Sorbonne Université — Faculté des sciences
Année universitaire 2020-2021

Réorganisation d'un réseau de fibres optiques

SENIHJI WASSIM et **BILLA TONY**

Encadre par : Madame Magdaléna Tydrichova et Monsieur Raja Yehia

Table des matières

1 Préliminaires:Description globale du projet et de l'ensemble des fichiers

Le projet a été élaboré de la façon suivante :

- Le dossier bin contient les executables .
- Le dossier src contient les fichiers C
- Le dossier headers contient les fichiers .h

Pour compiler le programme on tape make dans le terminal , et pour l'execution , on execute chaque des executables suivant avec un format d'appel different :
ChaineMain.c : ./bin/ChaineMain
GrapheMain.c : /bin/GrapheMain ChaineTest.c : /bin/ChaineTest PerformanceReconstitution2 : /bin/PerformanceMain2 'nom_fichier_destination_rapport_liste' 'nom_fichier_destination_rapp
'nbPointsChaine' 'xmax' 'ymax' 'min_nb_chaines' 'max_nb_chaines' 'pas'
GenerationAleatoire2 : /bin/generationAleatoireFileMain2 'chemin_dossier_sauvegarde' 'nb_fichier_a_g
'xmax' 'ymax' 'nb_min_chaines' 'nb_max_chaines' 'nb_min_points' 'nb_max_points'

Pour éviter que le rapport soit long , chacune des fonctions utilisées et non mentionées dans le rapport est commentée lors de l'implémentation de la fonction dans les fichiers .c

2 Lecture,Stockage et affichage des données

2.1 Manipulation d'une instance de"Liste de Chaînes" ***(Exercice 1)***

Dans cette première partie, nous allons construire une bibliothèque de manipulations d'instances : lecture et écriture de fichier, affichage graphique de réseaux, calcul de la longueur totale des chaines, et calcul du nombre de points.

On implémente les fonctions *lectureChaine* qui permet d'allouer, de remplir et de retourner une instance de notre structure à partir d'un fichier , et *ecrireChaine* qui écrit dans un fichier le contenu d'une Chaines en respectant le même format que celui contenu dans le fichier d'origine dans le fichier Chaine.c
On teste ces 2 fonctions dans le fichier ChaineMain.c

On ajoute la fonction *afficheChaineSVG* qui permet de créer le fichier SVG en html à partir

d'une Chaines dans "Chaine.c", puis on la teste dans "ChaineMain.c" pour obtenir le fichier "chaine1.html".

On implémente les fonctions *longueurChaine* , *longueurTotale* et *comptePointsTotal* dans "Chaine.c"

3 Reconstitution du réseau

3.1 Stockage par lise chaînée *(Exercice 2)*

Dans cette partie , on désire implémenter l'algorithme de reconstitution de réseau en codant l'ensemble des noeuds du réseau par une liste chainée. On aura besoin de plusieurs fonctions manipuler les structures du fichier "Reseau.h" , pour cela on implémente les fonctions de :

1. **Création**:*CreerNoeud* , *CreerCellNoeud*(crée une Cellnoeud à partir d'un point), *CreerCellNoeudnd*(crée Cellnoeud à partir d'un noeud , et *CreerCommodite* .
2. **Désallocation**:*libererNoeud* , *libererercellNoeuds*, *libérerCommodite* , et *libererReseau*
3. **Ajout**: *rechercheVoisinAjoute* (qui vérifie si un noeud n1 est le voisin d'un autre noeud n2 , sinon n1 est ajouté à la liste des voisins de n2) , et *ajouterCellnoeud*,

On implémente la fonction *rechercheCreeNoeudHachage* qui retourne un Noeud du réseau R correspondant au point (x, y) dans la liste chainée noeuds de R.Si ce point n'existe pas dans noeuds , la fonction crée un nœud et l'ajoute dans la liste des noeuds du réseau de R
On implémente la fonction *reconstitueReseauListe* qui reconstruit le réseau R à partir de la liste des chaines C
On crée un menu qui permet à l'utilisateur de choisir quelle méthode de reconstitution utiliser et on implémente un programme main qui utilise la ligne de commande pour prendre un fichier .cha en paramètre et un nombre entier indiquant quelle méthode l'on désire utiliser
Toutes ces fonctions sont sur "Reseau.c" .

