

Brooklyn

Titouan
LOCATELLI

Guillaume
MERCIER

Hamza
OUHMANI

2022



Premier rapport de soutenance

Table des matières

1	Introduction	3
1.1	Introduction	3
1.2	Reprise cahier des charges	3
2	Présentation du travail	3
2.1	Guillaume	3
2.1.1	Structures et fichiers	3
2.1.2	Implémentation du premier algorithme(dijkstra)	5
2.1.3	Première fonction de retour	5
2.1.4	Tentative d'implémentation du tas de Fibonacci	6
2.1.5	Implémentation d'une file a priorité	6
2.1.6	Amélioration fonctions de retour	7
2.1.7	Objectifs pour la prochaine soutenance	7
2.2	Titouan	7
2.2.1	Graph.txt	8
2.2.2	Imprimer les resultats sur le terminal.	10
2.2.3	Objectifs pour la prochaine soutenance	12
2.3	Hamza	12
2.3.1	Pourquoi GTK et Glade?	13
2.3.2	Prise en main et lancement	13
2.3.3	Site Web	14
2.3.4	Objectifs pour la prochaine soutenance	14
3	Conclusion	15
3.1	Récapitulatif de qui a fait quoi	15
3.2	Conclusion	15

1 Introduction

1.1 Introduction

Nous sommes Guillaume Mercier, Hamza Ouhmani et Titouan Locatelli un groupe formé en vue d'accomplir le projet de S4 de l'EPITA. Nous formons le groupe Brooklyn, un nom qui ne renvoie pas à nous ou à nos personnalités mais à notre projet. Notre projet consistant à proposer une version simplifiée, mais modifiée de façon unique, de Google Map en est à ses débuts. Bien qu'il en soit à ses débuts, une quantité significative de travail a été accomplie. C'est pour cela que nous allons ici présenter notre travail accompli entre la remise du cahier des charges et cette soutenance.

1.2 Reprise cahier des charges

Dans le cahier des charges nous avons écrit vouloir une base technique viable, c'est-à-dire, les structures pour le graph, le graph représentant la carte d'une taille de 10 par 10, un algorithme pour donner le plus court chemin avec au minimum la marche et un prototype d'interface graphique. En comparaison à cela nous pouvons dire que nous avons bien avancé, en effet les structures sont toutes opérationnelles, la carte de base comprend tous les transports et fait 15 par 20. Ladite carte est stockée dans des fichiers qui sont capables d'être facilement lu par la suite. L'algorithme est opérationnel comme prévu mais avec tous les transports et de manière efficace. Nous sommes donc bien dans les temps par rapport à ce qui était prévu avec même un peu d'avance.

2 Présentation du travail

2.1 Guillaume

2.1.1 Structures et fichiers

La première partie de son travail a été d'implémenter les différentes structures et de décider du format des fichiers. Pour ce qui est des fichiers, nous avons besoin de charger une grille comprenant plusieurs modes de transports sous la forme d'un graph. Après avoir un peu réfléchi et expérimenté une fois le projet un peu avancé, j'ai choisi de représenter notre carte sous le format suivant. Premièrement un fichier comprenant tous les arcs de bases, ainsi que leurs poids, soit les chemins piétons, ce fichier aura pour en tête le nombre de nœuds du graph du graph. Dans le même dossier il y aura un fichier par moyen de transport soit le bus, le métro, le tram et le vélo, ces fichiers ne comprendront que la liste des arcs correspondant au mode de transport ainsi que le poids dudit arc.

