FRANCOISE Raphaël
ZENG Fanxiang

# Compte rendu
# -
# Projet : Réorganisation d'un réseau de fibres optiques

## Préambule

**But de ce projet :**

Ce projet consiste en la réorganisation d'un réseau de fibres optiques d'une agglomération.

Dans une première partie, nous allons reconstituer le plan du réseau. Pour cela, nous utiliserons le fait que chaque opérateur connaît le tronçon de fibres optiques qu'il utilise dans son réseau. Nous faisons l'hypothèse qu'il y a au moins une fibre utilisée par câble. Nous allons répondre à cette problématique via l'utilisation de différentes structures telles que les listes chaînées, les tables de hachage et les arbres.

Dans une seconde partie, nous allons réorganiser ce réseau afin de l'optimiser au maximum et d'éviter la sur-exploitation ou la sous-exploitation de certains câbles. Pour cela, nous allons attribuer aux opérateurs des chaînes moins longues et mieux réparties dans le réseau.

En vue de réaliser cet objectif, plusieurs algorithmes vont être mis en place.

**Modélisation :**

Comment est modélisé un réseau ?

Un réseau est constitué de câbles. Chaque câble contient $n$ fibres optiques ($n>0$). Les câbles relient deux points du plan, dont on peut distinguer deux catégories :

- Les clients : client particulier, entreprise cliente ou encore un local technique de l'opérateur.
- Les concentrateurs : points du réseau où se rejoignent plusieurs câbles.

Un point peut être un client, un concentrateur, ou bien les deux à la fois.
Une chaîne correspond à la concaténation de plusieurs tronçons de fibres optiques. Toute chaîne relie toujours deux points du plan. Ce couple de points est appelé une commodité.
Il existe plusieurs opérateurs et chaque opérateur possède plusieurs chaînes de fibres optiques.

**Structures manipulées et description globale de notre code :**

- Un fichier *Chaine.h* contenant les structures suivantes :
    - Structure *CellPoint* : liste chaînée de points, composée des coordonnées du point *(double)* et d'un pointeur sur le point suivant dans la liste *(struct cellPoint)*.
    - Structure *CellChaine* : cellule d'une liste chaînée de chaîne. Elle est composée du numéro de la chaîne *(int)*, d'un pointeur sur la liste des points de la chaîne *(CellPoint)* et d'un pointeur sur la cellule suivante dans la liste *(struct cellChaine)*.
    - Structure *Chaines* : correspond à l'ensemble des chaînes. Elle est composée du nombre maximal de fibres par câble *(int)*, du nombre de chaînes *(int)* et de la liste chaînée des chaînes *(CellChaine)*.
- Un fichier *Chaine.c* composé des fonctions de lecture et d'écriture de fichier, d'affichage graphique de réseau, de calcul de la longueur totale des chaînes et de calcul du nombre de points. Tout cela est généré à partir du fichier *0014_burma.cha (cf ci-dessous)*.
- Un fichier *0014_burma.cha* modélisant un réseau et donnant le nombre de chaînes et le nombre maximal de fibres optiques par câble. Il contient aussi les différentes chaînes du réseau. Pour chaque chaîne, il est mentionné son numéro, son nombre de points et la liste de ses points. Chaque point est donné par ses coordonnées. (idem pour *05000_USA-road-d-NY.cha* et *07397_pla.cha*).
- Un fichier *ChaineMain.c* permettant de tester les fonctions présentes dans le fichier *Chaine.c*.
- Deux fichiers, *SVGwriter.c* et *SVGwriter.h*, librairie C mise à disposition pour avoir une représentation graphique des instances en HTML.
- Un fichier affichage.html généré à partir de la fonction `afficheChainesSVG(Chaines *C, char* nomInstance)`, qui permet d'avoir une représentation graphique des instances du fichier *0014_burma.cha.*
- Un fichier *Reseau.h* contenant les structures suivantes :
    - *CellNoeud* : une liste chaînée de nœuds, composée d'un pointeur vers le nœud stocké (*Nœud*) et d'un pointeur sur la cellule suivante de la liste (*CellNoeud*).
    - *Nœud* **:** Un nœud du réseau, composé d'un numéro de nœud (*int*), des coordonnées du nœud x et y (*double*) et d'un pointeur sur le premier élément de la liste des voisins du nœud (*CellNoeud*).
    - *CellCommodite :* une liste chaînée de commodités, composée de deux pointeurs sur les nœuds à l'extrémité de la commodité (*Nœud*) et d'un pointeur sur la cellule suivante de la liste (*CellCommodite*).
    - *Reseau* : un réseau, composé du nombre de nœuds du réseau et du nombre maximal de fibres par câble (*int*), d'un pointeur sur le premier élément de la liste des nœuds du réseau (*CellNoeud*) et d'un autre pointeur sur le premier élément de la liste des commodités à relier (*CellComodités*).
- Un fichier *Reseau.c* contenant toutes nos fonctions en rapport avec le réseau, que ce soit pour la liste chaînée, la table de hachage et l'arbre.

