Rapport de projet

Réorganisation d'un réseau de fibres optiques

Projet réalisé par

ZHANG Elodie & GRANET Sofija

Projet encadré par

Julia Sauvage & Nathanael Gross–Humbert

LU2IN006 - Structures de données

Sorbonne Université - Licence Informatique

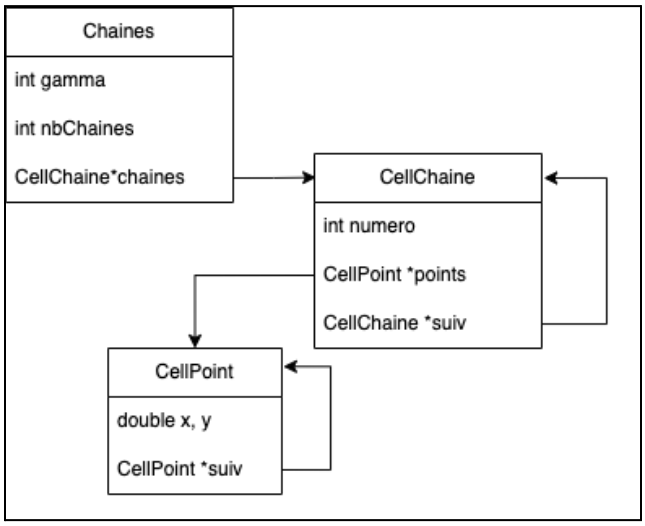
I - Résumé du projet

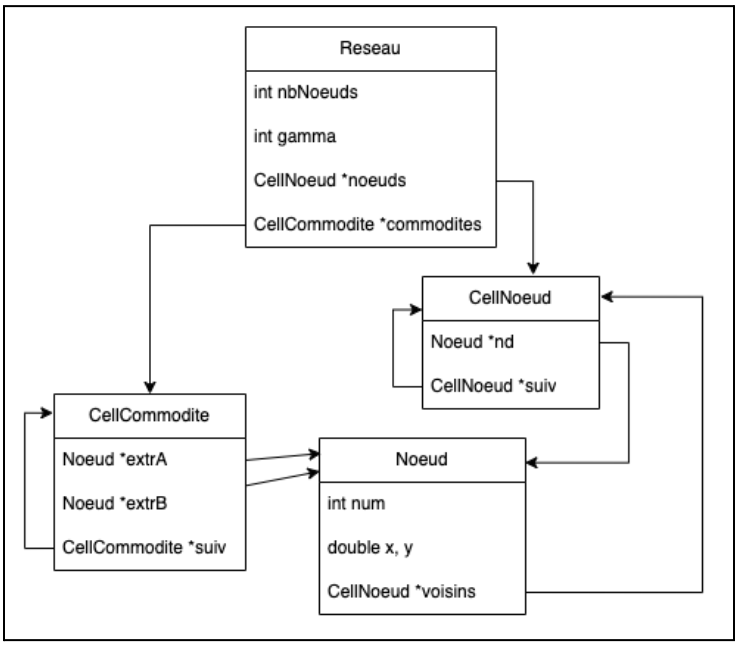
Nous nous pencherons sur un réseau de fibres optiques et sa gestion. Un réseau est un ensemble de câbles, chacun contenant un ensemble de fibres optiques et reliant des clients.

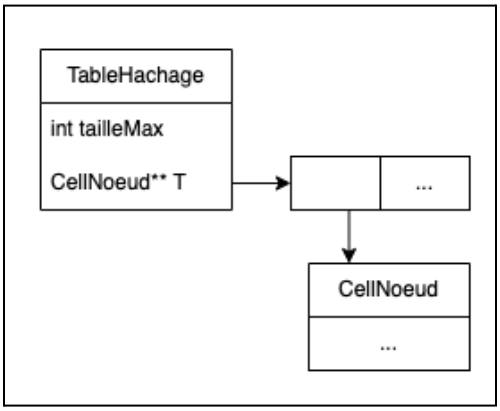
Nous verrons d'abord la reconstitution du réseau de fibres optiques dans son intégralité à partir de chaînes, à l'aide de trois structures de données : liste chaînée, table de hachage et arbre quaternaire *(cf partie II - Structures et fonctions principales)*.

