LEVY Gabi PET C

DAREYS PIERRE

Rapport du projet d’informatique 2020/2021

SOMMAIRE

1.Implantation

1a) Etat du logiciel

Notre fonction ouverture fonctionne ainsi que la fonction graph_print. Nos 2 programmes avec Astar et Dijkstra fonctionnent, ils retournent tous deux le plus court chemin entre le sommet de départ et d'arrivé que ce soit pour les réseaux routiers ou les réseaux de métro. Nous avons également réussi à reconstituer le chemin entre le départ et l'arrivé grâce à la fonction recuperation_chemin (fonction commune) qui prend même en compte les correspondances et les changements de ligne pour le métro. Les stations de métro sont appelées par leur nom pas leur numéro dans notre programme. Enfin notre outil de compilation Makefile fonctionne.

Pour finir notre programme fonctionne sur les graphiques d'assez grande taille (cf analyse de performance).En effet grâce à l'outil Valgrind nous avons pu identifier et corriger les problèmes de mémoires. Cependant notre programme ne fonctionne pas sur le plus gros graphique (USACentral) , L'erreur est segmentation fault . La partie graphique cependant n'a pas abouti.

1b) choix des structures de données

Atteint est un tableau d'entier. Tandis que Attraiter est un liste chainée dynamique de sommet. En effet les listes se prêtent bien à l'ajout, la modification et la recherche du plus petit.

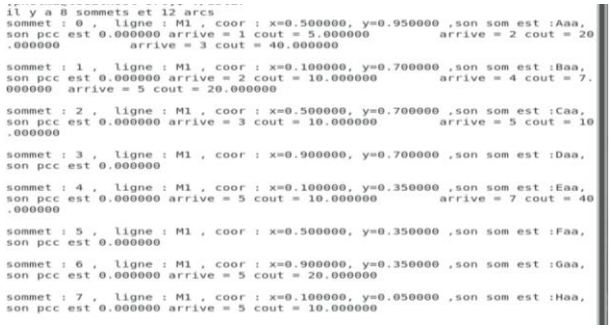
```
37
38  typedef struct _link {
39    sommet* som ; /* un élément de la liste*/
40    struct _link *next ; /* l'adresse du maillon suivant */
41  } *list_t, List ;
42
```

Pour ce qui est de structures principales du programme (graphe, sommet, maillon_arc) nous avons choisie d'adopter une syntaxe en français. A noter l'ajout d'un entier ,precedent, dans la structure d'un sommet qui permet de reconstituer le chemin. Pour retrouver le chemin sommet par sommet nous avons utilisons une structure Lifo. Pour chaque sommet, on empile le sommet puis son précèdent et on les dépile => on retrace le chemin dans l'ordre.

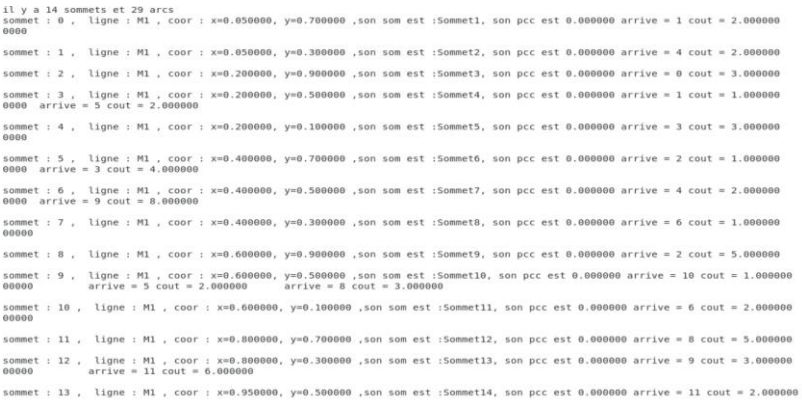
```
43  typedef struct _link2 {
44    int val; /* un élément de la liste*/
45    struct _link2 *next ; /* l'adresse du maillon suivant */
46  } *lifo_t ;
47
```

1c) Tests réalisées

A) Ouverture et lecture des graphes 1,2



Ouverture et affichage graphe 1



Ouverture et affichage graphe 2

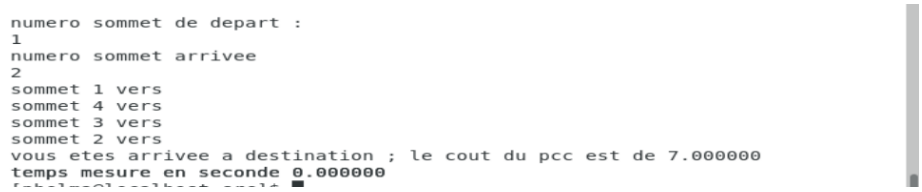
Conclusion : La fonction ouverture ouvre bien les données. La fonction print_graph nous retourne bien les données.