- Un fichier *ReconstitueReseau.c* nous permettant de tester nos fonctions pour les différents algorithmes mis en place.
- Un fichier *00014_bruma.res* qui est le résultat de fonction *ecrireReseau* (idem pour *05000_USA-road-d-NY.res* et *07397_pla.res*).
- Un fichier *affichageReseauListe.html* qui est un affichage du réseau reconstitué avec une liste chaînée.
- Un fichier *affichageReseauHachage.html* qui est un affichage du réseau reconstitué avec une table de hachage.
- Un fichier *affichageReseauArbre.html* qui est un affichage du réseau reconstitué avec un arbre.
- Un fichier *Hachage.h* comprenant notre structure *TableHachage*.
- Un fichier *ArbreQuat.h* composé de la structure *ArbreQuat.* Cette dernière modélise un arbre quaternaire composé des coordonnées du centre de la cellule (*int*), de la longueur et de la hauteur de la cellule (*int*), d'un pointeur vers le nœud du réseau (*Nœud*) et de quatre pointeurs vers les différents sous arbres (*ArbreQuat*)
- Un fichier *main_chiffre_compa.*c qui contient le code permettant d'exécuter automatiquement les trois fonctions de reconstruction et qui calcule uniquement leur temps de calcul.
- Un fichier *comparaison.c* qui contient la fonction *generationAleatoire* de l'exercice 6
- Un fichier *main_graph_compa.c* qui contient le code qui construit les graphiques prenant en abscisse le nombre de points total des chaînes et en ordonnée le temps de calcul selon la structure de données utilisée.
- Un dossier *donne_exo6* qui contient les résultats de l'exercice 6, à savoir :
    - Trois fichiers *07397_pla.txt, 05000_USA-road-d-NY.txt* et *00014_burma.txt* dans lesquels apparaissent les temps de calcul pour les différentes fonctions de reconstruction.
    - Six fichiers (.png) correspondant aux graphiques obtenus avec la question 6.3
- Un fichier *Graphe.h* composé des structures suivantes :
    - *Arete* : composée des numéros des sommets aux extrémités (*int*).
    - *Cellule_arete* : une liste chainée d'arêtes, composée d'un pointeur sur l'arête (*Arete)* et d'un pointeur vers l'arête suivante (*Cellule_arete*).
    - *Sommet* : composée du numéro du sommet (*int*) de ses coordonnées x et y (*double*) et d'une liste chainée des voisins (*Cellule_arete*).
    - *Commod* : composée des deux extrémités de la commodité (*int*)
    - *Graphe* : composée du nombre de sommet (*int*), d'un pointeur vers un tableau de pointeurs sur sommet (*Sommet*), du nombre de commodités (*int*) et d'un tableau des commodités (*Commod*).
- Un fichier *Graphe.c* qui contient le code permettant d'optimiser le réseau via des graphes et permettant de créer un graphe à partir d'un réseau.
- Les fichiers *Struct_File.c, Struct_File.h, Struct_Liste.c, Struct_Liste.h* mis à disposition pour effectuer les opérations sur les graphes.
- Un fichier *Makefile* afin de faciliter la compilation.