Ensuite en ce qui concerne les structures de bases : il y en a deux, une pour les graph : graph, et une pour les nœuds : node. node est définie avec les parametres suivant :

```
size_t vertex  
struct node* next  
size_t weight  
int transport
```

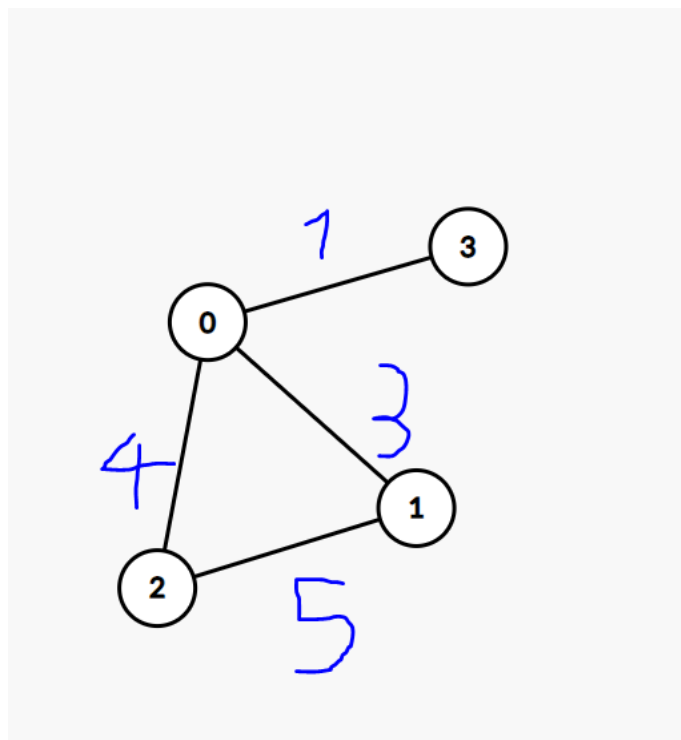
vertex est le numéro du nœud dans le graph. next est un pointeur vers le prochain nœud de la liste d'adjacence. weight est le poid de l'arc entre le nœud source et celui ci. Le transport représente le type de transport de l'arc entre le la source et ce nœud.

graph est défini avec les paramètres suivants :

```
size_t order  
struct node** adjlists
```

order est bien entendu le nombre de nœuds dans le graph. adjlists est une liste ou l'index i pointe vers une liste chaînée de noeuds représentant les voisins du noeud i.

Prenons l'exemple suivant :



Dans le graph ci dessus, la liste chaînée de adjlists[0] comprendra 3 éléments, s'ils sont dans l'ordre numérique, adjlists[0] pointrai vers un élément node dont le vertex est 1 et le poid 3, celui ci pointrai vers un élément node dont le vertex

est 2 et le poids 4, ect... Il est important de noter que ces éléments nodes ne sont pas les mêmes pour tout le monde, par exemple, l'élément ayant pour vertex 2 dans les voisins 0 n'est pas le même que celui des voisins de 1, en effet le poids est différent.

2.1.2 Implémentation du premier algorithme(dijkstra)

Après quelque recherche sur les algorithmes de plus court chemin, j'en ai conclu que le plus approprié pour notre projet, au moins dans sa version initiale est l'algorithme de dijkstra, en raison du fait que nous ne cherchions que le plus court chemin d'un point à un autre. Cependant il faudra peut-être considérer changer de l'algorithme quand nous rajouterons plusieurs points de destination, notamment l'algorithme Floyd-Warshall semble plus adapté à ce problème.

La fonction dijkstra est l'implémentation de l'algorithme pour notre problème, le prototype est le suivant

```
struct node* dijkstra(struct graph* graph, size_t source, size_t destination);
```

La fonction prend donc en paramètre le graph, le nœud d'origine du chemin à chercher et le nœud d'arrivée.

La première étape est d'enfiler tous les index des nœuds du graph a une file a priorité (voir plus loin pour l'implémentation), tous les nœuds auront initialement une priorité virtuellement infini, sauf la source pour qui elle sera de 0. Il faut également créer une liste "distances" tel que l'index représente le coût du chemin entre l'origine et le noeud i. Et enfin créer une liste "list_prev" tel que l'index i de la liste renvoie le noeud précédent a i dans le plus court chemin.

Ensuite l'algorithme va, tant que la file n'est pas vide, défiler le nœud n avec la priorité la plus basse et parcourir tous ses voisins. Pour chaque voisin sa distance devient le minimum entre sa distance actuelle et la somme de la distance de n plus le poids de l'arc, la priorité du voisin est changé si besoin.

