Alaboud Alaa -> 21206964
Mohammad Ichraq -> 21206967

Rapport sur le Projet de Réorganisation d'un Réseau de Fibres Optiques

Ce rapport présente en détail le travail réalisé dans le cadre d'un projet qui vise à implémenter des algorithmes pour la reconstitution et la réorganisation d'un réseau de fibres optiques. Le projet est se base autour de trois parties principales, chacune abordant des aspects spécifiques de la gestion du réseau.

Partie 1: Reconstitution du Réseau

Cette partie du projet consiste à reconstituer le plan du réseau de l'agglomération, en partant du constat qu'aucun plan complet du réseau n'existe actuellement. L'algorithme de reconstitution du réseau se base sur l'hypothèse qu'il y a au moins une fibre optique utilisée par câble, permettant ainsi de reconstituer le réseau dans son intégralité.

La première partie du projet consiste à reconstituer le réseau à partir d'une liste de chaînes de points en utilisant trois méthodes différentes : la liste chaînée, la table de hachage et l'arbre quaternaire.

Partie 1 : Liste chaînés

Description :

Nous avons commencé par implémenter une structure de données pour représenter les chaînes de points du réseau. Chaque chaîne est stockée sous forme d'une liste chaînée de points. Ensuite, nous avons écrit des fonctions pour parcourir cette liste et reconstruire le réseau en établissant des liaisons entre les nœuds. Cette méthode, bien que simple à implémenter, présente des limitations en termes de performances, notamment pour un grand nombre de chaînes dans le réseau.

Exercice 1 – Manipulation d'une instance de "Liste de Chaînes" (TME5)

Description des algorithmes :

1. **lectureChaines(FILE f)* : Lit un fichier contenant une instance de "Liste de Chaînes", alloue et remplit une structure Chaines avec les données lues.
2. **ecrireChaines(Chaines *C, FILE f)* : Écrit le contenu d'une structure Chaines dans un fichier, en respectant le format spécifié.
3. **afficheChainesSVG(Chaines C, char * nomInstance)* : Génère un fichier SVG pour afficher graphiquement les chaînes dans un navigateur Web.
4. **longueurChaine(CellChaine c)* : Calcule la longueur physique d'une chaîne en additionnant les distances entre ses points.
5. **longueurTotale(Chaines C)* : Calcule la longueur totale des chaînes en sommant les longueurs de toutes les chaînes.
6. **comptePointsTotal(Chaines C)* : Compte le nombre total d'occurrences de points dans toutes les chaînes.

Jeux d'essais utilisés :

1. **Jeux d'essais pour la lecture et l'écriture de fichiers :**
    - Utiliser des fichiers de données au format .cha avec différentes configurations de chaînes et de points pour tester la fonction de lecture et d'écriture.
2. **Jeux d'essais pour l'affichage graphique :**
    - Créer des instances de chaînes avec des coordonnées spécifiques pour vérifier que l'affichage graphique généré est correct.
3. **Jeux d'essais pour le calcul de longueur et de nombre de points :**
    - Générer des chaînes avec des points disposés de manière à faciliter le calcul de la longueur totale et du nombre de points.

Analyse des performances :

Les performances du programme sont généralement satisfaisantes.

Exercice 2 – Première méthode : stockage par liste chaînée

Description :

Dans cet exercice, l'objectif est d'implémenter l'algorithme de reconstitution de réseau en utilisant une liste chaînée pour stocker l'ensemble des nœuds du réseau. Chaque nœud du réseau est identifié par ses coordonnées, et on connaît la liste de ses nœuds voisins avec lesquels il est relié par un câble. Le réseau comprend également des câbles et des commodités, représentées par des paires de nœuds à relier par une chaîne.