Nos fonction ouverture et graph_print fonctionnent !

B) Test de nos fonctions Dijkstra et Astar et de la reconstitution de chemin.

Test réalisé avec Astar sur le graphique 2.

Le plus court chemin entre 1 et 4 vaut 2. On emprunte l'arc 1->4 (cout 2). Le programme fonctionne ! La fonction récupération _chemin reconstitue aussi bien le chemin !



Test réalisé avec Astar sur le graphique 2.

Le plus court chemin entre le sommet n°1 et le n°2 vaut 7. En effet sur le fichier graphe2, on observe que pour aller au sommet 2, on prend l'arc 1->4 (cout 2), puis l'arc 4->3 (cout 3), enfin l'arc (3->2) (cout 2). On retrouve le PCC = 7 (=2+3+2).

Conclusion : Le programme fonctionne Astar.

```
nom de la station de depart :
La Défense
nom de la station d'arrivee :
Esplanade de la Défense
La Défense
 depart  = 0, arrivee = 1
sommet 0 vers
sommet 1 vers
vous etes arrivee a destination ; le cout du pcc est de 88.525368
```

Test réalisé avec Dijkstra sur les stations de métro

Pour aller de La défense à L'esplanade de la Défense on prend l'arc 0->1 (cout 88.52) .

Conclusion : Le programme s'adapte aux stations de métro et fonctionne avec Dikistra, la recherche des stations de métro par nom fonctionne aussi et le programme affiche les correspondances.

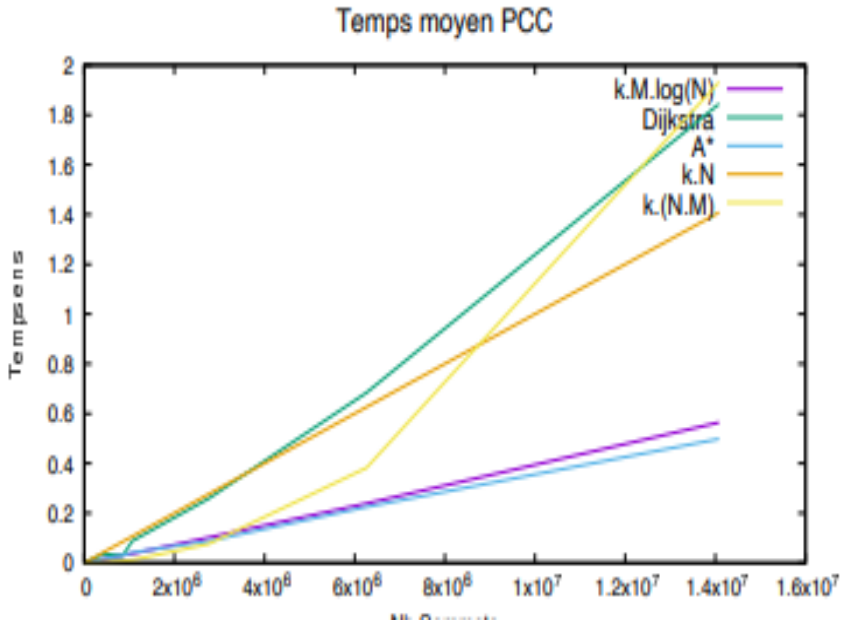
C) Test sur les gros fichiers

```
numero sommet de depart :
1
numero sommet arrivee
6
sommet 1 vers
sommet 0 vers
sommet 1362 vers
sommet 1357 vers
sommet 1354 vers
sommet 1273 vers
sommet 1182 vers
sommet 1180 vers
sommet 1181 vers
sommet 1178 vers
sommet 1176 vers
sommet 1175 vers
sommet 1177 vers
sommet 1153 vers
sommet 1152 vers
sommet 1071 vers
sommet 1147 vers
sommet 6 vers
vous etes arrivee a destination ; le cout du pcc est de 35129.000000
temps mesure en seconde 0.010000
```

Test réalisé avec Astar sur le graphique de New-York

Conclusion : Notre programme est fonctionnel même sur assez gros volume de données (Ne fonctionne tout de même pas sur les 3 plus gros graphiques, segmentation fault) . Le choix de nos structures et la réalisation des fonctions semble donc être pertinent (Le temps d'exécution est ici de seulement 0.01000 seconde).

1d) Analyse des performances



Nous avons testé les performances de nos programmes grâce à l'ajout d'un timer dans nos codes. On a bouclé un nombre i de fois des chemins de maniéré aléatoire et on a affiché la somme des temps d'exécution. On en déduit le temps moyen d'exécution de nos programmes Astar et Dijsktra pour les différents graphiques.

