Binome: Kashperuk Kristina, Kravchenko Alina

Rapport sur le projet : Réorganisation d'un réseau de fibres optiques

Ce rapport détaille le travail effectué dans un projet qui vise à utiliser des algorithmes pour remettre en état et réorganiser un réseau de fibres optiques.
Le projet est divisé en deux parties principales, chacune traitant des aspects spécifiques de la gestion du réseau.

Partie 1. Lecture, stockage, affichage des données et reconstitution du réseau

Exercice 1. Manipulation d'une instance de "Liste de Chaînes"

Liste chaînés:

- 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.
- Écrire des fonctions pour parcourir cette liste et reconstruire le réseau en établissant des liaisons entre les nœuds.

Cette méthode présente des limitations en termes de performances, (pour un grand nombre de chaînes dans le réseau.)

Des algorithmes

1. *lectureChaines(FILE f)
   Lit un fichier contenant une instance de "Liste de Chaînes" 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
3. *afficheChainesSVG(Chaines C, char * nomInstance)
   Créer un fichier SVG pour afficher les chaînes dans un navigateur.
4. *longueurChaine(CellChaine c)
   Calcule la longueur d'une chaîne (additionne les distances entre ses points)
5. *longueurTotale(Chaines C)
   Calcule la longueur totale des chaînes (somme des longueurs de toutes les chaînes)
6. *comptePointsTotal(Chaines C)
   Compte le nombre total d'occurrences de points dans toutes les chaînes.

Tests

1. La lecture et l'écriture de fichiers

- Utiliser des fichiers de données au format .cha avec différentes chaînes et points pour tester la fonction de lecture et d'écriture.

2. L'affichage graphique

- Créer des instances de chaînes avec des coordonnées pour vérifier que l'affichage graphique est correct.

3. Le calcul de longueur et de nombre de points:
- Créer des chaînes avec des points disposés de manière à faciliter le calcul de la longueur totale et du nombre de points.

Analyse des performances

En règle générale, le programme offre des performances satisfaisantes.

Exercice 2. Première méthode: stockage par liste chaînée

- implémenter l'algorithme de reconstitution de réseau en utilisant une liste chaînée
- 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

Des algorithmes

1. *rechercheCreeNoeudListe(Reseau R, double x, double y)
   recherche un nœud correspondant aux coordonnées (x, y) dans la liste chaînée des nœuds du réseau:
   Si lenœud 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)
   reconstitue le réseau à partir de la liste des chaînes C. Elle utilise la fonction rechercheCreeNoeudListe pour créer et relier les nœuds du réseau
3. main ReconstitueReseau.c
   Ce programme principal utilise la ligne de commande pour prendre un fichier .cha en paramètre et un nombre indiquant la méthode à utiliser pour la reconstitution du réseau (liste, table de hachage, ou arbre).

Tests

1. La création de nœuds et de liaisons :
   - Utiliser différentes configurations de chaînes pour tester la création des nœuds et des liaisons.
2. La reconstitution du réseau :
- Créer des instances de chaînes variées, vérifier que le réseau correspond à la structure initiale.

Analyse des performances

Les performances de ce programme varient en fonction de la taille du réseau et du nombre de chaînes à reconstituer. Si nécessaire, des améliorations pourraient être envisagées afin d'optimiser les performances dans des situations d'utilisation intensive.

Exercice 3. Manipulation d'un réseau

Tests

1. L’écriture du réseau dans un fichier
   - Créer des réseaux avec différentes configurations et vérifier que le fichier généré est correct.
2. 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 se changent avec la taille du réseau et du nombre de chaînes à reconstituer

Exercice 4 – Deuxième méthode : stockage par table de hachage

- structure de table de hachage qui associe chaque point à un index calculé à partir de ses coordonnées.
- gestion des collisions par chaînage pour stocker les nœuds d'un réseau.
- La table de hachage permet rapidement déterminer si un nœud a déjà été stocké dans le réseau

Des algorithmes

1. Structure TableHachage
   un tableau de pointeurs vers une liste de nœuds
2. int fonctionHachage(int k, int m)
   fonction de hachage pour générer les clés à partir des coordonnées (x, y) d'un point.
3. Noeud* rechercheCreeNoeudHachage(Reseau* R, TableHachage* H, double x, double y)
   fonction pour rechercher ou créer un nœud dans la table de hachage. (Si le point n'existe pas, nous le créons et l'ajouterons à la table de hachage et à la liste des nœuds du réseau)
4. Reseau* reconstitueReseauHachage(Chaines *C, int M)
   fonction pour reconstruire le réseau à partir d'une liste de chaînes en utilisant une table de hachage

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.)

Tests

1. 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. La recherche et la création de nœuds
- Utiliser des points déjà présents dans la table de hachage et des points nouveaux
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.

Exercice 5. Troisième méthode : stockage par arbre quaternaire

- arbre quaternaire peut diviser efficacement l'espace en quatre parties et organiser les nœuds du réseau en fonction de leurs coordonnées.
- Chaque nœud de réseau sera stocké au niveau des feuilles de l'arbre quaternaire, où chaque feuille représente une cellule

Des algorithmes

1. Fonction chaineCoordMinMax
   détermine les coordonnées minimales et maximales des points constituant les différentes chaînes du réseau
2. Fonction creerArbreQuat
   crée une cellule de l'arbre quaternaire avec les coordonnées du centre, la longueur et la hauteur
3. Fonction insererNoeudArbre
   insère un nœud du réseau dans l'arbre quaternaire. Elle gère trois cas : arbre vide, feuille et cellule interne, et divise récursivement la cellule en quatre (si nécessaire)
4. Fonction rechercheCreeNoeudArbre
   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
   reconstruit le réseau à partir de la liste des chaînes . Elle parcourt les chaînes, recherche ou crée les nœuds puis ajoute les liaisons et les commodités au réseau.

Tests

1. La création de l'arbre quaternaire
- Création de différents ensembles de points avec des configurations variées.
- création de l'arbre quaternaire avec ces ensembles de points pour évaluer son efficacité
2. 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. 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 se changent avec la structure et l'efficacité de l'arbre quaternaire et la qualité de la fonction de recherche et d'insertion.

Exercice 6. Comparaison des trois structures

Comparaison des temps de calcul de trois structures de données (la liste chaînée, la table de hachage et l'arbre quaternaire)

Réponses

- 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
- 6.2 et 6.3
  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 (slides)
2. graphique comprenant les résultats obtenus avec la table de hachage et l'arbre quaternaire (slides)

Analyse des résultats

- dans la méthode de liste chaînée le temps de calcul augmente au fil du nombre de chaînes dans le réseau.
- méthode de hachage. Le temps de calcul augmente, mais de façon plus optimisé (par rapport à la méthode des listes chaînés)

- La méthode de l’arbre quaternaire a un temps de calcul qui change très peu et reste très bas. Donc cette méthode est assez efficace. Les arbres quaternaires ont un meilleure temps de calcul pour des réseaux de tailles moyennes.

l’arbre quaternaire est mieux dans les réseaux de taille petite et moyenne
Si la taille du réseau est assez grande, la méthode de hachage sera plus efficace.

Tests
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

réorganiser les attributions de fibres de chaque opérateur dans le réseau pour optimiser son utilisation et réduire les problèmes d' exploitation et de longueurs excessives.

Exercice 7. Parcours en largeur

- 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