Description des algorithmes :

1. **rechercheCreeNoeudListe(Reseau R, double x, double y)* : Cette fonction recherche un nœud correspondant aux coordonnées (x, y) dans la liste chaînée des nœuds du réseau. Si le nœud existe, il est renvoyé ; sinon, un nouveau nœud est créé avec le numéro nbNoeuds+1 et ajouté à la liste des nœuds du réseau.
2. **reconstitueReseauListe(Chaines C)* : Cette fonction reconstitue le réseau à partir de la liste des chaînes C. Elle utilise la fonction rechercheCreeNoeudListe pour créer et relier les nœuds du réseau en suivant l'algorithme spécifié.
3. **main ReconstitueReseau.c** : Ce programme principal utilise la ligne de commande pour prendre un fichier .cha en paramètre et un nombre entier indiquant la méthode à utiliser pour la reconstitution du réseau (liste, table de hachage, ou arbre).

Jeux d'essais utilisés :

1. **Jeux d'essais pour la création de nœuds et de liaisons :**

- Utiliser différentes configurations de chaînes pour tester la création des nœuds et des liaisons.

2. **Jeux d'essais pour la reconstitution du réseau :**
    - Créer des instances de chaînes variées et vérifier que le réseau reconstruit correspond bien à la structure initiale.

Analyse des performances :

Les performances de ce programme dépendent de la taille du réseau et du nombre de chaînes à reconstituer. Des optimisations pourraient être envisagées si nécessaire pour améliorer les performances dans des cas d'utilisation intensifs.

Exercice 3 – Manipulation d'un réseau

Jeux d'essais utilisés :

1. **Jeux d'essais pour l'écriture du réseau dans un fichier :**
    - Créer des réseaux avec différentes configurations et vérifier que le fichier généré est correct.
2. **Jeux d'essais pour l'affichage graphique du réseau :**
    - Utiliser des réseaux de différentes tailles et structures pour vérifier que l'affichage SVG représente correctement le réseau.

Analyse des performances :

Les performances de ce programme dépendent de la taille du réseau et du nombre de chaînes à reconstituer.

Partie 2 : Méthode de la Table de Hachage

Description :

Pour améliorer les performances de recherche et d'insertion par rapport à la méthode de la liste chaînée, nous avons opté pour l'utilisation d'une table de hachage. Nous avons créé une structure de table de hachage qui associe chaque point à un index calculé à partir de ses coordonnées. Cette méthode permet une recherche et une insertion plus rapides, en particulier lorsque le nombre de chaînes est important. Cependant, elle peut également présenter des limitations en cas de collisions fréquentes.

Exercice 4 – Deuxième méthode : stockage par table de hachage

Dans cet exercice, nous allons utiliser une table de hachage avec gestion des collisions par chaînage pour stocker les nœuds d'un réseau. La table de hachage va nous permettre de rapidement déterminer si un nœud a déjà été stocké dans le réseau..

Description des algorithmes :

1. **Structure TableHachage** : Cette structure comprend un tableau de pointeurs vers une liste de nœuds, ainsi que des fonctions pour initialiser la table de hachage, ajouter un nœud, rechercher un nœud et libérer la mémoire.
2. **Fonction de hachage** : Nous utiliserons une fonction de hachage spécifique pour générer les clés à partir des coordonnées (x, y) d'un point.
3. **Recherche ou création d'un nœud** : Nous implémenterons une fonction pour rechercher ou créer un nœud dans la table de hachage. Si le point n'existe pas, nous le créerons et l'ajouterons à la table de hachage et à la liste des nœuds du réseau.
4. **Reconstitution du réseau avec table de hachage** : Nous implémenterons une fonction pour reconstruire le réseau à partir d'une liste de chaînes en utilisant une table de hachage de taille spécifiée.

Q 4.2 :

Tester cette fonction avec différentes valeurs de x et y pour vérifier si elle produit des clés appropriées.

Voilà les valeurs obtenues :

4 8 13 19 26 34 43 53 64 76 7 12 18 25 33 42 52 63 75 88 11 17 24 32 41 51 62 74 87 101 16 23 31 40 50 61 73 86 100 115 22 30 39 49 60 72 85 99 114 130 29 38 48 59 71 84 98 113 129 146 37 47 58 70 83 97 112 128 145 163 46 57 69 82 96 111 127 144 162 181 56 68 81 95 110 126 143 161 180 200 67 80 94 109 125 142 160 179 199 220

- Oui la fonction semble appropriées et les valeurs bien distancées.