3.2 Manipulation d'un réseau *(Exercice 3)*

Dans cette partie , on veut présent construire des méthodes pour manipuler et afficher un struct Reseau. Pour cela, on va stocker sur disque un Reseau en implémentant la fonction *ecrireReseau* qui écrit dans un fichier le contenu d'un Reseau en respectant le même format du fichier "00014 burma.res"

On implémente les fonctions *nbCommodites*(compte le nombre de commodites dans le réseau) , *nbLiaisons*(compte le nombre de liaisons dans le réseau) et *ecrireReseau* (écrit dans un fichier le contenu d'un Reseau). On teste la fonction *afficheReseauSVG* sur la meme instance que dans la question 1.3 (00014burma) et on obtien le fichier "reseau1.html" qui correspond bien à "chaine1.html".

3.3 Stockage par table de hachage *(Exercice 4)*

Pour cette partie , nous allons utiliser une table de hachage avec gestion des collisions par chainage. La table de hachage va donc contenir un tableau de pointeurs vers une liste de noeuds .
On définit une structure TableHachage dans le fichier "Hachage.h" ,on implémente la fonction clef $f(x, y) = y + \frac{(x+y)(x+y+1)}{2}$ dans le fichier "Hachage.c" qu'on teste avec des valeurs

entre 1 et 10 .
Puis , on implémente la fonction de hachage $h(k) = \lfloor M(kA - \lfloor kA \rfloor) \rfloor ou A = \dfrac{\sqrt{5}-1}{2}$.

Par la suite , on aura besoin de quelque fonctions comme :
insererNoeudTable : Insere un noeud dans une table de hachage a la position pos
creerTableHachage: Crée une table de hachage de taille taille .
rechercheNoeudH : Recherche le noeud de coordonné 'x', 'y' dans la table 'H' et le met en tete de sa chaine , rend le NoeudH coorespondant si le noeud est dans la table et NULL sinon . Cette fonction aidera dans la fonction *recherchecreeNoeudHachage*
rechercheNoeudHnd : Comme *rechercheNoeudH* elle recherche le noeud 'nd' dans la table 'H' ,rend 1 si le noeud est dans la table et 0 sinon . *rechercheVoisinAjouteHachage* : Comme *rechercheVoisinAjoute* vu dans l'exercice 2 mais celle la correspond à un Noeudh qui est un Noeud qu'on relie à sa liste de Voisins dans une table H (la struct est implementé dans "Hachage.h") .
libererNoeudH : qui libère la mémoire alloué par un NoeudH .
libererTableHachage : qui libére la mémoire alloué par une table de hachage .
Finalement , on implémente la fonction *reconstitueReseauHachage* .

3.4 Stockage par arbre quaternaire *(Exercice 5)*

Pour cet exercice, un arbre quaternaire sera utilisé pour la reconstitution du réseau .
On implémente les fonctions *chaineCoordMinMax* et *creerArbreQuat* . Pour répondre à la suite des questions , on aura besoin de quelques fonctions outils .
La fonction *determine_emplacement* qui determine a quel emplacement doit etre insere le noeud 'n' dans 'parent' , et *rechercheNoeudArbre* qui retourne un Noeud correspondant au point (x, y) dans l'arbre quaternaire 'Aq' , si le noeud n'existe pas, le pointeur NULL est retourné . On ajoute une fonction de désallocation mémoire *LibererArbreQuat* qui libére l'espace memoire occupé par un arbre quaternaire 'Aq' ainsi que ses fils .
On peut finalement implémenter les fonctions *insererNoeudArbre* , *rechercheCreeNoeudArbre* , et *reconstitueReseauArbre* qui sont demandées dans l'exercice .