**NB : toutes les informations concernant le Makefile se situent à la dernière page.**

# Lecture, stockage et affichage de données

Exercice 1 :

Construction d'une bibliothèque de manipulations d'instances : lecture et écriture de fichier, affichage graphique de réseaux, calcul de la longueur totale des chaînes, et calcul du nombre de points.

Voir le code dans le fichier **Chaine.c.**

1.3) Le fichier d'affichage est *affichage.html.*

# Reconstitution du réseau

But de la partie : reconstituer efficacement le réseau à partir de chaînes.

## Liste chainée :

Exercice 2 :

Algorithme de reconstitution du réseau en codant l'ensemble des nœuds du réseau par une liste chaînée.

Voir le code dans le fichier **Reseau.c.**

Exercice 3 :

Méthodes pour manipuler et afficher un struct *Reseau.*

Voir le code dans le fichier **Reseau.c.**

3.2) Les fichiers .res sont disponibles dans le dossier principal.

3.3) Le fichier d'affichage est *affichageReseauListe.html.* Nous obtenons vraisemblablement la même chose que l'affichage des chaînes, on peut donc valider (en partie) nos fonctions.

## Table de hachage :

Exercice 4 :

Reconstitution du réseau par le biais d'une table de hachage avec gestion des collisions par chaînage.

4.1) Notre structure *TableHachage* est composée du nombre d'éléments (*int*), de la taille de la table (*int*) et de la table de hachage (double pointeur : *CellNoeud*). Voir code dans le fichier **Hachage.h.**

4.2) Voir code dans le fichier **Reseau.c.**
La fonction clef semble appropriée. En effet, la fonction prend en paramètres les coordonnées d'un point du réseau. Chaque point étant différent, et ayant donc des coordonnées différentes, il y a donc peu de chance d'avoir plusieurs fois la même clef. Il n'y aura donc pas beaucoup de collisions.

4.3, 4.4, 4.5) Voir code dans le fichier **Reseau.c.**
Le fichier d'affichage est *affichageReseauHachage.html*

**Arbre :**

Exercice 5 :

Utilisation d'un arbre quaternaire pour reconstituer le réseau.

Voir code dans le fichier **Reseau.c.**
Le fichier d'affichage est *affichageReseauArbre.html*

**Comparaison des trois structures :**

6.1) Voir le code dans **main_chiffre_compa.c.**
Si on regarde les fichiers *00014_burma.txt, 05000_USA-road-d-NY.txt* et *07397_pla.txt* (dans le dossier *donne_exo6)*, on peut faire plusieurs observations.
Pour un petit réseau, la table de hachage sera à privilégier si elle a une petite taille. Cependant, lorsque la taille de la table augmente cela n'est pas rentable. L'utilisation d'un arbre et d'une liste chainée est à peu près équivalente et légèrement plus lente qu'une table de hachage avec une petite taille.
Pour un réseau de grande taille, l'utilisation d'une liste chaînée n'est pas envisageable car cela prend beaucoup plus de temps que pour un arbre ou bien une table de hachage. Arriver à un certain stade, peu importe la taille de la table, le temps de calcul reste le même.
Le plus rapide étant l'utilisation d'un arbre.
Dans tous les cas il est toujours mieux d'utiliser une reconstitution par arbre quaternaire, car quand le réseau est petit ce n'est pas plus lent et lorsque le réseau est grand cela permet d'aller 10 à 100 fois plus vite.