Jeux d'essais utilisés :

1. **Jeux d'essais pour la création de la table de hachage :**
    - Tester différents jeux de données avec des points de différentes configurations pour évaluer l'efficacité de la fonction de hachage.
2. **Jeux d'essais pour la recherche et la création de nœuds :**
    - Utiliser des points déjà présents dans la table de hachage ainsi que des points nouveaux pour tester la fonction de recherche et de création de nœuds.
3. **Jeux d'essais pour la reconstruction du réseau :**
    - Utiliser différentes tailles de tables de hachage et différentes configurations de chaînes pour évaluer l'efficacité de la reconstruction du réseau.

Analyse des performances :

Les performances de ce programme dépendent de la taille du réseau et du nombre de chaînes à reconstituer.

Partie 3 : Méthode de l'Arbre Quaternaire

Description :

Pour surmonter les limitations des méthodes précédentes, nous avons exploré l'utilisation d'un arbre quaternaire. Nous avons conçu une structure d'arbre quaternaire capable de diviser efficacement l'espace en quatre parties et d'organiser les nœuds du réseau en fonction de leurs coordonnées. Cette méthode semble assez efficace en termes de temps de calcul mais plus complexe à mettre en place.

Exercice 5 – Troisième méthode : stockage par arbre quaternaire

Description du sujet :

Dans cet exercice, nous allons utiliser un arbre quaternaire pour reconstituer le réseau. L'arbre quaternaire permettra une recherche rapide des nœuds déjà présents dans le réseau. Chaque nœud de notre réseau sera stocké au niveau des feuilles de l'arbre quaternaire, où chaque feuille représente une cellule rectangulaire dans un espace à deux dimensions.

Description des algorithmes :

1. **Fonction chaineCoordMinMax** : Cette fonction détermine les coordonnées minimales et maximales des points constituant les différentes chaînes du réseau en parcourant la liste des chaînes.
2. **Fonction creerArbreQuat** : Cette fonction crée une cellule de l'arbre quaternaire avec les coordonnées du centre, la longueur et la hauteur spécifiées.
3. **Fonction insererNoeudArbre** : Cette fonction insère un nœud du réseau dans l'arbre quaternaire. Elle gère trois cas : arbre vide, feuille et cellule interne, en divisant récursivement la cellule en quatre si nécessaire.
4. **Fonction rechercheCreeNoeudArbre** : Cette fonction recherche ou crée un nœud dans l'arbre quaternaire à partir des coordonnées spécifiées. Elle gère les mêmes cas que la fonction d'insertion.
5. **Fonction reconstitueReseauArbre** : Cette fonction reconstruit le réseau à partir de la liste des chaînes en utilisant l'arbre quaternaire. Elle parcourt les chaînes, recherche ou crée les nœuds à l'aide de la fonction précédente, puis ajoute les liaisons et les commodités au réseau.

Jeux d'essais utilisés :

1. **Jeux d'essais pour la création de l'arbre quaternaire :**
    - Création de différents ensembles de points avec des configurations variées.
    - Test de la fonction de création de l'arbre quaternaire avec ces ensembles de points pour évaluer son efficacité.
2. **Jeux d'essais pour l'insertion et la recherche de nœuds :**
    - Utilisation de points déjà présents dans l'arbre quaternaire.
    - Ajout de nouveaux points pour tester la fonction d'insertion et de recherche de nœuds.
3. **Jeux d'essais pour la reconstruction du réseau :**
    - Utilisation de différentes tailles de tables de hachage.

- Utilisation de différentes configurations de chaînes de points.
- Évaluation de la performance de la reconstruction du réseau en fonction de ces variations.

Analyse des performances :

Les performances de cette méthode dépendent de la structure et de l'efficacité de l'arbre quaternaire, ainsi que de la qualité de la fonction de recherche et d'insertion.