3.5 Comparaison des trois structures *(Exercice 6)*

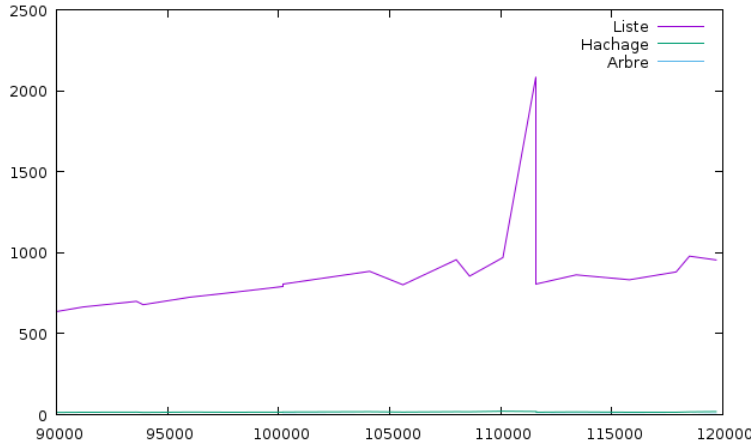
Dans cet exercice, on va comparer les temps de calcul obtenus avec les trois structures de données utilisées pour tester l'existence d'un noeud dans le réseau : la liste chainée, la table de hachage et l'arbre quaternaire.

On crée un programme main "PerformanceMain.c" ou on fait appel à la fonction *Performance* qui calcule le temps pris par les 3 fonctions de reconstitution un nombre n de fois et écrit les resultats obtenus dans un fichier dest . Pour les instances fournies , on execute ./bin/PerformanceMain ./docs/rapport_performance_reconstitution_fournis.txt 10 ./docs/chaines_fournis . On obtient le fichier rapport_performance_reconstitution_fournis.txt

.
Dans l'exercice 4 , il nous a été demandé de tester plusieurs valeurs de M (taille de la table de hachage) . On prend 5 valeurs differentes de M . M=500 ,M=nbChaines , M=nbChaines*5 , M=comptePointTotal(chaines)/2 , et M=comptePointTotal(chaines) . On remarque que pour nbChaines*5 est la plus rapide donc pour la suite on prend M= nbChaines*5 . Les resultats du test sont dans le fichier "rapport_performance_tableH.txt"

On implémente la fonction *generationAleatoire* qui permet de créer des chaenes de points et *generationFileAleatoire* qui retourne le flux d'un fichier generer avec 'nbChaines' chaines, 'nbPointsChaine' points et des coordonnees de point qui oscillent entre (0,xmax) pour x et (0,ymax) pour y elle retourne NULL en cas d'erreur . Ces deux fonctions sont dans "Chaine.c" .

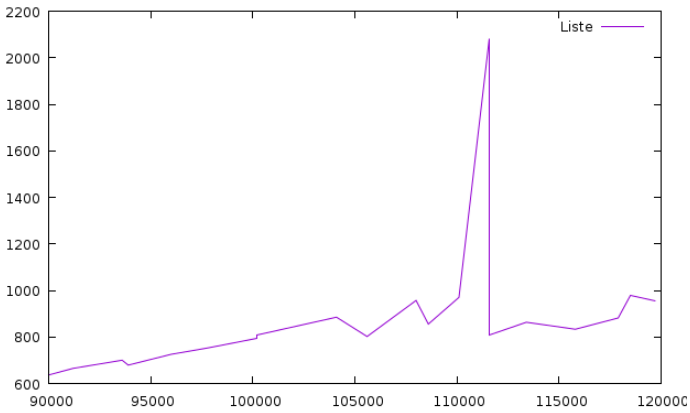
On teste ces deux fonctions en générant 20 fichiers contenant 300 chaines qui contient chacune entre 300 et 400 points qui varient entre 0 et 100 avec la commande ./bin/generationAleatoireFileMain2 ./docs/chaines_aleatoires 20 0 100 300 300 300 400 puis en générant le rapport "rapport-performance_reconstitution1.txt" avec la commande ./bin/PerformanceMain ./docs/rapport_perfromance 10 ./docs/chaines_aleatoires .



