Université de Haute-Alsace
ENSISA

ENSISA IR 2ème année
Année 2025/2026

Algorithmie et Programmation

Projet : Réseau de transport et parcours de graphes

Instructions

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile !

Coder proprement avec des noms de variables et de fonctions/méthodes bien choisis. Ajoutez des commentaires utiles à votre code afin de pouvoir encore le comprendre dans quelques jours.

Respecter les coding style. Je vous conseille de suivre les consignes du CS50[1] ou du projet GNU[2].

Simple is beautiful. N'essayez pas de coder compliqué, restez simple, cela sera plus efficace et moins sujet aux erreurs.

Réfléchissez avant de coder, prenez un peu de temps pour dessiner/écrire vos idées sur une feuille. Le temps de la réflexion que vous prendrez, vous économisera énormément de temps de debug.

Objectif du projet

On souhaite modéliser un réseau de transport (bus, métro, tram, ...) sous forme de graphe et proposer des fonctionnalités d'analyse et de recherche d'itinéraires.

Le projet a pour but de :

— manipuler des structures de données dynamiques en C (listes chaînées, tableaux dynamiques) ;
— manipuler des structures complexes (tableau de hachage) ;
— mettre en œuvre des parcours de graphes (Dijkstra) ;
— utiliser des algorithmes de tri simples et mesurer leur coût ;
— structurer un programme C de taille moyenne (découpage en fichiers, fonctions, tests).

Votre programme sera écrit en langage C, compilable avec `gcc` sans avertissements (option `-Wall`) et fonctionnera en ligne de commande.

1 Description générale

Le réseau de transport est décrit dans un fichier texte. Votre programme doit :

1. Lire le fichier et construire une représentation en mémoire du réseau sous forme de graphe
2. Permettre différentes requêtes interactives :
   — afficher les informations sur une station ;
   — lister les voisins d'une station ;
   — vérifier si deux stations sont reliées (existence d'un chemin) ;
   — calculer un chemin de longueur minimale (en nombre d'arêtes) entre deux stations ;
   — afficher les composantes connexes du réseau.
3. Analyser le réseau en calculant le degré des stations et en les triant selon ce degré.

Toutes les entrées/sorties se feront via le terminal (stdin/stdout). L'utilisateur choisira les opérations via un menu texte.

1. CS50 : `https://cs50.readthedocs.io/style/c/`
2. GNU : `https://www.gnu.org/prep/standards/html_node/Writing-C.html`

2 Format du fichier d'entrée

Le réseau est défini dans un fichier texte (`metro.txt` comportant deux types de lignes :
— les lignes `STATION` décrivent les stations ;
— les lignes `EDGE` décrivent les liaisons (arêtes) entre stations.

Contraintes générales :
— Les identifiants des stations sont des entiers dans l'intervalle `[0..N-1]`.
— Le fichier peut contenir des lignes de commentaires commençant par le caractère `#`.
— Vous devez ignorer proprement les lignes vides ou de commentaire.
— Vous devez faire un minimum devérification d'erreurs :
  — station référencée par une arête mais non définie ;
  — doublons évidents (par exemple deux lignes d'arête identiques).

3 Structures de données

3.1 Arcs et sommets

Le graphe sera représenté en listes d'adjacence. Vous définirez et justifierait le choix des structures de données.

Remarques importantes :
— le réseau est représenté par un tableau de `stations` de taille `nb_stations` ;
— vous utiliserez `malloc` / `free` pour gérer la mémoire dynamique.

Il est interdit d'utiliser des bibliothèques de conteneurs « toutes faites » (par exemple `glib`, C++ STL, etc.). Seule la bibliothèque standard C est autorisée.

4 Fonctionnalités obligatoires

4.1 Chargement du réseau depuis le fichier

Vous devez écrire une fonction qui :
— ouvre le fichier dont le nom sera passé en argument du programme (par exemple `./metro metro.txt`) ;
— lit l'ensemble des lignes et détermine le nombre de stations ;
— alloue le tableau de `station` ;
— initialise le nom de chaque station ;
— crée les listes d'adjacence pour les arêtes :
  — pour chaque ligne `EDGE`, créer un arc ;
— corriger les erreurs (doublons, arrêtes entre des stations inexistantes, données incomplètes, stations sans arrête, ...).

Le programme doit afficher un message d'erreur clair en cas de problème avec le fichier (fichier introuvable, format invalides, etc.).