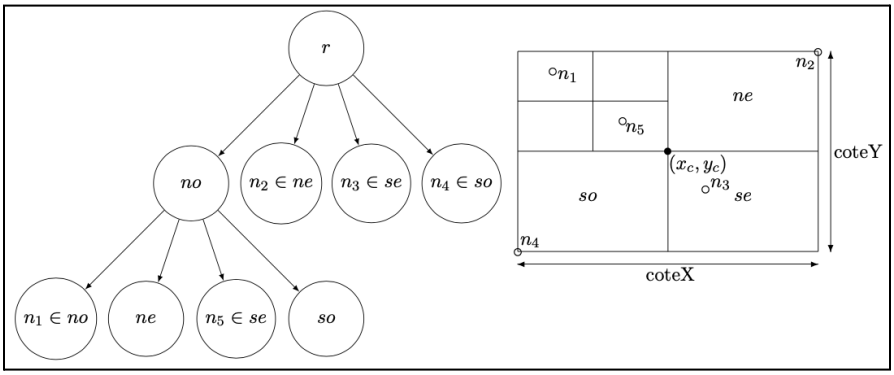
II - Structures et fonctions principales

Les structures utilisées dans ce projet sont les suivantes :

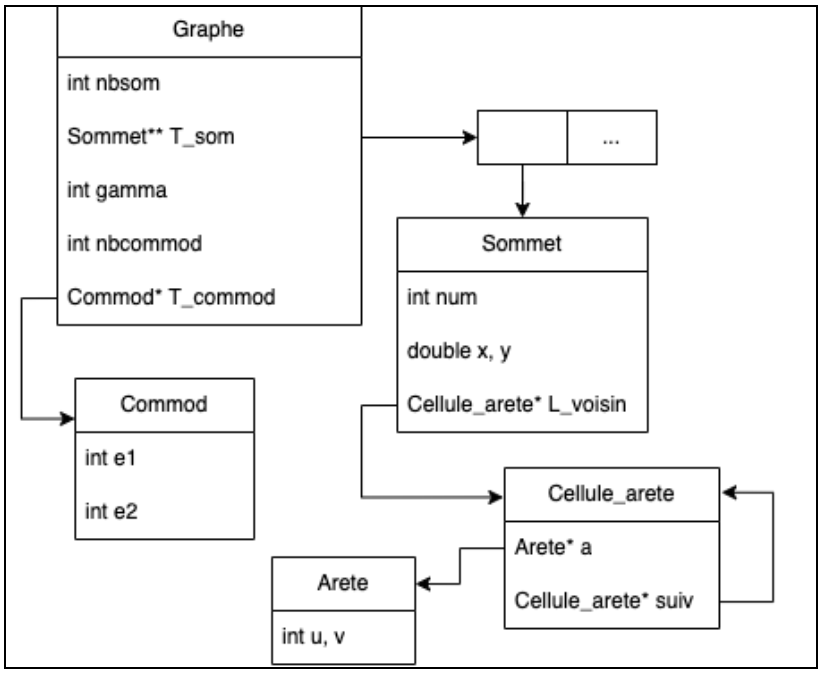








Abre quaternaire - Schéma tiré du sujet



Les fichiers *.h* contiennent les définitions des structures de données utilisées pour chaque partie du sujet et les fonctions spécifiques à celles-ci.

Les principales fonctions sont celles de reconstitution du réseau, à savoir les fonctions **`reconstitueReseauListe`** *(cf Reseau.c)*, **`reconstitueReseauHachage`** *(cf Hachage.c)* et **`reconstitueReseauArbre`** *(cf ArbreQuat.c)*.

Les fonctions de libération de mémoire sont également très importantes ; elles nous permettent de libérer les espaces alloués afin d'éviter les fuites de mémoire.

