Projet CSH
Visualisation d'un réseau de transport
2012-2013

Manon Signoret G32
Thibaut Smith G32

1 Introduction

Nous avons choisi le sujet 4, un programme permettant de visualiser un réseau de transport. Le programme permet de définir et visualiser un réseau de transport d'un type de marchandise : on peut définir l'espace géographique de travail de manière aléatoire (pour les tests), ou manuellement (avec un fichier .txt déjà existant ou avec une saisie manuelle sur la console). De plus, on peut aussi générer des chemins entre plusieurs dépôts dans un ordre choisi. Le programme est écrit en C, et fonctionne entièrement en ligne de commande. Le code est disponible sur GitHub.

2 Cahier des charges

Le programme avait plusieurs objectifs à réaliser. En voici la liste, avec les solutions choisies pour atteindre l'objectif.

Objectifs	**Solutions**
Représentation des graphes dans la mémoire	Structures personnelles
Saisie d'informations venant de l'utilisateur	Entrées (**scanf**)
Représentation des dépôts	Structure et listes chaînées simples
Algorithme de calcul du chemin optimal	Travelling salesman problem, Dijkstra
Possibilité de définir le réseau (nombre de dépôts, leur localisation, les liaisons existantes)	IHM
Choix du dépôt source/destination(s)	IHM
Coût de chaque liaison	IHM
Visualisation du réseau	Graphviz
Transformation fichier texte vers dot (Graphviz)	Programme en C
Interface Homme-Machine (IHM)	Terminal (max 80 colonnes)

3 Choix de conceptions

Dans notre logiciel, une seule structure de données a été utilisée, nous avons donc choisi de ne créer que la structure dépôt, qui est caractérisée par son nom, sa distance avec le dépôt précédent et le sous-ensemble de dépôts suivants. En effet, au début du projet nous avions réfléchi sur le nombre de structures à créer en pensant également à une structure liaison qui pourrait symboliser les arcs entre les différents sommets (dépôts), puis nous avons choisi de ne pas la faire car après avoir commencé à coder, nous nous sommes rendus compte qu'elle était inutile.

```c
typedef struct depot
{
    int    num_depot;        //numéro du dépot
    float  dist;             //distance entre ce dépot et le précédent
    struct depot *suiv;      //sous-ensemble suivant
}S_Depot;
```

FIGURE 1: Structure utilisée dans notre logiciel

Problème majeur survenu Tout d'abord, nous avons eu du mal à comprendre la documentation, et nous n'étions pas sûrs encore de quoi faire, et comment, ce qui a retardé le projet. Nous sommes donc allés chercher de l'aide d'abord vers un binôme en particulier [1], et ensuite sur Internet. Cela nous a débloqué, et à partir de ce moment, le projet a pu commencer.

