Cours de programmation séquentielle - TP17, noté

Recherche du plus court chemin dans un réseau ferré

Buts

Dans ce travail vous verrez les concepts suivants:

— Manipulation de chaines de caractères.
— Implémentation d'une interface en ligne de commande.
— Appliquer un algorithme de plus court chemin dans un graphe.
— Afficher une représentation graphique avec la librarie SDL.

Énoncé

Le but de ce TP est d'implémenter un outil permettant d'afficher le réseau ferré Suisse et d'effectuer une recherche d'itinéraire dans celui-ci (recherche du plus court chemin dans un graphe).

Lecture des données

Dans un premier temps, vous devez construire la représentation du graphe à l'aide des fichiers `cities.csv` et `connections.csv`. Ceux-ci contiennent le nom des villes, leurs positions géographiques et le temps de parcours entre chacune d'elles. On considèrera un graphe non orienté pour représenter le réseau dans le cadre de ce TP.

Vous devez :

— Créer un tableau contenant les noms des villes, ainsi que leurs positions sur la carte.
— A partir du tableau des villes et des informations de la connectivité, créer la matrice d'adjacence. Les poids des arêtes représentent le temps de parcours entre les villes.

Calcul du plus court chemin avec l'algorithme de Floyd

L'algorithme de Floyd(–Warshall)

L'algorithme de Floyd (ou Floyd–Warshal) est un algorithme qui sert à trouver les plus courts chemins entre toute paire de sommets d'un graphe.

Dans cette section nous allons d'abord décrire l'algorithme puis voir son implémentation.

Description de l'algorithme

Considérons un graphe avec N sommets, numérotés de 0 à $N-1$. Soit la fonction `shortest_path(i,j,k)` la fonction retournant le plus court chemin entre le sommet `i` et le sommet `j` en utilisant uniquement les sommets `0, ..., k-1` sommets intermédiaires. En supposant cette fonction connue, nous voulons trouver le plus court chemin en utilisant les sommet `0` à `N-1`. Pour chaque `shortest_path(i, j, k)` nous avons deux possibilités:

1. le chemin ne passe pas par `k-1` (il ne passe que par les sommets `0` à `k-2`).
2. le chemin passe par `k-1`: d'abord de `i` à `k-1` puis de `k-1` à `j`.

Dans le premier cas, nous savons que le plus court chemin est défini par `shortest_path(i,j,k-1)`. Dans ce second cas, le chemin serait la réunion des chemins de `i` à `k`, puis de `k` à `j`. Ce chemin serait donc la somme des chemins `shortest_path(i, k, k-1)` et `shortest_path(k,j,k-1)`.

De la définition de la fonction `shortest_path(i,j,k)`, on peut déduire que `shortest_path(i,j,0)` est donné par

`shortest_path(i,j,0)=w(i,j)`,

où `w(i,j)` est le poids de l'arrête entre les sommets `i` et `j`.

Finalement, on peut définir le plus court chemin récursivement

```
shortest_path(i,j,k) = min(shortest_path(i,j,k-1),
    shortest_path(i,k,k-1) + shortest_path(k,j,k-1)).
```

Implémentation de l'algorithme de Floyd

L'algorithme de Floyd s'implémente à l'aide de la matrice d'adjacence. Il est décrit en pseudo-code comme

```
void floyd_warshall(int num, int dist[num][num], int next[num][num]) {
    // init dist with INT_MAX
    // init next with -1

    for each edge (i, j) {
        dist[i][j] = w(i, j)  // The weight of the edge (i, j)
        next[i][j] = j
    }
    for each vertex j {
        dist[j][j] = 0
        next[j][j] = j
    }
    for (int k = 0; k < num; ++k) { // standard Floyd-Warshall implementation
        for (int i = 0; i < num; ++i) {
            for (int j = 0; j < num; ++j) {
                if (dist[i][j] > dist[i][k] + dist[k][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j]
                    next[i][j] = next[i][k]
                }
            }
        }
    }
}
```

Pour retrouver le chemin il faut simplement suivre les indices dans la matrice `next` de la façon suivante

```
void get_path(int i, int j, int num, int next[num][num], list path) {
    if next[i][j] is negative {
        return
    }
    list_add(path, i)
    while (i != j) {
        i = next[i][j]
        list_add(path, i)
    }
}
```