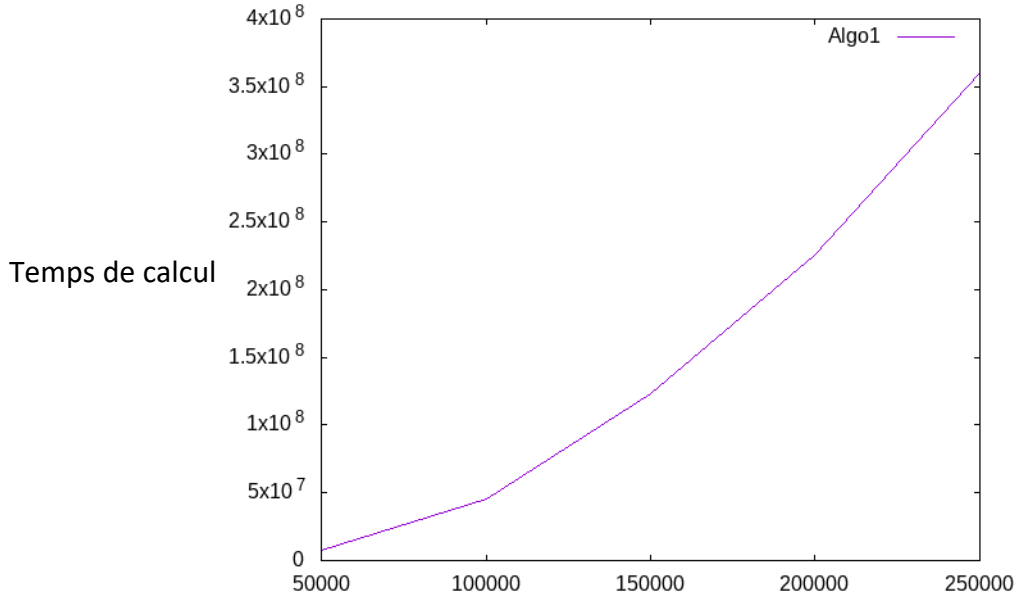
6.2) Voir le code dans le fichier **comparaison.c.**

6.3) Voir le code dans le fichier **main_graph_compa.c.**

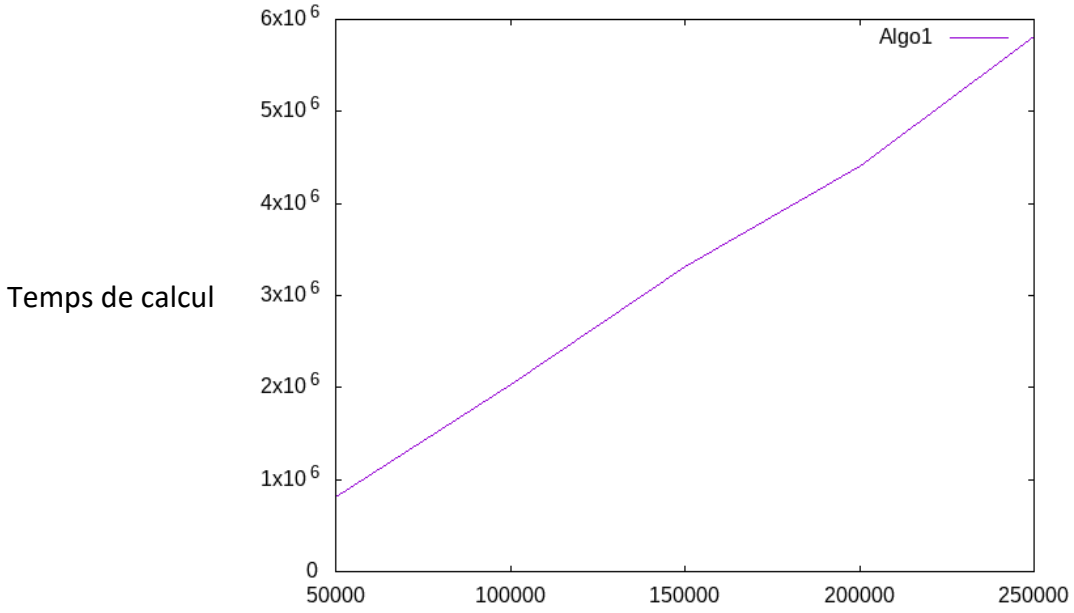
Graphiques obtenus :

NB : Nous avons réduit le nombre de chaines maximum à 2500 au lieu de 5000 (pour la table de hachage et la liste chaînée), car sinon cela prenait trop de temps. Cependant, nous avons considéré les résultats comme tout de même significatif.

