Compte rendu projet final

LU2IN006

Membres du Binôme:

- *XIA Alexandre*
- *MAKHLOUF Rayane*

Groupe : n°2

Dans ce projet, nous manipulons un réseau de fibres optiques par le biais de structures telles que les listes, les tables de hachage, les arbres n-aires (quaternaire dans ce projet) et graphes. Dans un premier temps nous allons reconstruire avec les listes, les tables de hachage et les arbres quaternaires un réseau lié à une chaîne. Par la suite, nous organiserons ces réseaux à l'aide des graphes.

Exercice 1

Question 1.1

La fonction lit un fichier pour construire une chaîne, le code source est donnée dans le fichier Chaine.c (avec Chaine.h pour les prototypes et ChaineMain pour le main).

Question 1.2

La fonction va utiliser la chaîne pour réécrire les données du fichier de base (ou) d'une nouvelle chaîne générée, le code source se trouve dans le même fichier que la fonction lectureChaine (de la question précédente).

Question 1.3

Question1_3.html est le résultat de la fonction.

Question 1.4

La fonction longueurChaine est une fonction intermédiaire de la fonction longueurTotale qui calcule la longueur totale de la chaîne en utilisant la formule de Pythagore. $sqrt(\ ((xB\ -\ xA)\ +\ (yB\ -\ yA))$ * $((xB\ -\ xA)\ +\ (yB\ -\ yA))\ )$.

Question 1.5

Cette fonction passe dans la chaîne en comptant le nombre de structure Cellpoint pour chaque Cellchaine.

Exercice 2

Question 2.1

Cette fonction permet de rechercher un nœud dans un réseau à partir de ces coordonnées et de le renvoyer ou le créer si il n'existe pas et le rajouter au réseau tout en initialisant ces différents champs et en mettant à jour le réseau.

Question 2.2

Cette fonction permet de reconstituer un réseau de fibres optiques à partir d'une liste chaînée de chaînes .Ce fichier contient également les différentes fonctions de libération de la mémoire allouée pour le réseau.

Question 2.3

Le fichier ReconstitueRréseau.c contient les différents tests de vérification des fonctions précédentes. A l'exécution du fichier ReconstitueReseau , le programme demande à l'utilisateur de choisir à partir de quelle structure(entre liste chaînée , table de hachage et arbres) veut-il reconstituer le réseau ainsi que le nom du fichier contenant la liste des chaînes du réseau .

Exercice 3

Question 3.1

Les fonctions nbCommodites(Reseau *R) et nbLiaisons(Reseau *R) permettent respectivement de compter le nombre de commodités et de liaisons présentes dans le réseau passé en paramètre.

Question 3.2

La fonction ecrireReseau(Reseau *R, FILE *f) permet d'écrire sous le format indiqué au début de l'exercice 3, toutes les informations relatives au réseau passé en paramètre de l fonction dans un fichier préalablement ouvert au travers de la structure f(File*). Les résultats des appels à la fonction ecrireReseau(Reseau *R, FILE *f) sont stockés dans les fichiers Ex3_qst3_test1.txt , Ex3_qst3_test2.txt et Ex3_qst3_test3.txt en appliquant la fonction d'écriture du réseau à différents fichiers de tests. Ces résultats sont stockés dans le dossier Données.

Remarque: Les fonctions de cet exercice sont présentes dans le fichier reseau.c

Question 3.3

La fonction AffichageReseauSVG(Reseau *R,Char*nomInstance) a été récupérée depuis le fichier AffichageReseau.txt et a été utilisé pour l'affichage du réseau obtenu avec la fonction de reconstitution du réseau.

Exercice 4

Question 4.1

La structure est dans le fichier Hachage.h et contient, le nombre d'éléments, la taille et le tableau contenant les éléments.

Fichier source dans Hachage.c.

Question 4.2

La fonction utilisera la formule $(y + (x + y)(x + y + 1)/2)$ pour créer la clé associée.

Fichier source dans Hachage.c.

Question 4.3

La fonction de hachage utilisera la formule :

$$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$$

Dans les tests de l'exercice 6, il faudrait prendre la taille en fonction du nombre de points totaux (dans le cas où chaque point serait unique) afin d'avoir une complexité temporelle très faible. Cependant, la complexité spatiale peut être très élevée et certaines cases du tableau pourraient être vides (dues aux collisions, régler ce problème avec probing linéaire et adressage ouvert peut augmenter de manière significative la complexité spatiale).

Fichier source dans Hachage.c.

