Rapport de projet - programmation semestre 2

SUN Quan FILIPPI Angelo

Année 2015/2016

Table des matières

Chapitre 1

Introduction

L'objectif de ce mini-projet de programmation est d'implémenter l'algorithme de Bellman dans un cas simple : trouver le plus court chemin entre deux stations desservies par différents types de transports en commun. L'algorithme est donné dans le sujet, le but de cette réalisation est donc de nous faire manipuler des structures diverses et ainsi appliquer le cours de structures de données.

L'intégralité du projet est sur GitHub : `https://github.com/THEKINGAF/projet-s2-info`.

Chapitre 2

Spécifications

2.1 Données : description des structures de données

2.1.1 Graphe

C'est la structure principale, la plupart des fonctions vont travailler dessus. Elle représente le graphe (sommets et arcs). Elle comporte 4 éléments :

— le nombre de sommets
— le nombre d'arcs
— un pointeur sur la liste de sommets
— un pointeur sur la liste des arcs

Nous avons opté pour une allocation d'un bloc mémoire pour chaque liste plutôt que des listes chaînés comme le proposait le sujet.

Notre décision vient du fait que le nombre d'arcs et de sommets sont des informations connues (premières lignes du fichier, et la lecture du fichier est la première opération réalisée par le programme). De plus, le programme n'ajoute pas de sommets ni d'arcs. Or, d'après le cours de structures de données, il est préférable d'opter pour un tableau de valeurs contigües dans cette situation, un tableau occupe moins de mémoire (pas besoin de pointeur sur l'élément suivant) et réduit les accès à la mémoire vive en permettant au processeur d'utiliser le cache.

2.1.2 Sommet

La structure sommet doit contenir les informations stockées dans le fichier de description du graphe :

— id
— latitude
— longitude
— nom de la ligne
— nom de la station

ainsi que de nombreuses autres informations utiles pour le programme ou pour l'optimisation de l'algorithme de Bellman :

— présence ou non dans la file (optimisation de l'algorithme de Bellman)
— pointeur vers l'arc qui arrive à ce sommet (reconstruction du chemin)
— poids
— nombre d'arcs partant de ce sommet
— pointeur sur la liste des arcs partant de ce sommet

2.1.3 Arc

Le structure d'arc contient simplement les informations sur les arcs du fichier :

— sommet de départ
— sommet de d'arrivée
— poids de l'arc

2.2 Fonctions : entête et rôle des fonctions essentielles

Les fonctions d'affichage, de test (vide ou non), d'initialisation, d'allocation et de libération mémoire ne seront pas détaillées dans cette section.

Voici le prototype de la fonction principale :

```
CHEMIN bellman (GRAPHE g, SOMMET * depart, SOMMET * arrivee);
```

Cette fonction est une retranscription exacte de l'algorithme de Bellman tel qu'il est décrit dans le sujet, elle travaille sur le graphe g afin de mettre à jour les poids des sommets dans le but de déterminer le plus court chemin qui va du sommet depart au sommet arrivee.

```
CHEMIN reconstruit_chemin (GRAPHE g, SOMMET * depart, SOMMET * arrivee);
```

La fonction bellman utilise la fonction reconstruit_chemin qui renvoie le chemin menant du sommet depart au sommet arrivee après avoir calculé les poids les plus faibles pour chaque sommet à l'aide de l'algorithme de Bellman.

Dans un second temps, il faut une fonction qui détermine le plus court chemin entre deux stations, c'est à dire en les désignant non plus par leur id mais par leur nom. Elle a pour prototype :

```
CHEMIN plus_court_chemin (GRAPHE g, char * depart, char * arrivee);
```

Cette fonction utilise simplement la fonction bellman précédente sur toutes les combinaisons de sommets de départ et d'arrivée homonymes. Pour déterminer tous les sommets correspondant à un nom on utilise la fonction suivante :

```
STATION * construit_station (GRAPHE g, char * nom);
```

Elle renvoie un objet STATION qui liste les id des sommets nommés nom.