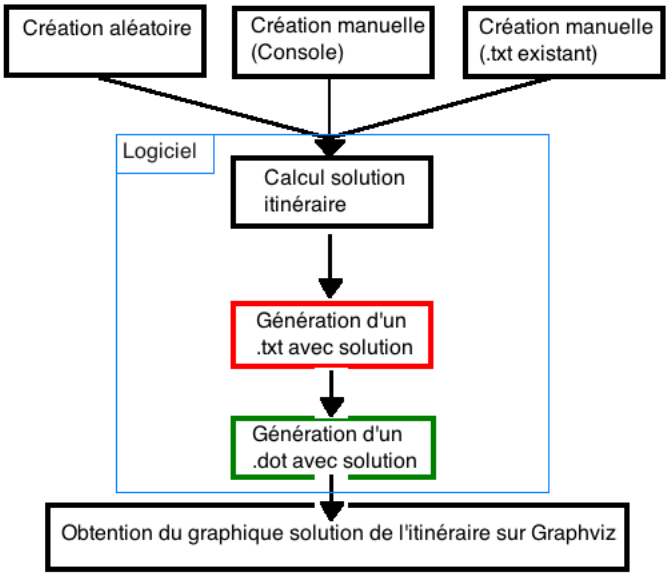

FIGURE 2: Schéma d'explications de notre logiciel

Calcul de l'itinéraire le plus court entre deux dépôts Cette partie là du logiciel a été assez longue. Nous avons donc créé une structure dépôt, et c'est dans cet ensemble là que le traitement des données se fait. Grâce à l'algorithme de Dijkstra (qui n'a pas posé de difficultés particulières), nous calculons le chemin le plus court entre le dépôt choisi en départ et celui en arrivée. Un problème survenu était que Manon Signoret travaille sous Mac et Thibaut Smith sous FreeBSD : l'encodage des fichiers textes contenant les informations était différent. Sous Mac, le programme se lançait mais indiquait une erreur, tandis que sur FreeBSD le programme fonctionnait comme prévu.

```
new-host:Reseau manon2$ make
gcc -o reseau1.o -c reseau1.c -W -Wall -std=gnu99
gcc -o main main.o reseau1.o
Making main binary...
Using following CFLAGS : -W -Wall -std=gnu99
Using following LFLAGS :
new-host:Reseau manon2$ ./main
 ====================
 Projet C :
 Visualisation d'un réseau de transport
 ====================
Nombre de dépôts : 4
Successeurs du dépot "" : (4, 67) (3, 46) (2, 23)
Successeurs du dépot "(null)" : (4, 46) (3, 45)
Successeurs du dépot "depot3" : (4, 27)
Successeurs du dépot "depot4" : Pas de successeur


Quel est le dépôt de départ ? depot1
Segmentation fault: 11
```

FIGURE 3: Problème rencontré sous Mac

Pour subvenir à ce problème Manon Signoret, a donc travaillé sur Ubuntu (en double boot).

1. Clémence Henry, Pierre Monnin

Génération d'un .txt avec la solution Avec la partie décrite ci-dessus nous obtenions donc le chemin optimal entre deux dépôts. Mais il a fallu générer cette solution sur un fichier .txt pour qu'il puisse être (dans l'étape suivante) transformé en un fichier *.dot* compatible avec **Graphviz**. La solution se rajoute donc à la fin du fichier .txt contenant les données, grâce à la commande $fseek(Fichier, 0, SEEK\_END)$; qui nous permet de nous placer à la fin d'un fichier.

```
4
depot1 3 2 023 3 046 4 067
depot2 2 3 045 4 046
depot3 1 4 027
depot4 0

depot4 depot2 depot1

depot3 depot1
```

FIGURE 4: Ajout aux .txt des dépôts solutions

L'un des problèmes redondants était qu'il fallait souvent réajuster le script qui transformait le fichier texte en fichier dot compatible avec Graphviz : au fur et à mesure que l'on rajoutait des informations dans le fichier texte, il fallait le prendre en compte dans le programme de traduction *.txt* > *.dot*.

Génération d'un .dot avec la solution Après l'étape précédente, le plus important était de connaître comment fonctionnait Graphviz pour générer un fichier .dot lisible par ce logiciel. Ainsi nous avons regardé la documentation sur internet et nous avons pu nous inspirer d'une structure générale.

```
digraph G {
depot1->depot2[label=23,color=red];
depot1->depot3[label=46];
depot1->depot4[label=67];
depot2->depot3[label=45];
depot2->depot4[label=46];
depot3->depot1[label=27,color=red];
}
```

FIGURE 5: Génération d'une structure en .dot

4 Représentation visuelle du logiciel

Voici quelques captures d'écran montrant ce que notre logiciel fait sur la console.