Exercice 6 – Comparaison des trois structures (TME9)

Reformulation courte du sujet :
Dans cet exercice, nous comparons les temps de calcul de trois structures de données pour tester l'existence d'un nœud dans un réseau : la liste chaînée, la table de hachage et l'arbre quaternaire.

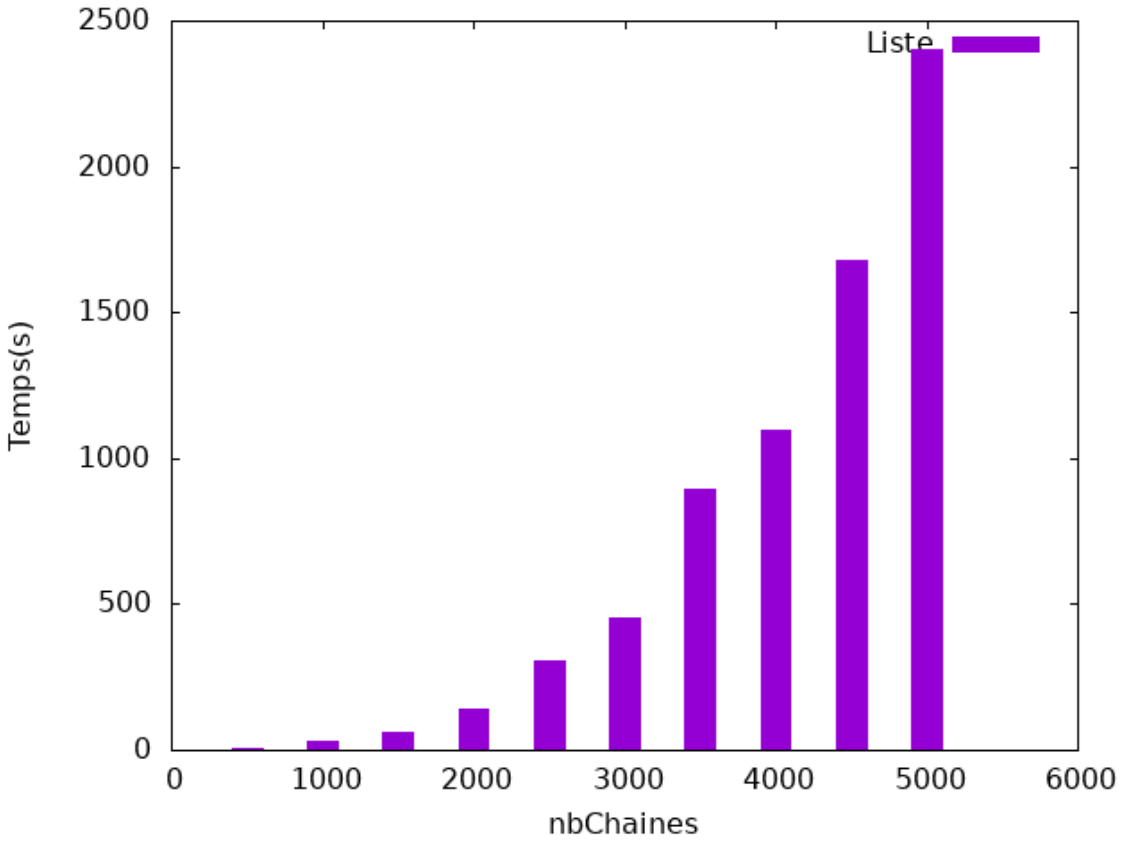
Réponses aux questions :

- **Q 6.1** :