Si le nœud défilé est la destination, alors l'algorithme appelle la fonction de retour puis s'arrête.

2.1.3 Première fonction de retour

La première implémentation de la fonction de retour prenait en paramètre la liste de précédent, la source et la destination. Tant que la destination a un précédent il ajoutait la destination à une liste chaînée et la destination devenait son précédent. A la fin il fallait inverser la liste chaînée pour avoir le chemin dans le bon ordre. Cette version était bien entendu sous optimale à cause de l'inversion mais également du fait qu'elle ne travaillait qu'avec les index des nœuds, ainsi les informations telles que le poids ou la méthode de transport étaient perdues.

2.1.4 Tentative d'implémentation du tas de Fibonacci

Après quelques recherches sur comment améliorer l'algorithme de Dijkstra, la première implémentation que j'ai trouvée était d'utiliser une file à priorité (comme dit précédemment) et selon wikipedia l'une des plus performante est l'implémentation passant par le tas de Fibonacci, je me suis donc fixé comme but de l'implémenter. L'implémentation du tas a demandé beaucoup de temps et d'effort car la structure est assez compliquée à installer correctement. Finalement la structure est presque fonctionnelle mais sur les graphes de grande taille elle crée une erreur mémoire, de plus après de plus ample recherche je me suis rendu compte que ce n'était sûrement pas la structure la plus adaptée. En premier lieu à cause du coup initial pour créer la pile, il faut initialiser une grande quantité de pointeurs et les fonctions notamment pour consolider le tas après un changement de priorité sont lourdes. Tout cela crée un coup initial très grand qui ne sera compensée que lors de l'utilisation de grand graphe. J'ai donc pour l'instant laissé tomber cette implémentation des listes à priorité mais si dans le futur j'en ai besoin car la taille des données augmentent, elle est presque opérationnelle.

2.1.5 Implémentation d'une file à priorité

Après l'abandon du tas de Fibonacci je me suis tourné vers une liste à priorité linéaire, il s'agit d'une liste chaînée dont chaque élément contient la priorité, elle est donc triée par ordre croissant. La structure nommée `queue_elt` contient les attributs suivant :

`size_t node`

`size_t priority`

`struct queue_elt* next`

`node` est l'index du nœud associé, `priority` la priorité dans la liste et `next` est un pointeur vers le prochain élément de la liste chaînée. Pour la faire fonctionner, plusieurs fonctions sont implémentées, une fonction d'insertion "insert" de prototype :

```
void insert(struct queue_elt** queue, size_t node, size_t priority);
```

Elle crée un élément de liste qui aura comme priorité "priority" et comme index de node "node", puis elle place cet élément à l'endroit où le suivant a une priorité supérieure ou égale à la sienne.

Ensuite il y a la fonction `extract_min` qui renvoie l'index de l'élément avec la plus faible priorité et le supprime de la file.

Enfin il y a la fonction `change_prio` qui a pour prototype :

```
void change_prio(struct queue_elt** queue, size_t node, size_t new_prio);
```

Cette fonction va changer la priorité de l'élément ayant comme index de nœud "node", pour cela elle va supprimer le nœud en question puis appeler la fonction `insert` pour le replacer à la bonne place.

2.1.6 Amélioration fonctions de retour

Comme dit précédemment une fois que l'algorithme de Dijkstra à trouver sa destination il va appeler la fonction de retour, la version de final de celle ci a pour prototype :

```
struct node* build_return(struct graph* graph, size_t source, size_t destination, size_t* list_prev);
```

Elle fonctionne globalement comme la précédente sauf qu'elle crée directement la liste chaînée dans le bon ordre en jouant avec les pointeurs et qu'elle récupère le poids et la méthode de transport des chemins. Pour cela elle appelle une fonction build_node qui va parcourir les listes d'adjacences pour récupérer les informations demandées. Cela pourrait être très coûteux si notre graph avait beaucoup d'arêtes par nœud mais heureusement ce n'est pas le cas.

