lue dasn le fichier par

```
fgets(mot,511,f);
```

4. TRAVAIL DEMANDE

Le travail se déroulera en 2 étapes :

1. une première étape consiste à déterminer le plus court chemin sur le réseau routier
2. une deuxième étape consiste à déterminer le plus court chemin sur le métro, qui comporte des correspondances

4.1. Réseau routier américain

L'objectif est donc de construire le graphe à partir du fichier représentant le réseau routier, puis de demander à l'utilisateur un numéro de sommet de départ et un numéro de sommet d'arrivée, de calculer la valeur du plus court chemin et d'afficher le chemin trouvé ensuite.
Il y a bien sur des sens uniques et il n'existe pas toujours d'arc dans les 2 sens entre 2 sommets.
La particularité de cette étape est donc uniquement de travailler sur des graphes de taille très importante : plus de 14 000 000 sommets et 34 000 000 arcs.
Vous pouvez obtenir une solution en utilisant l'exécutable pcc par la commande `pcc grapheNewYork.csv` et la visualisation du graphe en utilisant l'exécutable pccgraph par la commande `pccgraph grapheNewYork.csv 1`
Commencez par travailler sur un graphe de petite taille.
Tous les fichiers sont dans le répertoire `/users/prog1a/C/librairie/projets22018`.
Ne copiez pas les fichiers dans votre espace de travail car ils sont volumineux.

4.2. Metro et RER Parisien

Le graphe est ici un peu particulier : il y a une station différente par ligne, même si deux lignes ont le même nom de station. Par exemple, on trouve une station Gare du Nord sur la ligne 4 (sommet numéro 84), la ligne 4 (sommet numéro 114), la ligne B (sommet numéro 457), la ligne D (sommet numéro 580). Si vous partez de Gare du Nord, il faut donc soit calculer le plus court chemin à partir de chacun de ces sommets ou bien définir un arc de cout nul entre chacun de ces sommets ou tout autre solution à votre gout.
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. Les correspondances à pied (exemple : entre Gare du Nord et La chapelle) ont un coût pré-établi est de 600. Le coût des autres arcs dépend du moyen de transport (métro, tramway et RER).
Dans un premier temps, vous désignerez les stations en utilisant leur numéro (un entier). 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.

5. REALISATION

5.1. 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 de données utilisées, en explicitant le rôle de chaque élément des structures
- les modules : rôle de chaque module (couple de fichiers .c/.h)
- 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 du graphe avant même d'essayer de calculer un chemin.
  - tests d'intégration : quels sont les tests que vous allez faire pour prouver que l'application fonctionne, sur quels exemples.
- 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. Bien répartir le travail pour permettre au projet de se dérouler complètement.
- Le planning de réalisation du projet jusqu'à la date du rendu.

5.2. Conseils de développement

- Essayer de prévoir un ordre logique dans la réalisation de vos fonctions : par exemple, il faut commencer par écrire la fonction de lecture du graphe et la vérifier. Les sommets contiennent les voisins qui sont des listes d'arcs. Il faut donc commencer par écrire et tester les fonctions sur les listes d'arcs (création, ajout, visualisation). Ensuite, il faut écrire la fonction de chargement du graphe, celle d'affichage du graphe et un programme de test de ces 2 fonctions
- Partagez vous le travail en fonction de vos compétences afin de ne pas ralentir le deroulement du projet.
- Mettez régulièrement le travail en commun.
- Prévoyez un développement incrémental en testant toutes vos fonctions au fur et à mesure. Ne pas ecrire tout le code et compiler au dernier moment

- Validez le programme réalisé sur les graphes simples avant de le tester sur des graphes plus conséquents.
- Attention : vous devez vous assurer que la compilation et vos programmes fonctionnent sur les machines de l'école.

5.3. Rendu final :

Chaque membre du binome copiera l'ensemble des fichiers constituant le livrable dans les répertoires : /users/phelma/phelma2018/**mon-login**/tdinfo/seance15.
Le livrable sera constitué :

- du rapport du projet, format PDF. N'oubliez pas de faire figurer vos noms...
- des sources de votre programme
- du Makefile
- d'un fichier README expliquant comment compiler et lancer votre (vos) programme(s)

Vous ferez un rapport court (5 à 10 pages) explicitant les points suivants :

1. Implantation
    1. État du logiciel : ce qui fonctionne, ce qui ne fonctionne pas
    2. Tests effectués
    3. Exemple d'exécution
    4. Les optimisations et les extensions réalisées
2. Suivi
    1. Problèmes rencontrés
    2. Planning effectif
    3. Qu'avons nous appris et que faudrait il de plus?
    4. Suggestion d'améliorations du projet
3. Conclusion

6. EXTENSIONS

6.1. Optimisation

6.1.1. Recherche du minimum dans l'ensemble C (algo de dijsktra)

La recherche du minimum dans l'ensemble C (algorithme de Dijkstra) peut se révéler coûteuse, car elle est en o(n) dans une approche classique. Si on implante l'ensemble C sous forme d'un tas, le maintien du tas est en o(log(n)) et le minimum est au sommet de ce tas : sa recherche est donc en o(log(n)) car la suppression du minimum oblige ensuite à maintenir la structure du tas (voir le TD sur les tas). La principale difficulté ici est que le PCC des sommets déjà dans le tas peut être modifié par la suite (ligne 13 de l'algorithme). Dans ce cas, il faut aussi mettre à jour le tas, car le sommet modifié peut éventuellement ne plus respecter la condition sur les tas : le père est plus petit que les 2 fils. Il faut donc éventuellement faire remonter le sommet dont le PCC est modifié. Pour retrouver rapidement le sommet dans le tas, vous pouvez définir un champ supplémentaire dans la structure de sommet, qui contiendra l'indice du tableau tas où se trouve le sommet ou -1 si le sommet n'est pas dans le tas (ensemble C).

6.1.2. Recherche des stations par leur nom

La recherche des stations par leur nom au lieu de leur numéro peut être facilitée par une table de hachage, avec gestion des collisions par liste chainées. En effet, tous les sommets qui ont le même nom seront alors dans la même liste chainée, puisque le hachage se fait sur le même nom. Cette liste est très réduite par rapport au nombre de sommets total et la recherche rapide. Notez que d'autres noms de station peuvent cependant se retrouver dans cette liste de collisions, si la fonction de hachage de 2 noms de station différents donne le même indice entier.

6.2. Version Graphique

En disposant des coordonnées GPS des différents sommets, vous pouvez dessiner le graphe et proposer une interface Homme Machine autre que clavier et écran texte.