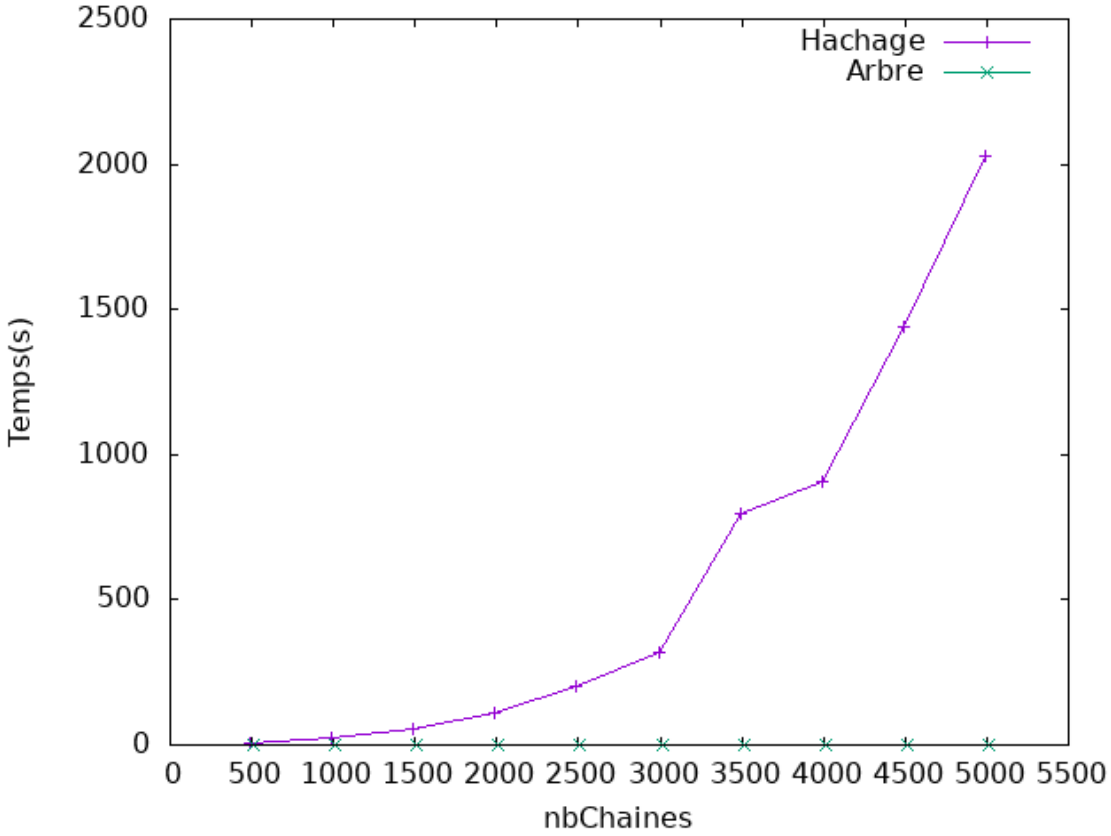
Les temps de calculs sont plus rapide pour les arbres en général, les listes chaînée moins optimisé et les tables plus rapide quand le nombre de chaîne est très élevé et la taille de la table est assez grand

Q 6.2 / Q 6.3 :

