Rapport : Astar

CHAHRIAR Benoît DU BUAT Victor

2019-06-03

1 Implantation

1.1 Etat logiciel

Dans l'état actuel du programme voici les fonctionnalités implémentées:

- Ouverture du fichier et lecture du graphe
- Fonctions élémentaires de gestion de liste chainée
- Fonctions élémentaires de gestion de tas (dans un tableau)
- Trouver un chemin via Astar à partir de numéros de sommets
- Transformer le coup arbitraire des arcs du graphe de métro
- La table de hachage pour rentrer le nom d'un sommet

Voici les fonctionnalités non implémentées, ce qui ne fonctionne pas:

- Trouver le meilleur chemin possible via Astar (en comparaison avec le programme solution fourni).
- Affichage graphique

1.2 Tests effectués

Astar et ses différents sous modules ont étés testés dans les conditions détaillées ci-dessous.

- Astar : graphe2.txt , grapheNewYork. Astar ne trouve pas la bonne solution sur notre chemin test (sommet 0 à 10). On soupçonne que l'erreur se situe probablement au niveau des tas mais nous n'avons pas pu l'identifier
- Liste : tests de chaque fonction dans un main simple
- Tas : test dans "testtas.c" avec un petit tas artificiel manipulé à la main
- Table de hachage : testée dans un petit main

1.3 Exemple d'exécution

Exécution sur notre chemin test New York de 0 à 10 :

```
Quel est votre point de depart
Sommets1
8Quel est votre point d'arrivee
Sommets11
9arrivee = 10
arrivee = 8
arrivee = 602
arrivee = 593
....
....
i = Sommets595
i = Sommets594
i = Sommets603
i = Sommets9
i = Sommets11

on arrive à 10
 avec un cout de 93248.000000
```

Exécution sur graphe2 :

```
Quel est votre point de depart
Sommet1
7Quel est votre point d'arrivee
Sommet11
8arrivee = 10
arrivee = 7
arrivee = 4
arrivee = 1
Chemin à emprunter:
i = Sommet2
i = Sommet5
i = Sommet8
i = Sommet11

on arrive à 10
 avec un cout de 12.000000
```

2 Suivi

2.1 Problèmes rencontrés

La gestion des tas a été compliquée pour prendre en compte tous les cas particuliers. Mon (Benoît) retard a ralenti le debug de la première écriture de Astar qui a été conséquent et a mis en évidence d'autres problèmes sur les tas. On a aussi découvert les difficultés liées au debug et test d'un programme un peu plus conséquent qu'un TD notamment la plus grande difficulté à isoler les problèmes et anticiper tous les cas possibles.

2.2 Planning effectif et répartition

- Gestion des tas : Benoît , semaine 0 à 5
- Makefile : Benoît, semaine 1
- Ouverture fichier, lecture graphe : Victor, semaine 0 à 2
- Astar : écriture et debug conjoint, semaine 2 à 5
- Table de hachage: Victor, semaine 5

3 Conclusion

Ne pas avoir pu livrer un programme totalement fonctionnel est source de grande frustration pour notre groupe. En particulier je (Benoît) prends une certaine responsabilité étant donné que je soupçonne fortement le problème de venir de la gestion de tas dont j'étais responsable mais malgré de nombreuses relectures nous n'avons pas trouvé l'erreur à la rédaction de ce rapport.