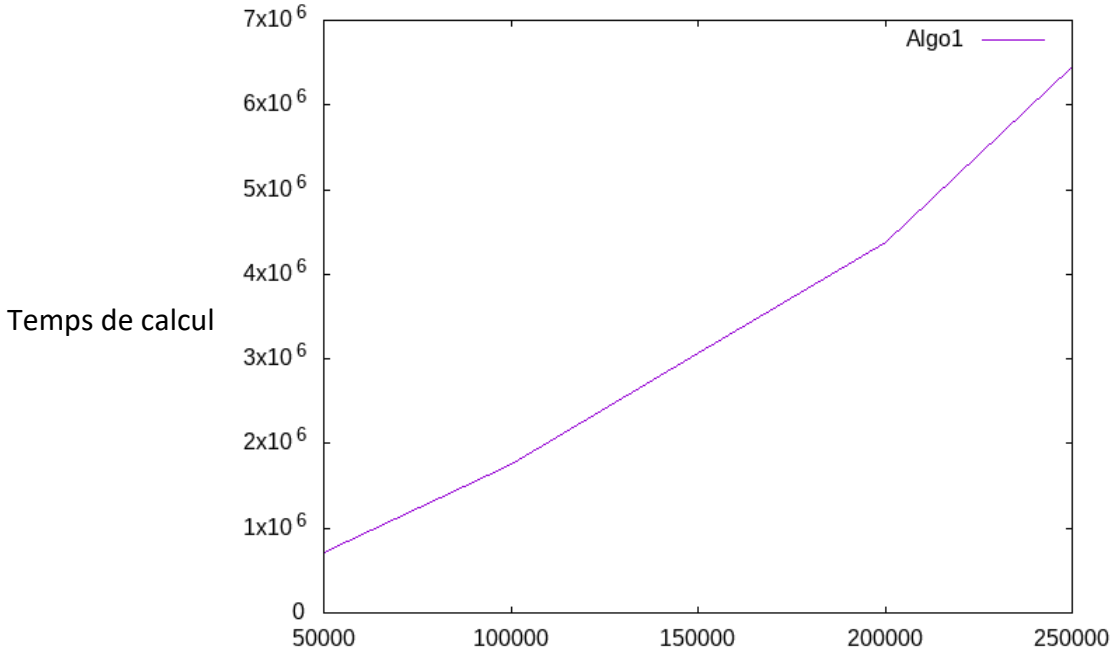
**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec la liste chaînée.**