Par exemple sur notre graphe principal détaillé plus tard par Titouan, le chemin le plus court du point 0 au point 299, donc du coin en haut à gauche au coin en bas à droite, donné par les fonctions est le suivant

```
Guillaume_tests $ ./main graphs/main/ 0 299
-(-1)->0-(0)->20-(0)->40-(0)->60-(0)->80-(2)->107-(2)->131-(2)->249-(0)->250-(1)->291-(1)->298-(0)->299
total time: 0.003971
```

Le temps est donné grâce à la librairie time.h, on peut donc voir que sur un petit graphe comme celui ci, même chercher dans l'un des pire cas est extrêmement rapide. Le chemin rendu donne le mode de transport entre les parenthèses, 0 est pour la marche, 1 le bus et 2 le métro, cela est définie par des macros dans un fichier à part.

2.1.7 Objectifs pour la prochaine soutenance

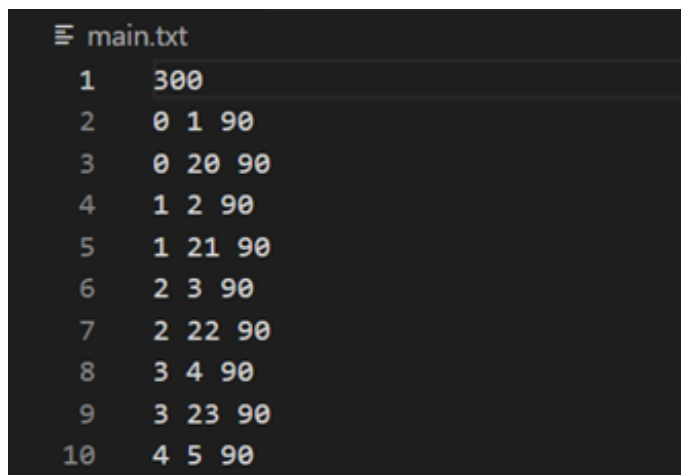
Pour la prochaine soutenance, j'ai un objectif principal, il faut que je travaille avec Titouan pour implémenter la possibilité de rajouter plusieurs destinations. Cela n'est pas facile ne soit mais il va probablement modifier ce qui existe déjà pour permettre cette fonctionnalité, je pense par exemple a l'algorithme de Dijkstra qui me semble peu adapté à ce problème. Dans tous les cas, une grande quantité de recherches vont être nécessaires pour permettre une implémentation correcte.

2.2 Titouan

J'ai décider de commencer le travail un peu plus tôt pour ce projet, pour éviter le travail dernière minute. Le début était évident, avec Guillaume on s'est divisé le début, il codait la classe graph et je me chargeais de coder le graph initial, nous avons décidé pour une taille initiale de travailler sur un 20*15, soit 300 nœuds.

2.2.1 Graph.txt

On a décidé de s'y prendre un peu comme dans les DM d'algo. J'ai donc codé une fonction assez simple qui met dans un fichier texte un graph. Donc nos graphs sont stockés en fichiers texte, manière simple de stockage et rapide à récupérer le graphe de celui-ci.



```
main.txt
1 300
2 0 1 90
3 0 20 90
4 1 2 90
5 1 21 90
6 2 3 90
7 2 22 90
8 3 4 90
9 3 23 90
10 4 5 90
```

Première ligne est un int, celui-ci est la quantité de nœuds que notre graphe a. A partir de la deuxième ligne : nous avons les liens entre les nœuds. Le premier chiffre (ex de la deuxième ligne : 0) est l'origine de l'arc et le deuxième chiffre (1) est la destination de l'arc. Nous travaillons avec des graphs pondérés, donc le troisième nombre (90) est le poids de l'arc.

En parlant de graphs pondérés : Nous cherchons à représenter plusieurs moyens de transports sur notre carte, car le but est de prendre en compte plusieurs moyens de transports, ou tout simplement marcher, pour ce rendre d'un point A à B. Nous avons donc associé un poids à chaque moyen de transport. Nous les avons choisis en fonction des vitesses de ces transports dans la vie réelle. Ceux-ci sont donc :

- Walking : 90
- Bike : 30
- Tram : 18
- Bus : 24
- Metro : est calculé différemment.