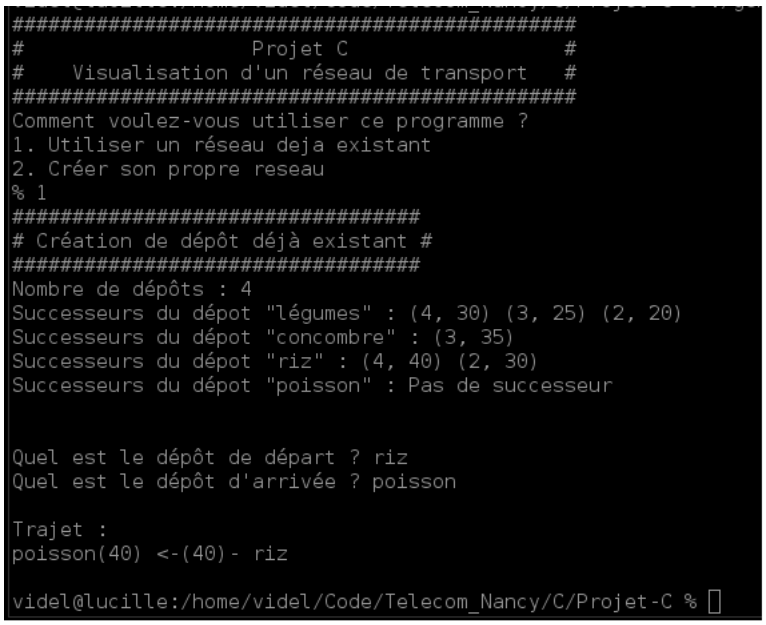

FIGURE 6: Avec un fichier .txt existant

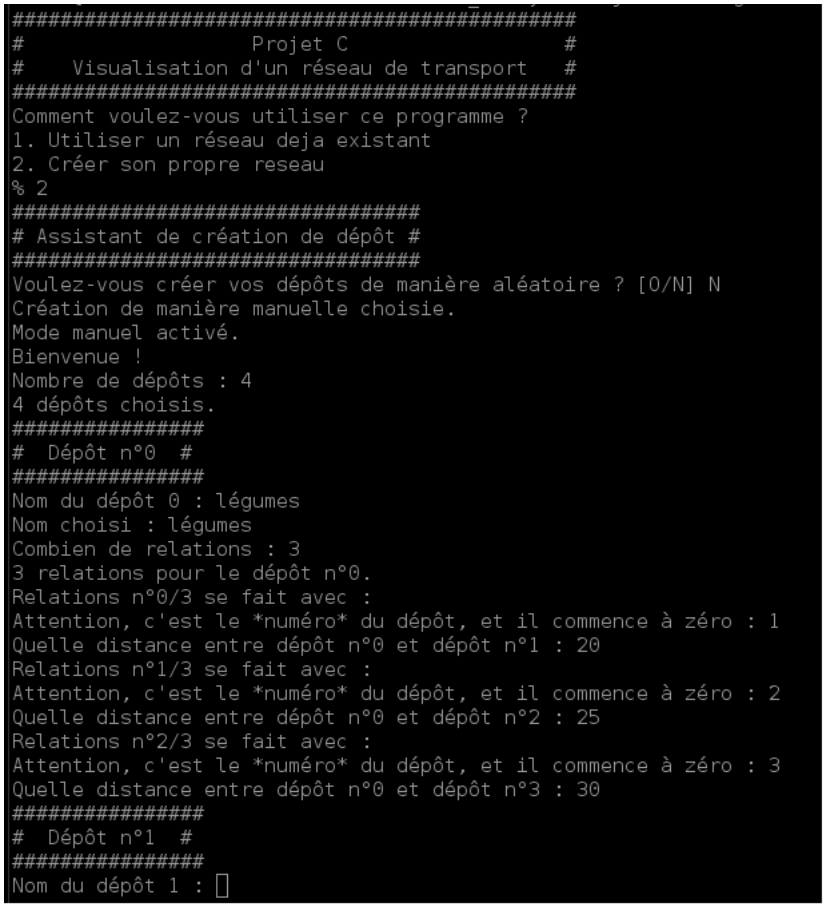

FIGURE 7: Création d'un réseau sur console

```
###############
#  Dépôt n°1  #
###############
Nom du dépôt 1 : concombre
Nom choisi : concombre
Combien de relations : 1
1 relations pour le dépôt n°1.
Relations n°0/1 se fait avec :
Attention, c'est le *numéro* du dépôt, et il commence à zéro : 2
Quelle distance entre dépôt n°1 et dépôt n°2 : 35
###############
#  Dépôt n°2  #
###############
Nom du dépôt 2 : riz
Nom choisi : riz
Combien de relations : 2
2 relations pour le dépôt n°2.
Relations n°0/2 se fait avec :
Attention, c'est le *numéro* du dépôt, et il commence à zéro : 1
Quelle distance entre dépôt n°2 et dépôt n°1 : 30
Relations n°1/2 se fait avec :
Attention, c'est le *numéro* du dépôt, et il commence à zéro : 3
Quelle distance entre dépôt n°2 et dépôt n°3 : 40
###############
#  Dépôt n°3  #
###############
Nom du dépôt 3 : poisson
Nom choisi : poisson
Combien de relations : 0
0 relations pour le dépôt n°3.
Fin de la capture.
Quel est le nom du fichier de sortie ? exemple
Nom du fichier de sortie : exemple
Fin de la génération du fichier exemple.
```