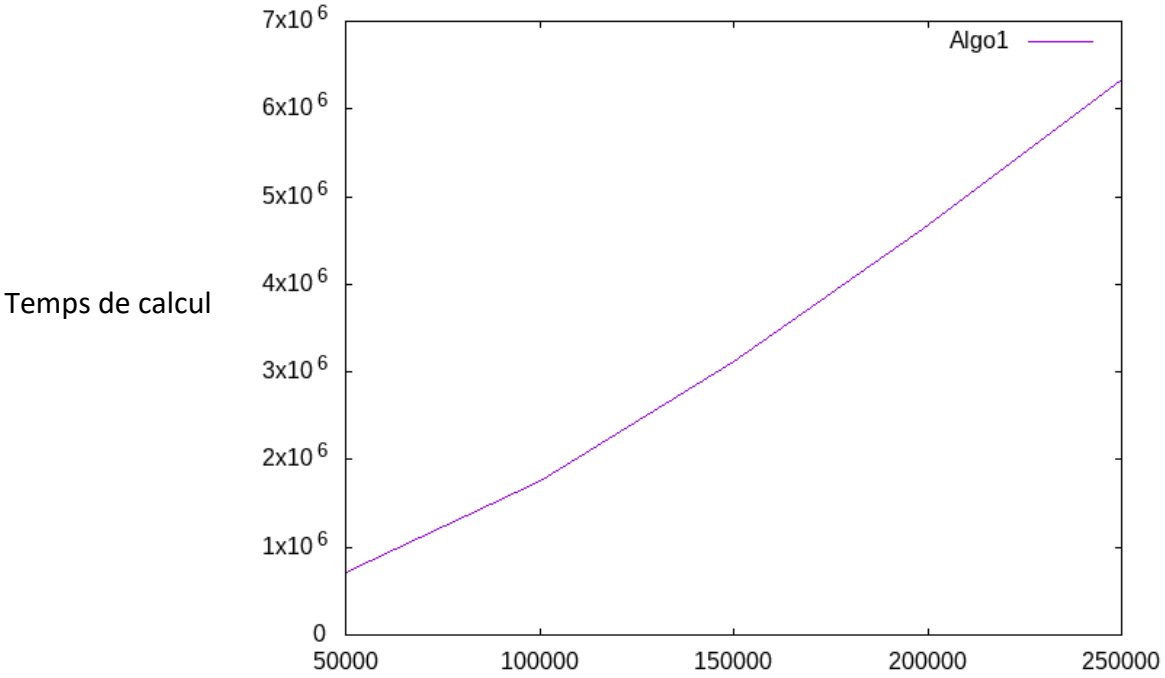

**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec la table de hachage de taille 5.**



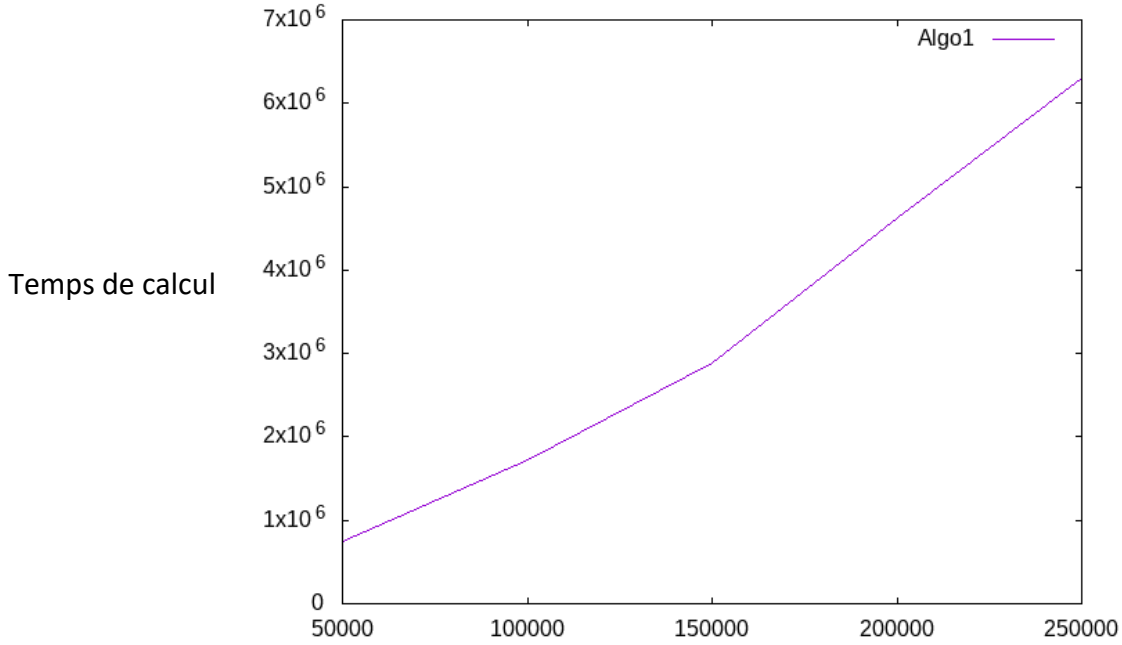
**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec la table de hachage de taille 10.**