4.2 Menu principal

Après le chargement du réseau, votre programme doit afficher un menu du type :

```
===== MENU RESEAU DE TRANSPORT =====
1 - Afficher les informations d'une station
2 - Lister les voisins d'une station
3 - Calculer un chemin minimal
4 - Afficher les stations triées par degré
0 - Quitter
Votre choix :
```

Vous devrez répéter l'affichage du menu jusqu'au choix `0` (avec vérification de saisie).

4.3 Affichage des informations d'une station

Pour un identifiant ou un nom de station donné, le programme doit :
— vérifier que l'identifiant ou le nom est valide ;
— afficher l'identifiant de la station ;
— afficher le nom de la station ;
— afficher le nombre de voisins sortants (degré sortant) ;
— optionnel : lister les arêtes sortantes avec le temps associé.
Notez que pour une recherche efficace des stations par nom, vous devrez également utiliser une table de hachage.

4.4 Liste des voisins d'une station

Pour un identifiant ou un nom de station donné, le programme affiche la liste des voisins atteignables directement, dans un format lisible (par exemple : `id_voisin - nom_voisin (temps)`).

4.5 Chemin minimal en nombre d'arêtes (Dijkstra)

On souhaite maintenant construire un chemin de longueur minimale (en temps de trajet) entre `s` et `t`, s'il existe.
— Afin de ne pas avoir à refaire de façon répété les calculs, vous utiliserez l'algorithme de Dijkstra au chargement des données et vous stockerez le résultat dans les noeuds de votre graphe.
— Si `t` est atteignable, vous reconstruirez le chemin en remontant les prédécesseurs, puis vous l'afficherez dans l'ordre `s -> ... -> t` ;
— S'il n'y a pas de chemin entre les deux stations vous l'indiquerez également ;
— les stations pourront être indiquées par leur identifiant ou leur nom et il faut gérer les erreurs (identifiant ou nom inexistant)

5 Degré des stations et tri

5.1 Calcul du degré

Pour chaque station, on définit son degré comme le nombre de voisins (nombre d'arcs sortants, ou nombre d'arcs incidents selon la définition que vous choisirez et expliquerez dans le rapport).
— Créez une structure permettant de stocker le couple `(id_station, degre)`.
— Remplissez un tableau de ces couples en parcourant le graphe.

5.2 Tri des stations par degré

On souhaite afficher la liste des stations triées par degré croissant (ou décroissant, à préciser). Pour cela :
— vous implémenterez les algorithmes de tri suivant parmi :
  — tri par sélection ;
  — tri par insertion ;
  — tri rapide (version simple, récursive).
— pour chaque algorithme de tri, vous compterez :
  — le nombre de comparaisons ;
  — le nombre de permutations (échanges d'éléments).
— vous afficherez ces statistiques à la fin du tri.

6 Contraintes de programmation

— Le projet doit être écrit en langage C standard (C99 ou C11).
— Le code doit compiler sans avertissement avec `gcc -Wall -Wextra`.
— Le programme sera découpé en plusieurs fichiers (`.c` et `.h`) de manière cohérente (par exemple : fichier pour la gestion du graphe, fichier pour les parcours, fichier pour le tri, fichier pour `main`, etc.).
— Vous devez gérer correctement la mémoire dynamique : toute allocation `malloc` doit être libérée par un `free` approprié.
— Le code doit être clairement commenté (au minimum pour toutes les fonctions non triviales).

7 Livrables

Chaque binôme rendra :

1. Le code source complet (`.c`, `.h`) avec un `Makefile` permettant de compiler le projet avec la commande `make`.
2. Un ou plusieurs fichiers de test de réseaux de transport.
3. Un rapport court (2 à 4 pages) au format PDF contenant :
   — une description et justification des structures de données utilisées (schémas des listes d'adjacence, tableau de sommets, etc.) ;
   — une description des algorithmes de tri choisis et des résultats de comparaison (nombre de comparaisons, nombre d'échanges) sur un même jeu de données et ce que vous pouvez en conclure
4. Le tout sera déposé sur moodle dans un fichier zip `nomEtudiant1_nomEtudiant2.zip`

8 Évaluation (indications)

L'évaluation prendra en compte :
— la correction fonctionnelle du programme (chargement, menu, fonctionnalités demandées) ;
— la qualité des structures de données et du découpage du code ;
— la qualité de l'implémentation des parcours de graphes (Dijkstra) et des tris ;
— la gestion de la mémoire (absence de fuites, libération correcte) ;
— la gestion des erreurs ;
— la clarté et la précision du rapport.