2.3 Tests : quels sont les tests prévus

Nous allons tester progressivement nos fonctions. Après avoir écrit les structures et les fonctions nécessaires à la lecture du fichier nous testerons la structure du graphe avec une fonction d'affichage qui devra parcourir les sommets et les arcs afin de vérifier que le fichier ait bien été lu et que les structures soit correctement remplies.

Après s'être assurés que nos structures contenaient des valeurs valides, nous testerons l'algorithme de Bellman appelé avec les numéros des sommets. Une fois l'algorithme de Bellman fonctionnel, il faudra réaliser les fonctions permettant d'appeler l'algorithme non pas via les numéros des sommets mais par leurs noms.

2.4 Répartition du travail et planning prévu : qui fait quoi et quand ?

Nous avons prévu de découper le travail en 4 parties :

2 premières séances :
— lecture du graphe
— structures graphe, sommet et arc

2 dernières séances :
— algorithme de Bellman + optimisations
— structure chemin, station et recherche par nom

Sur chacune des paires de séances chaque élève travaille sur un des deux points listés et on met en commun à la fin de chaque séance. En fonction de l'avancement on décide ou non de travailler en dehors des séances.

Chapitre 3

Implémentation

3.1 Etat du logiciel : ce qui fonctionne, ce qui ne fonctionne pas

Le logiciel fonctionne correctement, il est capable de donner le plus court chemin entre deux stations désignés par leurs noms.

3.2 Tests effectués

Les premiers essais ont été réalisés à l'aide du programme **testgraphe**. Ils avaient pour objectif de valider la structure de graphe et la lecture du fichier.

Les tests suivants portaient sur l'algorithme de Bellman en lui-même. Une fois fonctionnel il a été optimisé. Il a ensuite fallu tester la construction de la structure STATION qui, une fois vérifiée nous a permis de coder la recherche de plus court chemin à partir des noms. Nous avons ainsi pu vérifier le programme dans son intégralité.

Il a alors été possible de faire des test plus poussés, par exemple tester les fuites mémoires sur l'ensemble du programme :

```
$ valgrind --leak-check=full  ./bellman graphes/metro.csv "Montparnasse Bienvenue"
"Porte Maillot"
==5431== Memcheck, a memory error detector
==5431== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5431== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5431== Command: ./bellman graphes/metro.csv Montparnasse\ Bienvenue Porte\ Maillot
==5431==
Chemin :
(361)--->(360) ligne M13 |Duroc
(360)--->(359) ligne M13 |Saint-Francois Xavier
(359)--->(358) ligne M13 |Varenne
(358)--->(357) ligne M13 |Invalides
```

```
(357)--->(356) ligne M13 |Champs Élysées, Clémenceau
(356)--->(9) ligne M1 |Champs Élysées, Clémenceau
(9)--->(8) ligne M1 |Franklin D. Roosevelt
(8)--->(7) ligne M1 |George V
(7)--->(6) ligne M1 |Charles de Gaulle, Étoile
(6)--->(5) ligne M1 |Argentine
(5)--->(4) ligne M1 |Porte Maillot
coût : 1078.123449
==5431==
==5431== HEAP SUMMARY:
==5431==     in use at exit: 0 bytes in 0 blocks
==5431==   total heap usage: 6,434 allocs, 6,434 frees, 550,272 bytes allocated
==5431==
==5431== All heap blocks were freed -- no leaks are possible
==5431==
==5431== For counts of detected and suppressed errors, rerun with: -v
==5431== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Comme on peut le voir sur le résultat ci-dessus, tout l'espace mémoire alloué a été libéré à la fin de l'éxecution.

3.3 Exemple d'exécution