Tous les autres transports se déplacent sur les routes (c'est-à-dire les nœuds de base de notre grille) de notre grille de 20*15. Donc leurs poids représentent le

temps en secondes pour parcourir cent mètres. Donc pour une station de bus a 500 mètres de distance (5 nœuds de distance) de l'autre, le poids de l'arc reliant ces deux stations aura un poids de $5 \times 24 = 120$.

Le métro, contrairement, ce déplace sous terre. Donc nous avons calculé la distance (pas la distance Brooklyn cette fois) sur notre carte et associé un poids indique a chaque arc reliant deux stations.

J'ai donc procédé à dessiner une carte sur papier de lignes de métro, bus, tram, j'ai placé un paquet de stations de vélos etc. . . Tout cela pour que la carte est un peu un sens, qu'ils ne soient pas générés au hasard. Nous avons décidé avec Guillaume de faire des fichiers séparés pour les transports. Il y avait donc le fichier main qui était la grille avec chaque arc avec le poids à pied (=90). Donc tous les fichiers textes des transports étaient séparés du fichier texte main.

En fonction de la carte que j'ai dessiné sur papier. J'ai implémenté à la main dans chaque fichier texte respective, les arcs reliant les moyens de transports. Sauf pour les vélos. Sur ma carte j'avais à peine représenté les stations de vélo. J'ai donc codé une fonction qui prenais le numéro du nœuds et créait un nouveau fichier texte avec tous les arcs entre ce nœuds stations. Nous pensons que c'est la manière la plus simple d'implémenter les vélos pour notre projet. Nous avons médité quelque temps avec Guillaume sur le sujet, nous avons notamment pensé à réutiliser la grille main (celle de marche), mais à rajouter un paramètre d'activation qui diviserait le temps de transports (pour atteindre celle du vélo) jusqu'à atteindre une nouvelle station vélo, mais bon bref, c'était trop compliqué, et probablement beaucoup plus couteux.

```
≡ bikes.txt
1 4 13 270
2 4 20 150
3 4 30 240
4 4 38 480
5 4 46 150
6 4 62 180
7 4 75 480
8 4 89 300
9 4 123 270
10 4 131 450
```

Voici en exemple le fichier texte de vélo avec (par ligne) en premier le nœud d'origine, puis le nœud d'arrivé et le poids (proportionnel a la distance entre les

deux stations de vélo). 4 est ici le départ sur les dix premières lignes car nous avons liés toutes les stations entre elles, donc il y a énormément d'arcs vélos. Nous pourrions rajouter une contrainte, un peu comme celle de velov, où on a le droit qu'à 30 minutes de déplacement (mais bon, notre carte est trop petite encore pour cela).

2.2.2 Imprimer les resultats sur le terminal.

J'ai pris la tache de l'affichage de notre parcours de graphe avec l'algorithme Dijkstra. Pour la première soutenance je vais afficher ces résultats sur un terminal, puis Hamza prendra le relais et son programme gtk affichera les résultats. Nos graphes ont été implémenter en C, après de la recherche en ligne, les voisins d'un nœud seraient donc mis dans une liste chaînée. Et donc, nous avons mis le résultat de notre parcours de Dijkstra sous forme de liste chaîné.