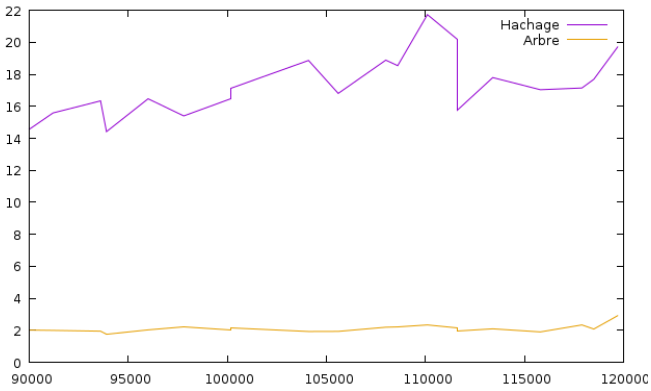
Courbe1(Liste;Arbre,Hachage)

On remarque que d'après la courbe 1 que le temps de reconstitution du reseau par une table de hachage et par un arbre est négligeable par rapport aux listes chainées de façon à ce qu'on arrive à peine de les voir dans la courbe . On en déduit que la reconstitution par Liste est beaucoup moins efficaces que les 2 autres .

D'après la courbe 3 , on remarque que le temps de reconstitution du reseau par Arbre est inférieur au temps de reconstitution par table de hachage . On en déduit que l'arbre est la structure la plus efficace .



Courbe 2 (Liste)



Courbe 3 (Arbre , Hachage)

4 Optimisation du réseau *(Exercice 7)*

Le but de cette partie est d'optimiser l'utilisation des fibres optiques du réseau. L'objectif est de relier les commodités par une chaine dans le réseau .
Pour la première question on implémente la fonction *creerGraphe* qui crée un graphe a partir d'un reseau 'r'.Pour la suite de l'exercice , on aura besoin de quelques fonctions comme : *liberer*C_*arete* , *liberer*Sommet , *liberer*Graphe , *creerTabCommod* qui crée un tableau de commodite a partir d'un reseau 'R' en les ajoutant a *'commodites' et retourne le nombre de commodites , et *maj_matrice_s_s* qui met a jour la matrice sommet-sommet 'matrice' de 'nbcolonne' avec les aretes composant la chaine 'chaine' si une des valeurs de la matrices est superieur a 'gamma' la fonction s'arrete et rend 0, la fonction rend 1 sinon .
Dans la deuxième question on implémente *plusCourChemin* qui retourne le plus petit nombre d'aretes d'une chaine entre deux sommets 'u' et 'v' d'un graphe 'g' .
De meme pour la troisième question on implémente *plusCourChemin2* qui retourne la plus petite chaine issue de u allant à v en stockant les chemins issus de u dans une ListeEntier dont la structure est fournie .
Pour la quatrième question , on implémente la fonction *reorganiseReseau* qui retourne vrai si pour toute arete du graphe, le nombre de chaines qui passe par cette arete est inferieur a y, et faux sinon.
On teste cette fonction sur "00014_burma.cha" , "05000_USA-road-d-NY.cha" et "07397_pla.cha" , on remarque que pour les 3 instances il existe une arete du graphe tel que le nombre de chaines qui passe par cette arete est inferieur à γ .

On remarque que pour une arete , le nombre de chaines qui passe par cette arete dépasse γ . La solution qu'on propose c'est de trouver un autre chemin dont la capacité (γ) n'a pas encore été atteinte , i.e un autre chemin dont le nombre de chaine n'est pas supérieur à γ .