III - Réponses aux questions du sujet

On veut comparer les trois structures (liste chaînée, table de hachage et arbre quaternaire) par rapport au temps nécessaire pour effectuer leur fonction de reconstitution du réseau à partir de chaînes.

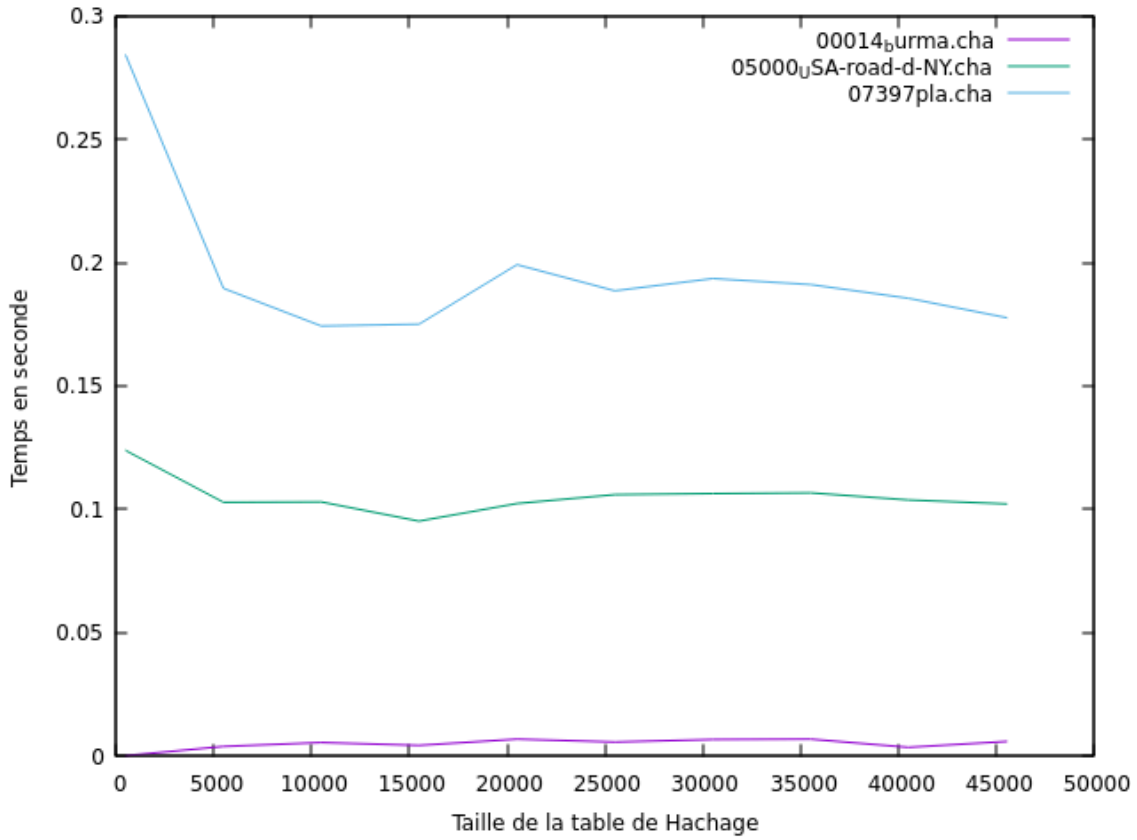
Remarque : Si les graphiques ne sont pas assez grands dans ce rapport pour être lisibles, ils sont disponibles en PDF dans le répertoire Graphiques.

6.1

On compare les temps de calcul de chaque structure à partir des instances contenues dans les fichiers *00014_burma.cha*, *05000_USA-road-d-NY.cha* et *07397_pla.cha* *(cf Graphes/temps_fichier.txt)*.

Lorsqu'on a affaire à de nombreuses chaînes composées de beaucoup de points, on s'aperçoit que la structure d'arbre quaternaire est la méthode la plus lente, derrière la liste chaînée et la table de hachage qui offrent de meilleures performances. La meilleure méthode semble être la table de hachage ; elle est bien plus rapide.

