Rapport du projet Metro

Clément Caumes 21501810 - Yassin Doudouh 21500127

05 mai 2017

1 Description des structures de données

Les déclarations de ces structures de données sont dans *graphe.h*.

La structure **SOMMET** contient les informations d'un sommet. Il y a le nom de la station, le numero du sommet dans metro.txt, le numero de la ligne dans le réseau du metro parisien, la géolocalisation sur la fenêtre graphique de l'application.

La structure **ARC** contient les informations d'un arc entre deux sommets, avec le numéro du sommet du terminus et la pondération de l'arc (temps qu'il faut pour aller d'une station à une autre).

La structure **DIJKSTRA**, qui stockera le plus court chemin calculé avec l'algorithme de Dijkstra, contient le numéro du sommet de départ, le numéro du sommet d'arrivée. Il y a également la durée qui sera calculée dans cree chemin et la liste des stations qu'il faut traverser pour aller du départ à la destination voulu.

La structure **GRAPHE** contient toutes les informations du graphe de l'application : un tableau de 2 sommets avec les 2 sommets choisis par l'utilisateur pour le calcul du plus court chemin, un entier représentant le nombre de sommets dans le graphe, un tableau des sommets avec leur nom et leur numéro de ligne, et un tableau à 2 dimensions où chaque arc est déterminés avec pondération.

La structure **METRO** contient toutes les informations relatives à l'application : le graphe qui représente le metro parisien, l'affichage (si on zoom ou non) et le mode (si on a quitté l'application).

2 Préparation du projet

L'application que nous devions créer devait permettre de choisir des stations de metro au clic afin de calculer le plus court chemin entre ces stations.
Nous avons rajouté quelques données dans le fichier metro.txt. En effet, il était

intéressant de connaître les lignes de chaque station ainsi que leur position sur la carte. Les terminus pour chaque arc étaient également une donnée intéressante pour connaître les directions.

3 Fonctionnement général

Tout d'abord, on initialise le graphe du metro. On alloue le bon nombre de sommets grâce à la lecture dans le fichier metro.txt. Les arcs sont nuls pour le moment. Cela correspond à la procédure *initialise_metro* dans *graphe.c*.

Le tableau des sommets est alors initialisé pour contenir les informations de chaque station (le nom de la station, le numéro du sommet, le numéro de sa ligne et ses coordonnées sur l'interface graphique). Cette procédure est faite par *initialise_stations* avec la lecture de metro.txt dans *lire_ecrire.c*.

Le tableau des arcs entre les sommets est aussi initialisé avec le numéro du terminus (direction de l'arc sur la ligne) et la pondération de cet arc. Sachant que le metro est à double sens, l'arc opposé est également initialisé avec la même pondération mais avec le terminus opposé. *initialise_reseau* fait ce travail grâce à metro.txt dans *lire_ecrire.c*.

En fonction de ce que veut faire l'utilisateur, il y a plusieurs manipulations (avec *interaction* de *affichage.c*) :

- cliquer sur la touche '**Q**' pour quitter.
- cliquer sur la touche '**Z**' pour zoomer. Ici, les flèches directionnelles sont utilisables pour se déplacer. Recliquer sur '**Z**' permet de dézoomer.
- le clic sur les différents sommets permet de sélectionner/désélectionner les stations que l'on a choisi pour calculer le plus court chemin (avec *ajoute_supprime_selection_sommet* de *affichage.c*). Ce chemin est calculé quand l'utilisateur a choisi 2 sommets et l'application utilise l'algorithme de Dijkstra avec *calcul_dijkstra* de *graphe.c*. Quand le tableau des successeurs de Dijkstra a été initialisé, on crée une liste de sommets par lesquels il faut passer pour avoir le plus court chemin.

Le plus court chemin est maintenant trouvé. *ecrit_chemin* de *lire_ecrire.c* écrit dans le terminal les stations par lesquelles il faut passer et le temps qu'il faut. Les stations sur la carte sont mises en valeur en jaune avec *affiche_chemin* dans affichage.c.

Finalement, la mémoire est libérée quand on quitte l'application avec *libere_graphe*, *libere_chemin_dijkstra* et *libere_fenetre*.

4 Principale procédure : Algorithme Dijkstra

A première vue, l'algorithme qui était le plus adapté pour calculer le plus court chemin entre des stations de metro était celui de Floyd-Warshall car il calcule entre toutes les paires de sommets. Pourtant, sa complexité est de o(n^3) avec n le nombre de sommets.
Mais, l'algorithme de Dijkstra a une complexité de o(n^2) : à chaque itération on traite un sommet (o(n)) et à chaque traitement de sommet, on parcourt le tableau des plus courtes distances (o(n)). Et s'il fallait atteindre la même que Floyd-Warshall, il aurait fallu n fois chemins différents o($n^2 \times n$) : ce qui n'est pas réaliste pour 375 stations.
C'est pour cette raison que nous avons choisi Dijkstra pour calculer le plus court chemin entre stations.

Son implémentation est la suivante :

— d'abord, on stocke le fait que aucun sommet n'ait été traité, aucun sommet n'ait encore de père, les plus petites distances sont encore infinis.
— on traite le sommet de départ et on met à jour les plus petites distances avec ses successeurs en utilisant les pondérations.
— tant qu'on a des sommets à traiter, on prend la plus petite des plus petites distances et on regarde si on améliore les plus petites distances de ces successeurs en passant par ce sommet. Si c'est le cas alors ce dernier devient son père.
— le tableau des pères est alors initialisé complétement.

5 Améliorations

Lorsque l'on est dans le mode zoom, on ne peut pas sélectionner de stations car les coordonnées ont changées. En effet, il aurait été difficile de recalculer les coordonnées de chaque station puisque la division de la carte n'est pas parfaitement symétrique. On aurait pu rencontrer des problèmes de sélections (décalages de coordonnées).