Tests

Afin de tester le code, vérifiez que si vous chercher les temps de parcours entre les villes :

```
Geneve; Sion
Andermatt; Coire
Schaffouse; Lausanne
Lucerne; Delemont
Coire; Geneve
Bale; Bellinzone
Andermatt; Geneve
Lausanne; Delemont
Neuchatel; Bellinzone
Bellinzone; Geneve
Sion; Bale
Lausanne; Geneve
St.-Moritz; Geneve
Sion; Schaffouse
St.-Gall; Lausanne
Neuchatel; Andermatt
Sion; Neuchatel
Bale; Geneve
Geneve; Bale
Berne; Bellinzone
```

on obtient:

```
101 100 188 107 271 205 263 89 257 316
190 34 387 255 212 227 107 157 157 215
```

Puis les chemins sont donnés par

```
[Geneve:Lausanne:Sion]
[Andermatt:Coire]
[Schaffouse:Zurich:Berne:Lausanne]
[Lucerne:Bale:Delemont]
[Coire:Zurich:Berne:Lausanne:Geneve]
[Bale:Lucerne:Bellinzone]
[Andermatt:Sion:Lausanne:Geneve]
[Lausanne:Neuchatel:Delemont]
[Neuchatel:Berne:Lucerne:Bellinzone]
[Bellinzone:Lucerne:Berne:Lausanne:Geneve]
[Sion:Lausanne:Neuchatel:Delemont:Bale]
[Lausanne:Geneve]
[St.-Moritz:Coire:Zurich:Berne:Lausanne:Geneve]
[Sion:Lausanne:Berne:Zurich:Schaffouse]
[St.-Gall:Zurich:Berne:Lausanne]
[Neuchatel:Berne:Lucerne:Andermatt]
[Sion:Lausanne:Neuchatel]
[Bale:Delemont:Neuchatel:Lausanne:Geneve]
[Geneve:Lausanne:Neuchatel:Delemont:Bale]
[Berne:Lucerne:Bellinzone]
```

Affichage graphique

Votre programme doit afficher le contour de la Suisse, dont les coordonnées se trouvent dans le fichier `swiss.txt`, avec une coordonnée par ligne. Vous devrez également afficher les villes (par exemple un carré ou un cercle pour charge ville) ainsi que les connexions du réseau. Lorsqu'un chemin sera calculé, celui-ci devra être imprimé en surimpression par dessus le réseau.

Pour l'affichage graphique du programme, on propose d'utiliser la librairie SDL2. Vous trouverez dans les fichiers `gfc.c`, `gfx.h` et `drawing.c` un exemple d'utilisation de la librarie. Pour compiler avec cette librairie, il faut ajouter le flag `-lSDL2` à la compilation.