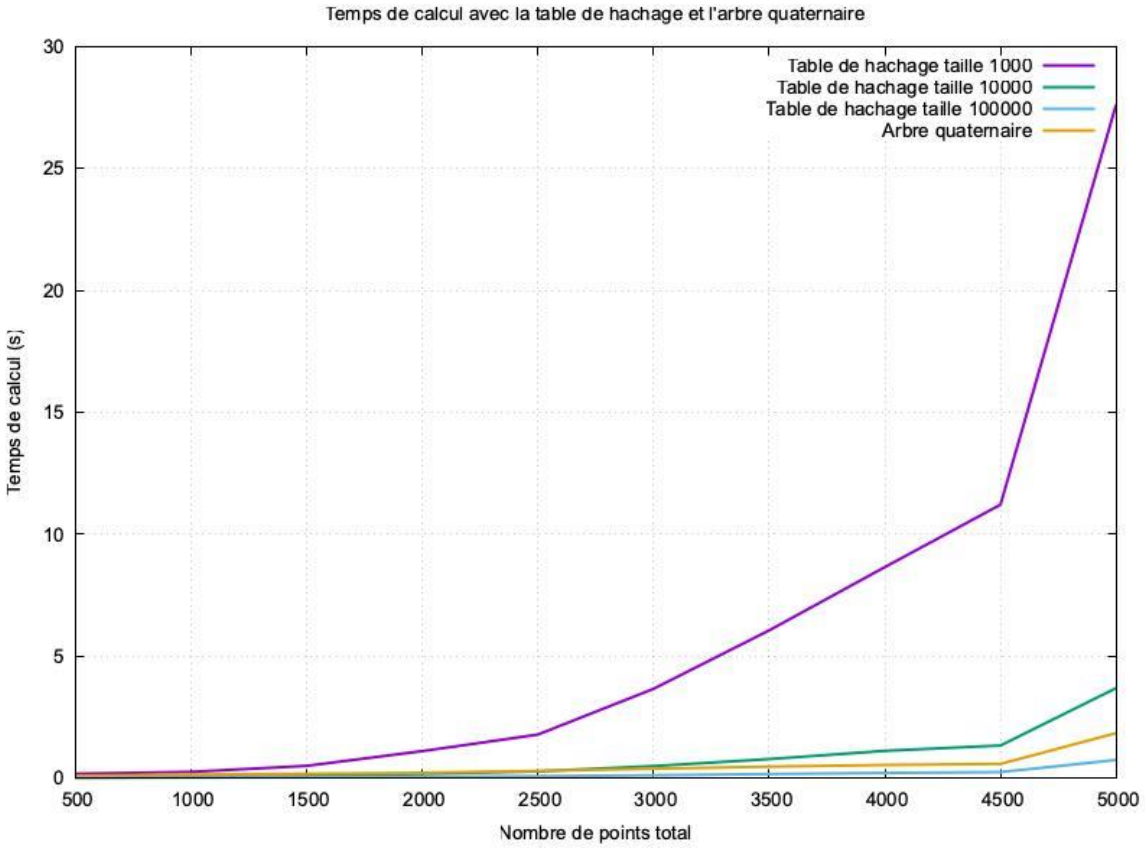
- Nous utilisons des données telles que nbPointsChaine = 100, xmax = 5000, ymax = 5000, en faisant varier le nombre de chaînes de 500 à 5000 par pas de 500.



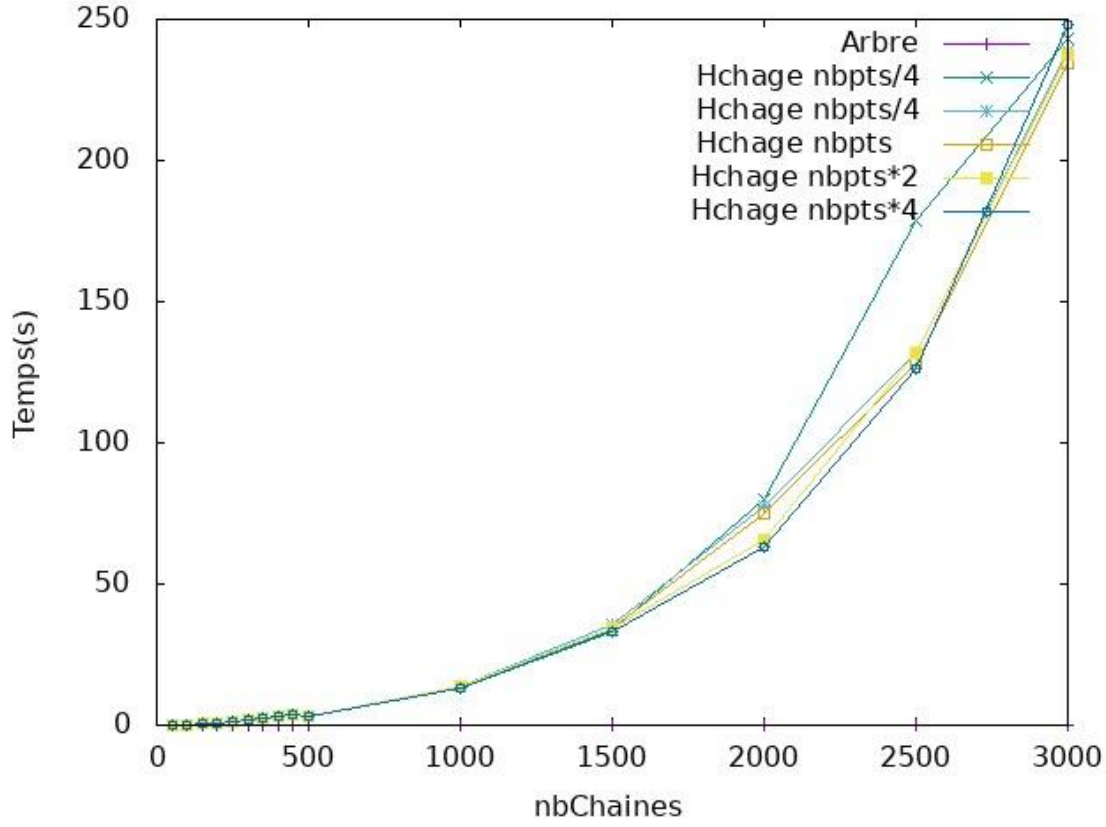
1.graphique donnant les temps de calcul avec la liste chaînée



