Rapport de projet

Gaya LAHOUAZI

Louiza CHENAOUI 21213239

Sorbonne Université

Structures de données - LU2IN006

Résumé

Dans ce projet, nous nous intéressons au plan du réseau de fibres optiques d'un agglomération, et comment l'optimiser.

Dans un premier temps, nous reconstituerons une instance de listes de chaines, à partir de laquelle nous reconstituerons le plan du réseau, et ceci en utilisant plusieur structures de données différentes: Les **Listes Chainées**, les **Tables de Hachage** et les **Arbres Quaternaires** nous trouverons la definiton de ces structures dans les fichiers ***Chaine.h***, ***Hachage.h*** et ***ArbresQuat.h***.

Enfin, nous allons optimiser le réseau de manière à avoir les commodités les plus courtes possibles grace a la structure de **Graphe non orienté,** dont nous trouverons la définition dans le fichier ***Graphe.h.***

Les Executables prévus et leurs buts:

cha: Pour tester les fonctions associée aux instances listes de chaines (ChaineMain.c)

res: Pour tester les fonction associées aux instances réseau (ReconstitueReseau.c)

comp: Pour comparer les temps de calcul des différentes fontcions de reconstitutions (MainComp.c)

timer.sh: Pour avoir les temps d'execution des fonctions de reconstitutions du réseau pour toutes les instances et toutes les structures, stockées dans un fichier (MainTime.c)

Lecture, stockage et affichage des données

Dans cette partie nous avons rédigé les fonctions présentes dans le fichier Chaine.c ainsi que le main ChaineMain.c, que l'onexecutera comme suit : "./cha <nom du fichier.cha>, cette partie nous permet d'avoir l'instance de chaines a partir de laquelle on reconstituera les reseaux.

Reconstitution du réseau - Listes Chainées

Dans cette partie, nous avons commencé la reconstitutions des reseaux grace aux listes chainées, par le biais des fonctions ***RechercheCreerNoeudListe(Reseau * R, float x, float y) et ReconstitueReseauListe(Chaines* C),*** on notera un inconvénient majeur à cette méthoque, car dans le cas ou le noeud n'est pas présent il faut parcourir tout le réseau avant de le créer et de l'ajouter. Le temps de calcul pour la recherche d'un élément est donc proportionnel à la taille de la structure dans laquelle on recherche. Les fonctions écrites dans cette partie sont dans le fichier ***Reseau.c*** ,on peut les tester dans le programme res, qui s'exécute comme suit : "./res <nom fichier> 1 (le 1 est pour spécifier qu'on reconstitue le réseau en utilisant les listes chainées.)

Reconstitution du réseau - Manipulation

Dans cet exercice nous avons simplement écrit des fonctions pour écrire le réseau, l'afficher sous forme de fichier SVG et obtenir des informations telles que le nombre de liaisons et le nombre de commodités dans un réseau, ces dernières s'éxecutent de la meme manière, indépendamment de la méthode utilisée pour reconstituer le réseau, elles sont testées dans le programme res, qui s'exécute comme suit : "./res <nom fichier><option de compilation>

Reconstitution du réseau - Table de Hachage

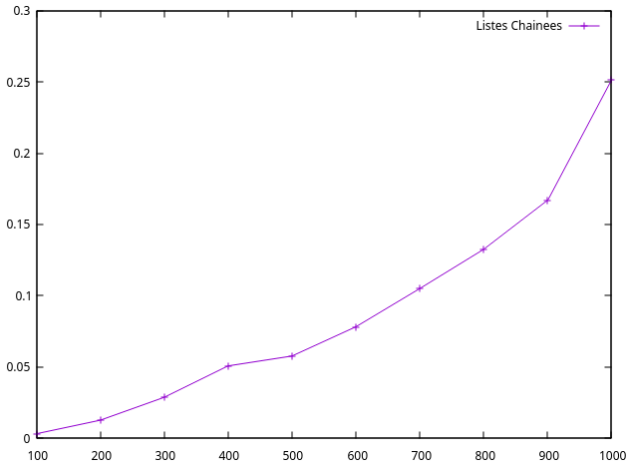
Dans cette partie, nous avons commencé la reconstitutions des reseaux grace aux listes chainées, par le biais des fonctions ***RechercheCreerNoeudHachage(Reseau * R, float x, float y) et ReconstitueReseauHachage(Chaines* C),*** on notera que le temps de recherche est bien plus court que pour les listes chainées, car grace à la fonction de hachage on peut accéder directement à la cas correspondante au noeud recherché, de plus la fonction de hachage générée (et testée dans le main ReconstitueReseau.c) semble ne causer que tres peu de collisions, enfin on notera que le choix de la taille est crucial car si la table est trop petite pour le nombre d'éléments, il y aura plus de collisions et donc plus de temps de recherche. Les fonctions écrites dans cette partie sont dans le fichier ***Hachage.c*** ,on peut les tester dans le programme res, qui s'exécute comme suit : "./res <nom fichier> 2 (le 2 est pour spécifier qu'on reconstitue le réseau en utilisant les tables de hachage.)