4 En détails

4.1 Les graphes

Comme proposé par l'énoncé, les graphes ont été définis comme un tableau de T_SOMMET. Le type T_SOMMET est une structure englobant toutes les informations attachées à un sommet utiles à l'algorithme. Pour ce fait il comporte nécessairement les informations fournies par le document texte :

- nom du sommet : chaine de caractère donc char*
- nom de la ligne de métro : chaine de caractère donc char*
- coordonnées : deux réels théoriquement donc deux doubles dans les faits
- la liste des voisins

On remarquera que le numéro du sommet n'est pas inclus comme il s'agit de sa position dans le tas. Afin de faciliter le fonctionnement de A* nous avons également stocké les informations suivantes :

- ListeFermee : entier valant 0 ou 1 indiquant si le sommet appartient à la liste fermée
- Liste Ouverte : entier valant 0 ou 1 position dans le tas indiquant si le sommet appartient à la liste ouverte.
- père : entier indiquant le rang dans le graphe du père de ce sommet(au vue de l'avancement de A*)
- G flottant double précision indiquant le cout nécessaire pour accéder à ce sommet depuis le sommet de départ en l'état actuel de A* Afin de pouvoir gérer les liens entre les sommets, nous avons eu recours à une structure de liste permettant de créer dynamiquement les liens après la création du graphe. L_ARC contient l'adresse d'un autre lien partant du même sommet et une structure nommée T_ARC contenant les informations relatives au lien : cout et destination.

Une fois ces structures définies il est nécessaire de définir les fonctions permettant leur utilisation :

1. création du graphe par création_graphe(int nbsommet)
2. initialisation d'un sommet à partir des informations issues du document texte : creation_sommet(T_SOMMET* psommet,char* nom,char* line, double longi, double lat). Les informations non fournies par le graphe car étant mise à jour au fur et à mesure du fonctionnement de A* sont initialisées comme suit :
   - ListeFermee=0 : la liste est vide initialement
   - ListeOuverte=0 : la liste est vide initialement
   - voisins=NULL : la liste sera complétée plus tard lors de la lecture des liens
   - F=0 : arbitraire jusqu'a ce que celui ci soit calculé
   - G= 0 : le cout de déplacent est nul initialement.
   - pere = -1 : on a un indice absurde qui indique que pour l'instant il n'a pas de père
3. création d'un arc :void creation_arc(int numdepart, int numarrivee, double cout,T_SOMMET* graphe). Il s'agit d'un ajout en tête.
4. fonctions de suppression :void suppression_arc(L_ARC arc) et void suppression_graphe(T_SOMMET* graphe,int nbsommet);
5. Fonctions d' affichage d'un arc : void affiche_arc(L_ARC voisins) , void affiche_sommet(T_SOMMET* psommet), afficher_graphe(T_SOMMET* graphe,int nbsommet)

Enfin, deux fonctions nécessaires au fonctionnement de A* ont été ajoutées : le calcul de l'heuristique, et la recherche du cout du trajet d'un sommet à l'un de ses voisins direct : double H(int sommets, int sommeta, T_SOMMET* graphe) double cout(int depart,int arrivee,T_SOMMET* graphe)

Afin de tester la constitution des graphes, l'ouverture d'un graphe a d'abord été testée avec affichage direct dans le terminal des données lues dans le document. On teste ensuite la création du graphe et son affichage : on crée un graphe et on affiche celui-ci et on le supprime : ceci permet de vérifier la bonne allocation du graphe et des contrôles élémentaires sur l'affichage et la suppression : après suppression, on affiche bien des données aberrantes, la mémoire des objets alloués a bien été réaffectée. On peut alors tester la fonction creation_sommet en remplissant tous les champs sauf ceux destinés aux listes. Après affichage on peut vérifier la conformité des données à ce qui est indiqué dans le document. On réitère le processus en ajoutant ensuite les liens entre les sommets. Une fois ces fonctions vérifiées, on peut affirmer que la création et l'affichage du graphe fonctionnent. Il reste donc à tester le calcul du cout et l'heuristique : On crée de nouveau le graphe à partir du programme ouvrir_fichier pour et on appelle la fonction cout pour chaque couple de sommet possible(deux boucles for parcourant le tableau graphe). On peut alors vérifier les données affichées dans le terminal à l'aide du document texte. Tous les tests précédents ont été fait d'abord avec graphe1.txt puis graphe2.txt. Il faut maintenant tester l'heuristique : ce test ne peut être fait qu'à l'aide de graphe2.txt : il faut des coordonnées. On appelle la fonction après création du graphe pour plusieurs points et on calcule à la main le cout théoriques à partir de la formule fournie par l'énoncé pour l'heuristique. Le fonctionnement du graphe est maintenant validé. On modifie l'ouverture d'un graphe dans le cas où on recherche un trajet en métro afin qu'il attribue un coût nul aux correspondances(de cout 360) et un cout faible(100) aux correspondances plus longues (cout affiché de 600). Seul l'affichage du graphe est testé de nouveau. Tous les points prononcés ont été validés.