De plus, en faisant varier la taille de la table de hachage, on obtient le graphique suivant :

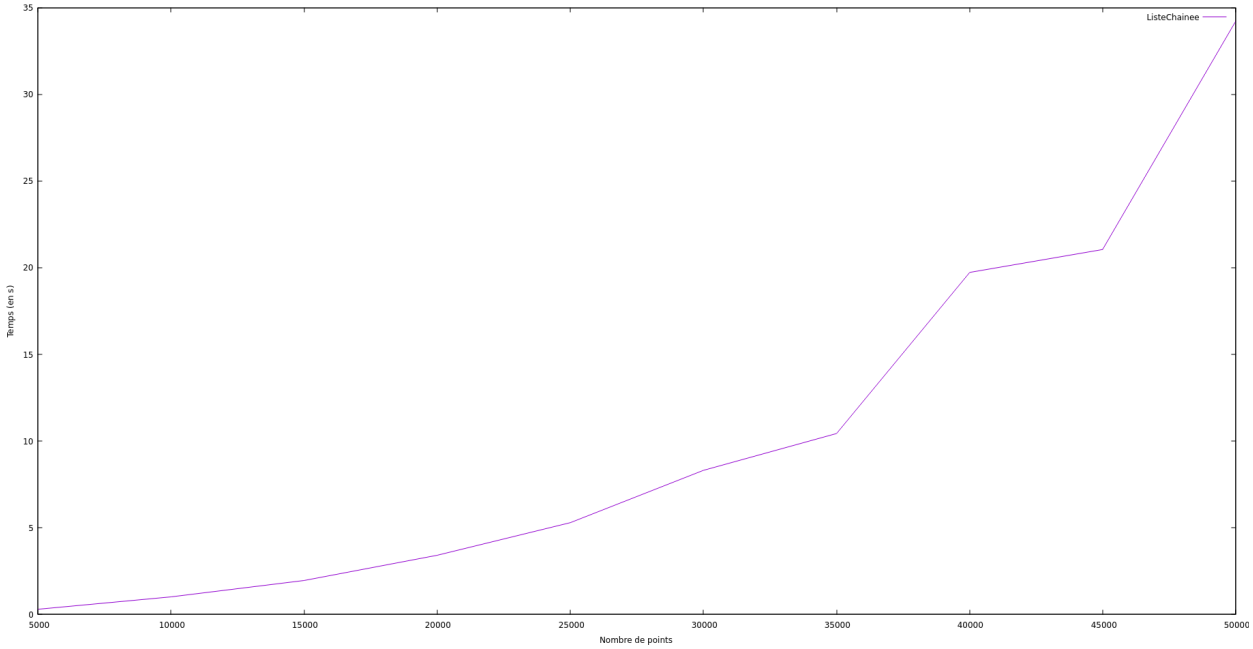


Graphe 1 - Temps de calcul de reconstitueReseauHachage en fonction de la taille de la table