Ex : pour aller de nœud 1 a nœud 46 :

1->2->6->5->45->46

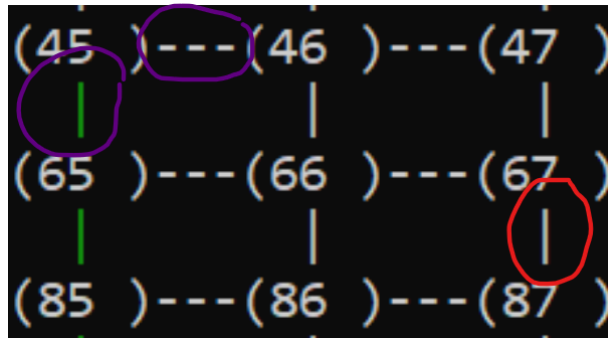
Il va parcourir 5 nœuds avant d'arriver à 45. Je vais d'abord analyser les modes de transport que l'algorithme de Dijkstra a empreinté. Disons que :

- Walking : 0
- Bus = 1
- Metro = 2
- Bike = 3
- Tram = 4

Et je vais voir comment ce déplace donc notre utilisateur du programme avec entre guillemets le mode de transport empreinté durant le chemin :

1—(walk)->2—(bus)->6—(walk)->5—(bike)->45—(walk)->46

Je vais donc chercher à compléter ce chemin (car j'imprime une grille et je n'imprime que les arcs de la grille – pas d'arcs qui vont d'un coin à l'autre de la grille en passant par le milieu ou par l'extérieur. . . peut être sur gtk après) en simulant un peu comme si le bonhomme qui parcourt la grille marchais plus vite ou plus lentement par moments. En effet en dessous par exemple, je n'imprime que les arcs à droite et en dessous de chaque noeud (sauf évidemment pour ceux sur les extrémités du graphe, 67, qui lui a un seul arc (en rouge) qui l'a pour origine).



La liste complétée donne donc :

1—(walk)->2—(bus)->3—(bus)->4—(bus)->5—(bus)->6—(walk)->5—(velo)->25—(velo)->45—(walk)->46

(Je n'ai pas mis l'exemple du métro car celui-ci passe sous terre et je ne vais chercher à compléter son chemin -> j'explique dans la suite)

J'imprime donc la grille dans le terminal en prenant en compte le chemin complété en imprimant les chemins d'une couleur particulière au mode de transport empreinté durant le passage sur ce chemin.

```

7  #define CWALK  "\x1B[31m" // red
8  #define CMETRO "\x1B[34m" // blue
9  #define CBIKE  "\x1B[33m" // yellow
10 #define CBUS   "\x1B[32m" // green
11 #define CTRAM  "\x1B[35m" // magenta
12 #define CNORMAL "\x1B[0m" // default

```

Le métro est en bleu (voir ci-dessus) mais je n'imprime pas les chemins souterrains qu'empreinte de métro, je mets plutôt la station de départ et d'arrivée du métro en bleu, c'est-à-dire les nœuds sont imprimés en bleu sur le terminal.

Ci-dessous un exemple d'un extrait de parcours du graph imprimé sur le terminal, le graphe n'est pas affiché en entiers, ça serait un peu dur à voir, mais on visualise bien le chemin avec des couleurs et une station de métro empreinté en bleu.

```
( 0 )---( 1 )---( 2 )---( 3 )---( 4 )---( 5 )---( 6 )---( 7 )---( 8 )---( 9 )---(10 )---(11 )---(12 )---
(20 )---(21 )---(22 )---(23 )---(24 )---(25 )---(26 )---(27 )---(28 )---(29 )---(30 )---(31 )---(32 )---
(40 )---(41 )---(42 )---(43 )---(44 )---(45 )---(46 )---(47 )---(48 )---(49 )---(50 )---(51 )---(52 )---
(60 )---(61 )---(62 )---(63 )---(64 )---(65 )---(66 )---(67 )---(68 )---(69 )---(70 )---(71 )---(72 )---
(80 )---(81 )---(82 )---(83 )---(84 )---(85 )---(86 )---(87 )---(88 )---(89 )---(90 )---(91 )---(92 )---
(100)---(101)---(102)---(103)---(104)---(105)---(106)---(107)---(108)---(109)---(110)---(111)---(112)---
(120)---(121)---(122)---(123)---(124)---(125)---(126)---(127)---(128)---(129)---(130)---(131)---(132)---
(140)---(141)---(142)---(143)---(144)---(145)---(146)---(147)---(148)---(149)---(150)---(151)---(152)---
(160)---(161)---(162)---(163)---(164)---(165)---(166)---(167)---(168)---(169)---(170)---(171)---(172)---
(180)---(181)---(182)---(183)---(184)---(185)---(186)---(187)---(188)---(189)---(190)---(191)---(192)---
(200)---(201)---(202)---(203)---(204)---(205)---(206)---(207)---(208)---(209)---(210)---(211)---(212)---
(220)---(221)---(222)---(223)---(224)---(225)---(226)---(227)---(228)---(229)---(230)---(231)---(232)---
(240)---(241)---(242)---(243)---(244)---(245)---(246)---(247)---(248)---(249)---(250)---(251)---(252)---
(260)---(261)---(262)---(263)---(264)---(265)---(266)---(267)---(268)---(269)---(270)---(271)---(272)---
(280)---(281)---(282)---(283)---(284)---(285)---(286)---(287)---(288)---(289)---(290)---(291)---(292)---
projetS4 $ _
```