4.2 Les tas

Pour la gestion des tas on a repris la structure de tas vue en TD en ajoutant les fonctionnalités qu'on estimait nécessaires à la réalisation de ce projet. PopTas permet notamment de supprimer le sommet et de le retourner pour la phase de sélection du meilleur sommet. On avait aussi besoin de pouvoir supprimer un élément précis dans le tas en le cherchant avec chercheDansTas puis on le supprime avant de réorganiser le tas pour qu'il se retrouve en bonne place puis on ajoute l'élément en fin. Cette phase pourrait être améliorée en stockant en tout instant la position dans le tas d'un élément.

4.3 Table de hachage

Lorsque l'utilisateur nous indique le trajet souhaité il nous donne les noms des sommets de départ et d'arrivée mais A* ne manipule efficacement que les positions dans le graphe(numéro du sommet). On a recourt à une table de hachage pour accéder efficacement à un sommet à partir de son nom. On défini la table de hachage comme un tableau de listes de sommets de même résultat de calcul de hachage. Chaque maillon de la liste est de type table : structure contenant l'adresse du prochain maillon, et et l'adresse du sommet dans le graphe.

Ainsi, pour manipuler cette objet, on défini :

- la fonction de calcul de hachage : unsigned int calcul_hachage(char* c,int taille_table) évaluation d'un polynôme de degré la longueur du mot dont les coefficients sont le code ascii des lettres du mot évalué en 31(convention héritée du TD).
- l'ajout d'un sommet : ajoute_sommet(table pos,T_SOMMET* psommet); il s'agit d'un ajout en tête dans une liste d'un sommet à la position indiquée.
- création de la table :table* creation_table(T_SOMMET* graphe,int taille_table,int nbsommet); allocation dynamique de la table de hachage et remplissage de celle-ci à partir du graphe préalablement constitué.
- libération de la mémoire allouée :liberer_table_hachage(table* table_hach,int taille_table);
- affichage de la table : on affiche chaque maillon chaque liste contenue dans la table.
- recherche d'un sommet dans la table : T_SOMMET* chercher_dans_table(table* table_hach,int taille_table,char* nom); calcule l'image du mot par la fonction de hachage, puis parcourt la liste adéquat pour retrouver l'adresse du sommet.

On teste d'abord la fonction de hachage : on vérifie qu'en lui demandant plusieurs fois le même mot elle renvoie le même résultat(il s'agit bien d'une fonction), puis que les résultats diffèrent dans l'ensemble pour des mots différents en entrée (il y a un intérêt à cette structure par rapport à un simple parcourt du graphe). On vérifie ensuite la création de la table ; on crée le graphe, puis on crée la table et la comète. On affiche alors la table et on vérifie que tous les sommets ont bien été affichés. On peut ensuite vérifier que la recherche dans la table fonctionne ce qui permet de valider le fonctionnement en tant que table : on entre un nom de sommet au clavier et on vérifie à l'aide du document texte que le rang renvoyé est bien celui du sommet (le rang dans le graphe est le numéro du sommet). On estime alors que la table de hachage est fonctionnelle. Ces tests ne peuvent être fait qu'avec des fichiers restreints du fait de la nécessité de vérifier soit même dans le fichier toutes les informations renvoyées par l'algorithme.