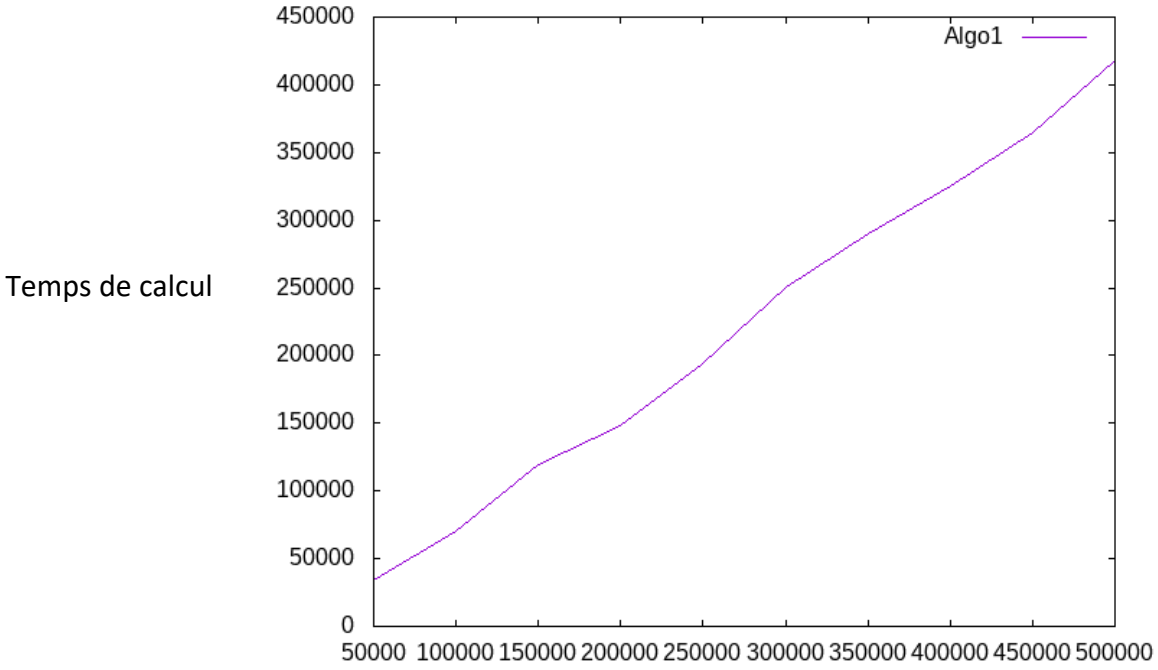

**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec la table de hachage de taille 100.**



**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec la table de hachage de taille 1000.**




**Titre : Graphique donnant les temps de calcul en fonction du nombre de points total des chaînes avec l'arbre quaternaire.**



6.4) Analyse des résultats

Nous observons grâce à ces graphiques que le temps mis par l'arbre quaternaire est beaucoup plus rapide que pour les autres structures. À nombre de points égal, l'arbre est environ 30 fois plus rapide que la table de hachage et 1000 fois plus rapide qu'une liste chaînée.
La vitesse avec la table de hachage ne varie pas beaucoup en fonction de la taille de la table mais reste moins rapide que la vitesse avec un arbre.
La vitesse avec la liste chaînée est vraiment plus lente qu'avec les autres structures.

Il est donc plus intelligent d'utiliser l'arbre quaternaire.

# Optimisation du réseau

## Graphe :

7.1, 7.2, 7.3, 7.4) Voir le code dans le fichier **Graphe.c**

7.5) La fonction permet de dire si un réseau est optimisé ou non. En effet, si pour chaque arête du graphe le nombre de chaînes qui passe par l'arête est inférieur à gamma, alors le graphe est optimisé.
Les arêtes non initialisées sont aussi testées.
*00014_burma.cha* et *05000_USA-road-d-NY.cha* ne sont pas optimisés.
Cependant, *07397_pla.cha*, lui est optimisé.

Pour améliorer la fonction nous pouvons remplacer la matrice 2D par un tableau de liste chaînée d'arêtes par lesquelles passe une chaîne. Nous considérons une liste chaînée par sommet. Cela nous permet donc de n'avoir que les arêtes qui existent.
De plus la fonction ne renvoie pas les chaînes les plus courtes entre chaque commodité. On pourrait donc modifier la signature en passant un double pointeur pour stocker les chaînes les plus courtes.

**Informations concernant le Makefile :**

- *ReconstitueReseau* a besoin des fichiers suivants : reseau.o, chaine.o, svgwriter.o et reconstituereseau.o
- *main_chiffre_compa* a besoin des fichiers suivants : *comparaison.o Chaine.o Reseau.o main_chiffre_compa.o SVGwriter.o*
- *main_graph_compa* a besoin des fichiers suivants *: comparaison.o Chaine.o Reseau.o SVGwriter.o main_graph_compa.o*
- *ReorganiseReseau* a besoin des fichiers suivants *: ReorganiseReseau.o Reseau.o SVGwriter.o Graphe.o Struct_File.o Struct_Liste.o*