FIGURE 8: Création d'un réseau sur console (suite)

Graphviz Voilà maintenant ce que nous obtenons grâce aux .dot générés par notre logiciel, grâce au logiciel Graphviz. Nous avons coloré en rouge le chemin optimal de l'itinéraire en solution.

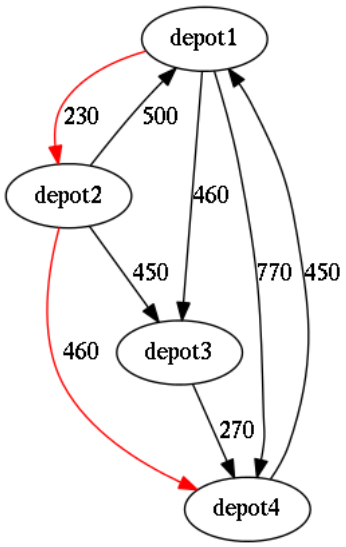

FIGURE 9: Graphique final sur Graphviz

Le programme de traduction *.txt > .dot* peut se découper en parties :

1. Lecture du fichier de données et demande de mémoire à l'OS
2. Capture des informations et sauvegarde dans des tableaux/variables
3. Sauvegarde de ces informations dans un fichier de sortie, dans un format compréhensible par **Graphviz**
4. Libération de la mémoire.

Des problèmes et des bugs sont apparus dans chaque partie de ce programme. Nous avons corrigé le plus de bugs possible grâce aux outils **gdb** et **valgrind** ainsi que les paramètres de compilation $-W - Wall - std = gnu99$, qui se sont avérés très utiles.

5 Gestion du code source

Nous avons choisi d'utiliser Git pour ce projet, étant donné que nous avions déjà utilisé SVN [2]. La forge de l'école ne supportant pas Git, nous avons choisi d'héberger notre code source sur GitHub. C'était une première pour Manon Signoret qui ne le connaissait pas encore. Il s'est avéré qu'il est très facile d'utilisation et très pratique, plus que SVN à notre goût.

6 Nombre d'heures

Nous avons passé un nombre important d'heures sur la compréhension de la documentation de Graphviz avant de savoir quoi faire, Graphviz fonctionnant avec ses propres structures.

Travail	Heures	Membre du binôme
Doc' Graphviz	5H	Les deux
Développement de Dijkstra	18H	Manon Signoret
Développement traitement .txt en .dot	10H	Thibaut Smith
Tests	10H	Les deux
Rapport	3H	Les deux

7 Conclusion

Bien qu'un peu impressionnés au début par le sujet, qui nous semblait le plus difficile, mais également le plus intéressant, nous avons pris plaisir à créer ce logiciel. Les difficultés ne furent que très rares et n'étaient pas là où on les attendait. Nous avions déjà travaillé en binôme pour le projet de TOP et celui de SD, et notre collaboration se passant bien, nous avons donc naturellement décidé de nous mettre de nouveau ensemble pour ce logiciel. Les heures de travail furent réparties équitablement, et chacun était présent pour aider l'autre tout au long de la conception. Ce projet nous a également permis de nous améliorer (ou de débuter) sur certains logiciels, comme Graphviz, Git et également en LaTeX(avec lequel nous avons rédigé notre rapport). Bien que l'algorithme de Dijkstra ne marche pas complètement, nous ne sommes pas déçus du travail accompli.

2. Il ne manque plus que Bazaar, Mercurial, etc !

8 Sources

- http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra
- http://www.techonthenet.com/c_language/standard_library_functions/string_h/strcat.php
- http://en.wikipedia.org/wiki/String.h
- http://gl.developpez.com/tutoriel/outil/makefile/