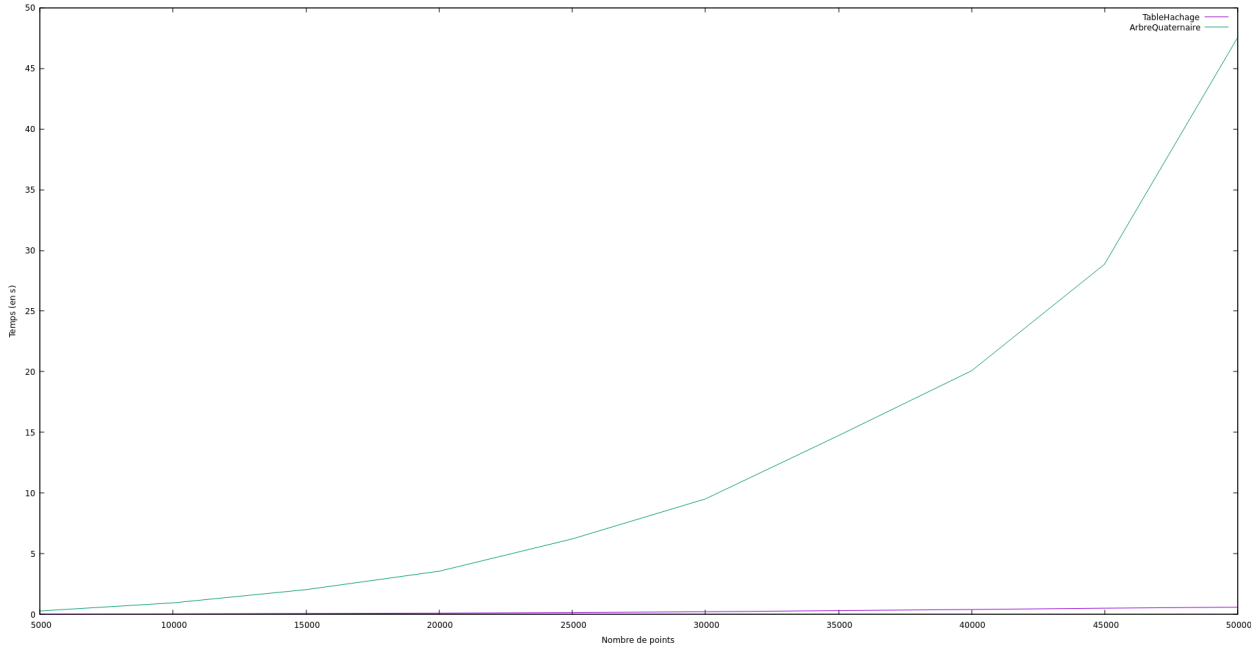
Lorsque la chaîne à traiter est très grande, il vaut mieux avoir une taille de table de hachage grande. Tandis que pour une petite chaîne comme celle provenant du fichier *00014_burma.cha,* une table de plus petite taille est plus performante.

Lorsque la taille de la table est relativement petite (500 cases), le temps de calcul est plus élevé que pour une table plus grande (45 000 cases).

6.3 On construit à présent des graphiques prenant en abscisse le nombre de points total des chaînes et en ordonnée le temps de calcul selon la structure de données utilisée.
Les chaînes sont générées aléatoirement à l'aide de la fonction **`generationAleatoire`**, avec **`nbPointsChaine = 10`**, **`xmax = 100`** et **`ymax = 100`** (*cf Calcul.c*). Nous avons choisi ces valeurs, car avec des valeurs plus grandes le temps de calcul dépasse 30 minutes.



Graphe 2 - Temps de calcul de reconstitueReseauListe en fonction du nombre de points total