2.2.3 Objectifs pour la prochaine soutenance

J'aimerais travailler plus directement avec Guillaume pour implémenter la partie plus algorithmiquement complexe, qui est d'avoir plusieurs objectifs à atteindre sur la carte. Nous pensions augmenter la taille de la carte, pour augmenter les temps et la complexité pour l'algorithme de recherche, notamment pour remarquer plus facilement nos améliorations et arriver plus rapidement à des solutions optimisées en fonction du temps.

Je communiquerai plus avec Hamza pour lui rendre des résultats simples qu'il pourra à son tour exploité pour imprimer sur sa fenêtre graphique gtk.

2.3 Hamza

En ce qui me concerne je me charge principalement de la partie interface graphique et du site web, ce qui implique toutes les interactions de l'utilisateur avec les différents algorithmes mis en place. Comme il a été expliqué dans le cahier des charges précédemment validé le but ici est de faire l'interface la plus intuitive et élégante possible pour l'utilisateur, pour cela j'ai opté pour la version 3.0 de GTK et Glade, un outil d'interface graphique.

2.3.1 Pourquoi GTK et Glade ?

Gérer la partie interface graphique d'un projet est une absolue nouveauté pour ma part je ne m'étais jamais occupé de cette partie et c'est une chance que j'ai de découvrir cet aspect important dans un projet.

Lors des mes premières recherches sur les différentes possibilités d'une mise en place d'interface graphique en C j'ai eu plusieurs résultats qui me renvoyaient vers GTK et étant donné que c'était une bibliothèque que nous avions déjà utilisée lors du précédent semestre mon choix s'est naturellement porté sur celle-ci.

Glade quand à lui est un outil qui permet de prendre en charge toute la partie de génération et de gestion de l'interface pour permettre de se concentrer uniquement sur les notions essentielles.

2.3.2 Prise en main et lancement

C'est ici que vont commencer pour moi les déboires de l'interface graphique, lors de l'installation et de l'utilisation de la bibliothèque GTK et de Glade. En ce qui concernait le TP du dernier semestre je l'avais fait sur les ordinateurs de l'école ce qui n'avait pas posé de problèmes alors j'ai tenté d'installer GTK ainsi que Glade sur la machine virtuelle que j'utilise régulièrement (Kali), ce qui s'est soldé par un premier échec.

N'arrivant pas à installer mes outils sous Linux je me suis redirigé vers Windows en pensant que la tâche serait plus simple, avec du recul j'ai été naïf. Pour opérer sur Windows j'ai d'abord installé Mysys2 qui permet de fournir un environnement similaire à Unix pour Windows.

Arrivé à cette étape, 2 choix m'étaient présentés, soit passer par CodeBlocks qui est un environnement de développement intégré libre et multi-plateforme qui se caractérise par les multiples possibilités de configuration qui auraient pu me permettre d'utiliser efficacement la librairie GTK ou de choisir Glade. Dans un premier temps j'ai d'abord tenté d'installer CodeBlocks mais ça n'a pas été fructueux. Je me suis alors tourné vers Glade qui était ma solution de départ mais malheureusement malgré le bon lancement de l'outil tout le travail effectué est à chaque fois irrécupérable ou corrompu voir même subi des crash pendant le développement.