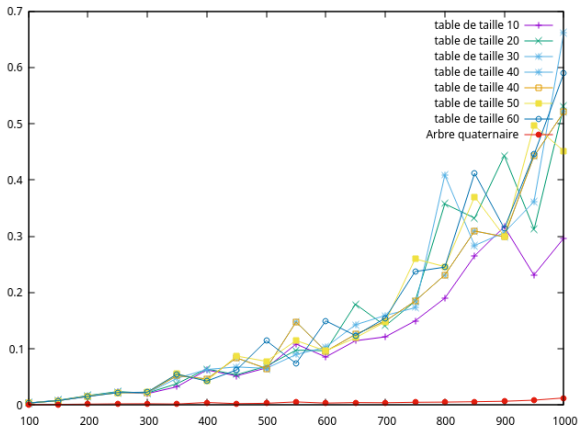
Reconstitution du réseau - Arbre Quaternaire

Dans cette partie, nous avons commencé la reconstitutions des reseaux grace aux listes chainées, par le biais des fonctions ***RechercheCreerNoeudArbre(Reseau * R, float x, float y) et ReconstitueReseauArbre(Chaines* C),*** on notera que l'arbre étant trié, on ne parcourt qu'un sous-arbre pour trouver le noeud s'il est présent dans l'arbre ou pas, cette méthode est donc largement plus efficace que les listes chainées avec l'avantage de ne pas avoir à determiner la taille idéale d'une table de hachage . Les fonctions écrites dans cette partie sont dans le fichier ***ArbreQuat.c*** ,on peut les tester dans le programme res, qui s'exécute comme suit : "./res <nom fichier> 3 (le 3 est pour spécifier qu'on reconstitue le réseau en utilisant les arbres quaternaires.)

Comparaison des structures

Afin de determiner quelle structure est la plus efficace, nous avons deux programmes que nous pouvons executer, le premier étant ***comp*** (avec le main dans CompMain.c) dans lequel on génère des chaines aléatoires de grandes tailles, à partir desquelles on reconstitue le réseau en utilisant les différentes structures de données, les temps de calcul sont sauvegardés dans les fichiers ***testChaine*** et ***testHach_Arbre***. On peut alors utiliser ces données pour tracer les graphes des temps de calcul en fonction du nombre de chaines (avec un nombre de points par chaine, un xmax et un ymax constants). On verra ci dessous les graphes obtenus.





On notera toutefois que les résultats de cette méthode ne sont pas totalement fiables, car la taille des tables de hachage est petite par rapport au nombre de points, et que nous n'avons itéré que sur un petit nombre de point, par soucis matériel. On ne peut donc pas observer de temps d'execution tres hauts pour les listes chainées,alors que pour des Reseaux plus grands le temps d'execution est bien plus grand que pour les tables de hachage.

Notre seconde méthode consiste à executer le script ***timer.sh*** qui lui itère sur les instances déja fournies, et renvoie le temps d'execution. On constate après l'avoir lancé que la reconstitution

selon les listes chainées est hautement inéfficace et prend plusieurs heures a s'executer, tandis que le temps d'execution pour les arbres quaternaires et les tables de hachage est plus raisonnable.

Optimisation du reseau

Dans cette partie, il est question d'utiliser un graphe pour optimiser le réseau et donc d'avoir des commodités plus courtes et plus efficaces.

Pour cela on utilise l'algorithme du parcours en largeur, avec lequel, a partir d'un noeud, on parcourt ses noeuds voisins, puis les voisins des voisins, ainsi de suite jusqu'a arriver au noeud qu'on recherche. Ceci aura pour effet de nous donner les commodités les plus courtes possibles.

Pour cela, on crée d'abord le graphe grace a la fonction ***CreerGraphe(Reseau * R),*** et à partir de là nous avons pu créer nos fonctions intermédiaires pour l'optimisation.

Malheureusement, par manque de temps nous n'avons pas pu obtenir de code fonctionnel sur cette partie.