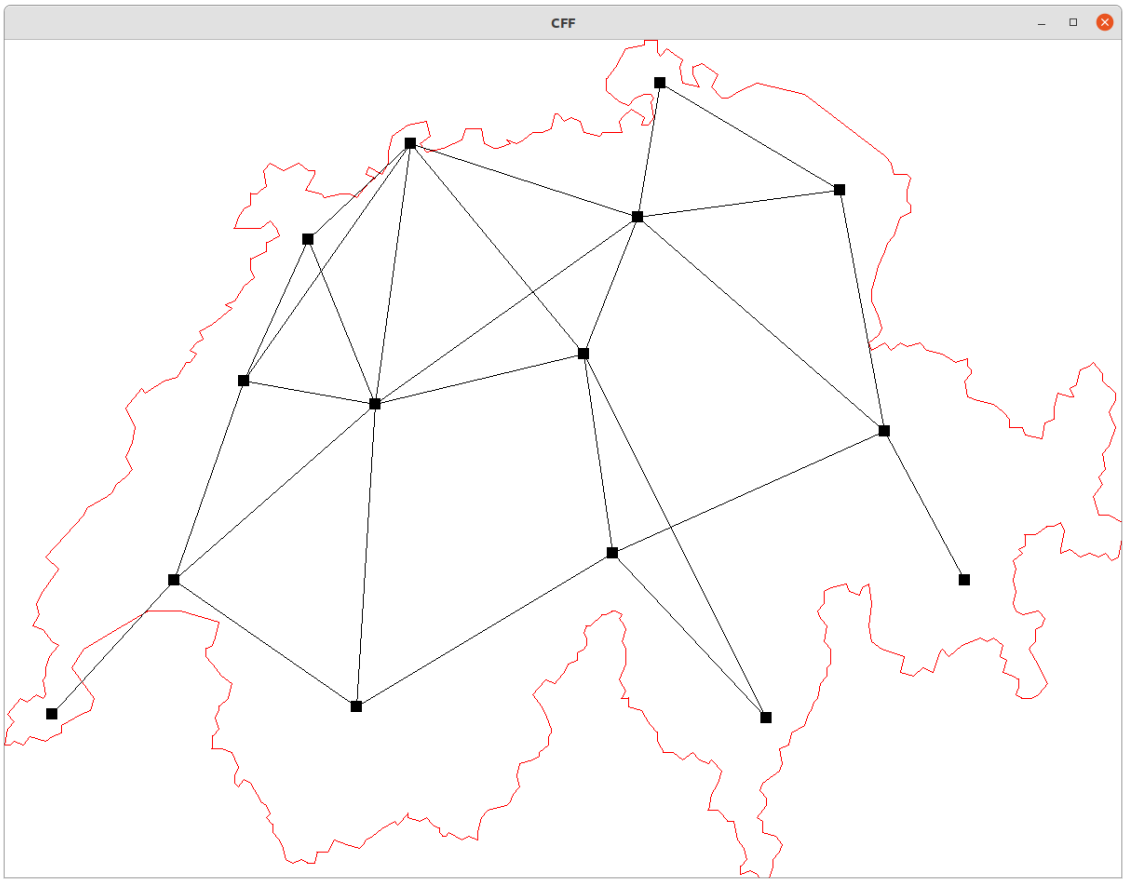

Figure 1 – Affichage de la Suisse et du réseau ferré.

Programme à implémenter

Vous devez implémenter un programme nommé `cff`. Au lancement de celui-ci, toutes les données seront chargées (liste de villes, réseau, contour de la Suisse) et le pays ainsi que le réseau seront affichés.

L'utilisateur pourra intérragir avec le programme à l'aide d'une interface textuelle qui permettra d'entrer une ville de départ et de destination. Si les villes sont trouvées dans la liste des villes, le chemin ainsi que le temps de parcours seront affichés sous forme textuelle. Le chemin sera également affiché sous forme graphique sur le réseau de transport.

Une fois un parcours affiché, l'utilisateur pourra choisir une nouvelle ville de départ et d'arrivée, jusqu'à ce que l'instruction `quitter` soit donnée en entrée. Voici un exemple d'interaction avec l'interface textuelle :

```
> Entrez une ville de départ : Geneve
> Entrez une ville d'arrivée : Sion
> Le trajet entre Genève et Sion est Geneve:Lausanne:Sion, le temps de parcours est 101 minutes
> Entrez une ville de départ : Andermatt
> Entrez une ville d'arrivée : Coire
> Le trajet entre Andermatt et Coire est Andermatt:Coire, le temps de parcours est 100 minutes
> Entrez une ville de départ : quitter
$
```

Modalités

Ce TP est évalué et devra être rendu au plus tard le 24.06.2022. Un dépôt GIT à votre nom sous la forme https://githepia.hesge.ch/prog_soir_2022_tp_17/VOTRE.NOM sera créé. La visibilité de ce dépôt doit rester privée et accessible uniquement à vous et à l'enseignant. Si vous n'avez pas accès à ce dépôt, faites le savoir à l'enseignant. C'est le code présent dans ce dépôt au moment du rendu qui sera pris en compte pour l'évaluation.

La racine de votre dépôt devra contenir un `makefile` capable de compiler votre programme. Votre programme doit être compilé avec les sanitizers (options `-fsanitize=address -fno-omit-frame-pointer`) et vous devez gérer correctement la mémoire.

L'intégralité de votre code doit se trouver dans un dossier nommé `src` à la racine de votre dépôt et ses sous dossiers. Le programme principal doit se trouver dans un fichier nommé `cff.c` et le programme généré doit s'appeler `cff`.

Si vous voulez ajouter des informations complémentaires à destination de l'utilisateur (exemple d'utilisation, bug connu et comment l'éviter, ...) vous pouvez le faire dans le fichier `README.md` à la racine du projet.

La structure, la modularisation et la lisibilité de votre code sont prises en compte pour l'évaluation. Préférez des noms de fonctions et de variables explicites, des fonctions les plus courtes possibles et dont le rôle est aisé à comprendre. Evitez au maximum la logique dans le programme principal. Restez constant dans le style de nommage (pas de mélange anglais / français ou de snake / camel case). Evitez de passer des pointeurs lorsque cela est inutile. ...

Vous devez faire figurer un commentaire de documentation pour chaque fonction (un commentaire expliquant ce que fait la fonction, quelles sont ses entrées et sorties). Vous pouvez faire figurer ces commentaires dans les fichiers `.h` ou `.c`. Ajoutez des commentaires explicatifs lorsque cela est nécessaire.