J'ai tenté d'installer une version d'Ubuntu sur Windows qui me permettrait de pouvoir profiter de l'environnement Linux (pas sous format machine virtuelle) sur Windows afin de directement pouvoir travailler dessus mais cette tentative aura été sans succès aussi. J'aurai compris que mon entêtement à vouloir essayer de trouver la solution tout seul n'aura servi à rien et n'aura fait que retarder un

aspect du projet pour notre groupe, il faudra que j'apprenne à demander plus souvent de l'aide notamment aux ASM qui sont à notre disposition.

2.3.3 Site Web

Comme je l'ai dit précédemment ce projet est rempli de nouveautés pour moi car c'est aussi la première fois que je m'occupe de créer un site web qui permettra de représenter l'état d'avancement de notre et l'évolution de notre travail. Mon choix s'est logiquement porté sur l'HTML afin de gérer cette partie-là étant donné la quantité importante de documentation disponible sur internet.

Contrairement à l'interface graphique où l'installation et le lancement des différents outils choisis a été un cauchemar, cette étape en ce qui concerne le site web aura été un rêve étant donné qu'il était possible de gérer directement sur Bloc-Notes pour le début puis de passer à l'éditeur SublimText afin d'avoir plus de couleurs et d'options disponibles lors du codage.

Etant donné que j'ai malheureusement accordé trop de temps sur mon temps de projet à tenter d'installer les outils de l'interface graphique je n'ai pas pu pour l'instant pleinement prendre conscience de toutes les capacités qu'offrent l'HTML c'est pour cela que je me suis dans un premier temps concentré sur l'édition de texte dans un premier temps dans l'HTML. C'est pour cela que pour l'instant notre site web correspond actuellement à une tentative de reprise dans le même format d'édition de notre rapport de soutenance.

2.3.4 Objectifs pour la prochaine soutenance

Malgré les différentes mésaventures que j'ai eu lors du lancement de l'interface graphique j'ai eu le temps étant donné les ressources disponibles par rapport à la bibliothèque GTK d'engranger une bonne quantité d'informations qui me permettront après avoir réglé j'espère pendant les vacances les soucis que j'ai afin de pouvoir reprendre directement le travail après la période des midterms afin de combler le retard accumulé sur ma partie de pouvoir offrir une interface à la hauteur des efforts fournis par mes collègues pour ce projet.

En ce qui concerne le site web, je compte accorder plus de temps à la documentation par rapport à l'HTML afin de pouvoir travailler non plus seulement sur de l'édition de texte mais aussi la possibilité d'ouvrir plusieurs fenêtres (par exemple pour le développement individuel de chacun dans ce projet) et de télécharger notre application ainsi que les différents rapports de soutenance.

3 Conclusion

3.1 Récapitulatif de qui a fait quoi

Tâches	Guillaume	Titouan	Hamza
Structures de données	x		
Création des fichiers de la carte		x	
Chargement des fichiers	x		
Implémentation de Dijkstra	x		
Implémentation des file a priorité	x		
Ajout des modes transports sur la carte		x	
Affichage des résultats sur le terminal		x	
Site web			x

3.2 Conclusion

La partie mis en place des types, créations de graphes étaient l'attente principale pour cette soutenance, et nous avons bien dépassé cela. Guillaume a mis en place un algorithme de parc ou du graph, et est déjà en train d'améliorer celui-ci. Titouan a implémenté tous les transports en commun au graph et affiche les résultats. Hamza a eu des complications de compatibilités avec gtk et son ordinateur mais a fait ses recherches sur gtk et est prêt à coder l'interface graphique. Nous pensons en général avoir de l'avance sur le projet, mais le plus dur reste encore à faire : faire un algorithme optimisé pour atteindre plusieurs points sur le graphe en un seul calcul et/ou rajouter plus de contraintes au projet.