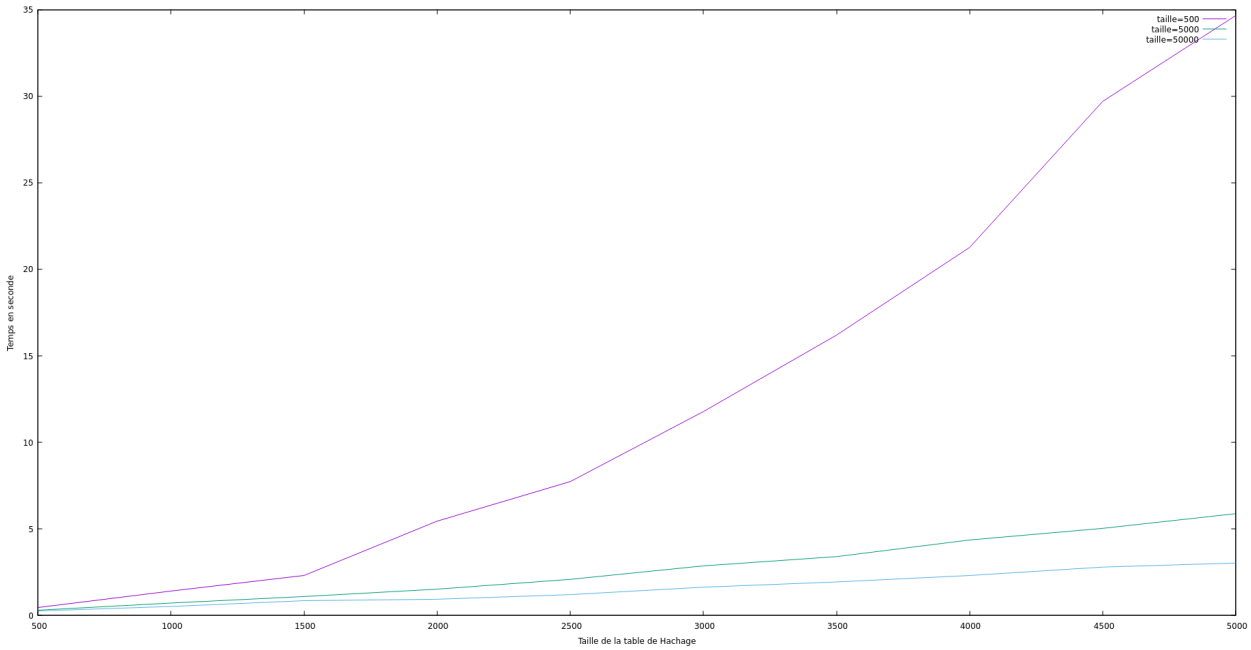
Graphe 3 - Temps de calcul de reconstitueReseauHachage et reconstitueReseauArbre en fonction du nombre de points total *(pour une table de hachage de taille fixe)*

D'après les courbes ci-dessus, nous pouvons observer que la structure de hachage est bien plus rapide que les autres. Ses temps de calculs ne dépassent pas la seconde. Nous pouvons aussi observer que les temps de calculs avec les structures de liste chaînée et d'arbre quaternaire n'augmentent pas proportionnellement mais d'une manière presque exponentielle.

À présent, on fait varier la taille de notre table de hachage et on étudie les temps de calcul de **`reconstitueReseauHachage`** en fonction de celle-ci.

On s'attend à ce que le temps de calcul diminue quand la taille table de hachage augmente, puisque les listes chaînées qui résolvent les collisions devraient être plus petites.

En pratique, pour une table de hachage de taille 500, le temps est beaucoup plus grand que pour une table plus grande ; en revanche il y a peu de différence entre une table de 5 000 et de 50 000 cases *(cf graphes ci-dessous)*.



Graphe 4 - Temps de calcul de reconstitueReseauHachage sur des chaines aléatoires, en fonction de la taille de la table

IV - Jeux d’essais

Nous avons effectué les tests des fonctions présentes dans le fichier Chaine.c en utilisant le programme ChaineMain. En effet, nous avons utilisé les fichiers 00014_burma.cha, 05000_USA-road-d-NY.cha et 07397_pla.cha pour reconstituer les chaînes puis nous les avons affichées avec la fonction **`afficheChainesSVG`**. Avant cela, la fonction **`lectureChaine`** a été validée à l'aide de la fonction **`ecrireChaine`**.

Le test de la fonction clef pour la table de hachage de la question **4.2** à été effectué à la fin du fichier **`Hachage.c`** *(cf lignes 152-162)*.

Les fonctions nécessaires à la gestion d'un réseau ont été testées dans le main de **`ReconstitueReseau.c`**.

Tout ce qui se rapporte aux graphes à été testé dans le fichier **`Graphe.c`**.

De plus, nous avons utilisé l'outil ***Valgrind*** pour détecter et corriger au maximum les fuites de mémoire sur tous les exécutables du projet.

V - Analyse des performances calculées

D’après les graphes, que les chaînes soient générées aléatoirement ou non, la fonction **`reconstitueReseauHachage`** possède le temps de calculs le plus bas parmi les trois manière de reconstituer un réseau.

Sources extérieures

Nous nous sommes servies de l’application **draw.io** afin de réaliser les diagrammes explicatifs des structures de données utilisées.