```
$ bellman graphes/metro.csv "Montparnasse Bienvenue" "Porte Maillot"
Chemin :
(361)--->(360) ligne M13 |Duroc
(360)--->(359) ligne M13 |Saint-Francois Xavier
(359)--->(358) ligne M13 |Varenne
(358)--->(357) ligne M13 |Invalides
(357)--->(356) ligne M13 |Champs Élysées, Clémenceau
(356)--->(9) ligne M1 |Champs Élysées, Clémenceau
(9)--->(8) ligne M1 |Franklin D. Roosevelt
(8)--->(7) ligne M1 |George V
(7)--->(6) ligne M1 |Charles de Gaulle, Étoile
(6)--->(5) ligne M1 |Argentine
(5)--->(4) ligne M1 |Porte Maillot
coût : 1078.123449
```

3.4 Les optimisations et les extensions réalisées

La première optimisation que nous avons implémentée est celle proposée par le sujet : utiliser une file pour stocker les sommets dont le coût a été modifié à l'itération précédente afin de réduire le champs des arcs à traiter. Pour ce faire nous avons applisué l'astuce également suggérée par le sujet qui consiste à rajouter un champ à la structure SOMMET afin de repérer la présence ou

non du sommet dans la file. Nous avons utilisé le code produit dans les séances précédentes pour la structure File :

```
// struture de file de sommets utilisee dans l'algorithme de Bellman
    optimise
struct cellule \{
   SOMMET * elmt;
   struct cellule* suiv;
\};

typedef struct cellule CELLULE;
typedef struct cellule* File;

// fonctions sur la file de sommets
File creer_file (void);
int file_vide (File f);
File enfiler( SOMMET * elmt, File f);
SOMMET * defiler (File* pf);
```

Chapitre 4

Suivi

4.1 Problèmes rencontrés

En dehors des erreurs de syntaxes et d'inattention qui sont apparues aux premières compilation, nous avons rencontré des fuites mémoires au cours des tests individuels des fonctions. Nous avons alors utiliser Valgrind pour trouver la source de ces erreurs : des pointeurs qui étaient perdus et donc des allocations mémoires qui n'étaient jamais libérées.

4.2 Planning effectif

Le planning défini en début de projet à été respecté en ayant recourt à deux séances non encadrées. Les bilans réalisés à chaque fin de séance nous ont permis de suivre précisément l'avancement au sein du binôme.

4.3 Qu'avons nous appris

En plus d'avoir appliqué les notions vues en cours sur les structures, nous avons appris à utiliser les programmes gdb et Valgrind destinés au débogage et à la visualisation des fuites mémoires. L'utilisation de GitHub pour la gestion du code est également un point enrichissant de ce projet.

4.4 Suggestion d'améliorations du projet

Notre système de recherche par nom est la première optimisation à réaliser, en effet, appliquer l'algorithme de Belman à toutes les combinaison sommet de départ/sommet d'arrivée portant le même nom n'est pas très efficace.

Une idée consiste à modifier le graphe pour créer des arcs de poids nul entre les sommets portant de même nom afin de représenter le fait que, physiquement, ces sommets désignent une seule et unique station. Cette solution n'est pas

idéale car elle oblige à modifier le graphe et, du fait de la création de liaison artificiellement, implique des modifications de la fonction de création de chemin et d'affichage du chemin : les arcs ainsi créé ne doivent pas apparaîtrent dans le chemin donné à l'utilisateur.

Il est également possible de réaliser un affichage graphique du chemin et du graphe du métro mais créer une interface graphique capable d'afficher un très grand nombre de stations et de liaisons tout en restant lisible demande un temps considérable.

Chapitre 5

Conclusion

Ce projet nous as permis de comprendre le cours de structures de données en l'appliquant à un problème intéressant en informatique et dans d'autres domaines : déterminer le plus court chemin dans un graphe. Dans notre cas, l'algorithme de Bellman a été appliqué à un réseau de transports en commun mais cette situation peut être retrouvée en routage.