2. **graphique comprenant les résultats obtenus avec la table de hachage et l'arbre quaternaire**



3. Graphique comprenant la méthode de hachage en variant la taille des arbres.



4.Graphique, taille des tables d'hachage en fonction de nombre total de points dans le réseau

Analyse des résultats :

On observe que dans la méthode de liste chaînée le temps de calcul augmente drastiquement au fil du nombre de chaînes dans le réseau.

Quant à la méthode de hachage, le temps de calcul augmente également mais de façon plus optimisé par rapport à la méthode des listes chaînés. Dans le 2e graphique, on a l'impression que le temps de calcul reste bien moins optimisé que celui des arbres. Cependant, en changeant le paramètre de la taille de la table en fonction du nombre de chaînes dans le réseau, la méthode s'avère être la plus efficace en observant le 3e graphe.

Enfin, la méthode de l'arbre quaternaire a un temps de calcul qui change très peu et reste très bas. Ce qui montre que cette méthode est assez eefficace.

Le 4e graphique montre également que les arbres quaternaire ont un meilleure temps de calcul pour des réseaux de tailles moyens.

On conclu que la structure et méthode la plus efficace dépend de la situation, l'arbre quaternaire s'en sort mieux dans les réseaux de taille petite et moyenne mais si l'on augmente beaucoup le nombre de chaine et la taille de la table (donc la taille du réseau), la méthode de hachage semble devenir plus efficace.

Jeux d'essais utilisés :

nbPointsChaine = 100, xmax = 5000 et ymax = 5000, en faisant varier le nombre de chaînes de 500 à 5000 par pas de 500

Partie 2: Réorganisation du Réseau

La deuxième partie du projet vise à réorganiser les attributions de fibres de chaque opérateur dans le réseau pour optimiser son utilisation et réduire les problèmes de sur-exploitation et de longueurs excessives.

Exercice 7 – Parcours en largeur (TME10)

Reformulation courte du sujet : Dans cet exercice, nous travaillons avec une structure de graphe fournie. Nous devons écrire des fonctions pour créer un graphe à partir d'un réseau, calculer le plus petit nombre d'arêtes d'une chaîne entre deux sommets, stocker l'arborescence des chemins, et réorganiser le réseau en vérifiant certaines conditions.

Réponses aux questions :

Q 7.4 :

…

Conclusion

En conclusion, ce projet a permis de concevoir et d'implémenter des algorithmes efficaces pour la reconstitution et la réorganisation d'un réseau de fibres optiques. Les différentes étapes ont été détaillées donnant un programme fonctionnel pour le projet.