Astar/Dijkstra New-York 0.010 seconde (sommet 1->6)

Colorado 0.37000 s (sommet 1->6)

Floride 0.07000s (sommet 1->6)

Segmentation fault pour USAcentral

Pour le graphe Colorado, nous avons testé avec 100 et 1000 appels de fonction dont les sommets de departs et d'arrivés sont tirés aléatoirement. Pour dikstra , la durée totale est de 138 secondes pour 100 appels et de 1351 secondes pour 1000 appels. Pour Astar la durée totale est 142 secondes pour 100 appels et de 1419 secondes pour 1000 appels. On a donc des temps moyens d'environ 1,35 secondes par appel pour Dikstra et de 1,42 secondes par appelle pour Astar pour le graphique du colorado. On devrait des temps beaucoup plus court, (cela peut être dû au fait qu'on utilise les listes non triées et non des tas par exemple).

2.Suivi

2a) organisation du travail

Nous avons commencé par faire ensemble en séance le choix des structures et l'ouverture des fichiers. Ensuite nous nous somme réparti le travail en deux. Gabriel s'est occupé de réaliser la fonction Astar quant à moi j'ai réalisé la fonction Dijkstra. Nous avons ensuite adapté ensemble les fonctions sur les listes à nos structures. Enfin Gabriel à réaliser seul la partie reconstituant le chemin le plus court entre les sommets. En effet j'ai pris du retard sur la réalisation de la fonction Dijkstra

(une boucle qui n'atteigne pas sans condition d'arret sans raison apparente). Gabriel m'a aidé à passer cette difficulté et j'ai finalement réussis à corriger le tir en m'inspirant de ce qu'il avait fait pour la fonction Astar. Enfin les différents tests ont été réalisé chacun de notre côté de façon disparate avant une mise en commun des tests pour la réalisation du rapport. Pour nous avons tous les deux participé à la réalisation du Makefile pour compiler. Le travail a été réalisé en séance d'informatique puis majoritairement à distance en partageant nos codes avec gitlab.

2b) outils de développement utilisés

Tout d'abord nous avons utilisé le logiciel gitlab qui nous a permis de mettre facilement en commun nos différents code. De plus nous avons coder en C sur la machine virtuelle de phelma avec l'éditeur open source Atom. Enfin nous avons analyser notre code avec Valgrind pour éviter les fuites de mémoires et debugger nos programmes.

2c) Problèmes rencontrées et solutions

Nous avons eu quelques problèmes tout au long du projet. La fonction Liste_remove fonctionnait mal et nous avons passé beaucoup de temps à identifier l'erreur. Ensuite pour la fonction ouverture nous avons choisi d'utilisé des char[512] au lieu de char*, ce qui a considérablement réduit ses performances. De plus la fonction Dijkstra n'a pas fonctionné pendant un long moment. En effet elle ne sortait pas de la boucle (ligne 12 sur la présentation du projet),le pointeur sur les maillons arcs ne devenait jamais nul. Pour résoudre ce problème que nous n'avons pas réussis à vraiment expliquer, nous avons totalement remodelé la fonction Dijkstra en s'inspirant de la fonction Astar. Enfin nous avons eu beaucoup de problème de mémoires (segmentation fault) . Grâce à l'outil Valgrind nous avons identifié et corrigé la plupart soucis. Cependant un problème de mémoire demeure dans la fonction ouverture et nous n'arrivons pas à le résoudre. Ainsi les 3 plus grand graphiques ne fonctionnent pas avec notre programme. De plus nous avons essayé de faire une représentation graphique du graph avec la bibliothèque SDL de Phelma mais cette tentative n'a toujours pas abouti.

2d) ce que nous retenons du projet, travail à faire

Ce projet nous a permis de développer nos compétences en informatiques. En effet nous avons pu réaliser un programme assez complexe avec nos propres structures et fonctions. Nous avons appris ainsi, parfois à nos dépends, qu'il était vital de réaliser de tests réguliers en informatique et de bien organiser les codes et les fichiers du projet. De plus nous nous somme sensibiliser à l'utilisation de Valgrind et à la réalisation d'un fichier Makefile. Enfin nous avons appris à choisir nos structures, nos fonctions pour allouer la mémoire de maniéré pertinente et optimiser notre programme.

Le projet n'est cependant pas totalement abouti. Il nous faut encore réussir à ouvrir le plus gros graphique mais nous ne savons pas où est le problème. La partie graphique n'a pas abouti par manque de temps. La performance n'est pas optimale, on pourrait utiliser des tas au lieu des listes chainées non triées.

3.Conclusion

Pour conclure ce projet nous a permis d'appréhender les difficultés auxquels sont confronter les programmeurs. Ils doivent réaliser des choix pertinents pour optimiser leur programme et vérifier très régulièrement le fonctionnement de leur code. Nous avons acquis une expérience pour organiser et réaliser

des projets assez complexes. Nous regrettons cependant que le programme ne fonctionne pas pour les très gros fichiers et que la partie interface graphique n'est pas était réalisé.