Question 4.4

On ajoute le nœud si on le ne trouve pas dans le réseau sinon on doit créer le nœud et l'ajouter au réseau (sans mettre les voisins et les commodités qui seront fait dans la question 4.5).

Fichier source dans Hachage.c

Question 4.5

On reconstruit le réseau sans oublier d'ajouter les voisins et les commodités après avoir ajouté une nœud dans le réseau. Fichier source dans Hachage.c.

Exercice 5

Question 5.1

Cette fonction va calculer les valeurs maximums et minimums pour l'abscisse et l'image. On utilisera des pointeurs qui récupéreront toutes les valeurs (on initialise au premier point rencontré).
Fichier source dans ArbreQuat.c.

Question 5.2

On alloue l'espace nécessaire pour stocker l'arbre (il faudra penser à libérer l'espace à la fin).
Fichier source dans ArbreQuat.c.

Question 5.3

On différencie les 3 cas : dans le cas Arbre vide on alloue l'espace nécessaire pour l'arbre et on l'ajoute par le biais du parent.
Dans le cas de la Feuille on garde l'ancien nœud dans une variable et on met à NULL le nœud de l'arbre courant avant de faire un appel récursif pour les 2 nœuds. Enfin dans le cas de la cellule interne, on fait un appel récursif vers la bonne cellule de l'arbre.
Fichier source dans ArbreQuat.c.

Question 5.4

On cherche le nœud dans l'arbre en fonction du cas, dans le cas de l'arbre vide le nœud n'existe pas, on le crée et insère. Dans le cas de la feuille, on vérifie si l'élément correspond aux coordonnées qu'on cherche sinon on insère le nœud. Enfin, dans le cas de la cellule interne on cherche en fonction des coordonnées du centre la cellule vers où faire l'appel récursif.
Fichier source dans ArbreQuat.c.

Exercice 6

Question 6.1

Pour la chaîne de l'exercice 1, on observe que pour la table de hachage de taille 100 la fonction de reconstruction est la plus rapide, suivi de la liste chainé, puis de l'arbre quaternaire et enfin de la table de hachage de taille 10000 (dû principalement à l'initialisation de toutes les cases).

Question 6.2

La fonction va générer une chaîne avec des coordonnées aléatoires pour x entre 0 et xmax et pour y entre 0 et xmax. Il ne faudra pas oublier de libérer l'espace à la fin
Le code source se trouve dans le fichier Chaine.c.

Question 6.3

Les graphes sont dans le répertoire Donnees/CourbeEx6/ avec le premier pour la liste chaînée et le deuxième pour la table de hachage et l'arbre quaternaire.

Question 6.4

Dans la reconstruction du réseau, plus le nombre de points est élevé, plus la complexité temporelle de la liste chaînée sera élevée, ce qui est aussi le cas d'une petite table de hachage. La reconstruction par l'arbre croît moins rapidement que les deux autres mais plus rapidement que pour la table de hachage de taille 10000 mais plus le nombre de points sera élevé plus la table de hachage de taille 10000 augmentera rapidement. La table de hachage de taille 10000 étant plus rapide que l'arbre, elle occupe un espace beaucoup plus important au niveau spatial. Donc, si l'espace n'est pas un problème, une table de hachage de taille convenable serait meilleure que l'arbre quaternaire, dans le cas contraire l'arbre serait une structure préférable.

Exercice 7

Question 7.1:

La fonction creerGraphe(Reseau *r) permet de créer un graphe représentant le réseau passé en paramètre de la fonction afin de mieux réorganiser ce dernier.

Question 7.2:

La fonction plus_petit_nombre_aretes(Graphe *g, int u, int v) permet de trouver le plus petit nombre d'arêtes entre deux sommets u,v du graphe. Dans cette fonction , on a fait appel à la structure de File fournie pour coder la bordure de manière efficace.

Question 7.3

On a divisé cette question en deux parties: Dans un premier temps on a implémenté une fonction pere(Graphe *g,int u,int s) qui permet de calculer et de rendre un tableau contenant à l'indice i le père du sommet de numéro i. Dans un second temps, on utilise le tableau renvoyé par la fonction précédente dans une fonction chemin(Graphe *g, int r, in s) qui permet de calculer et de renvoyer une liste d'entier correspondante au chemin entre les deux sommets r et s (l'arborescence) .

Question 7.4

La fonction reorganiseReseau(Reseau *r) crée un graphe à partir du réseau passé en paramètre, elle calcule la plus courte chaîne pour chaque commodité et stocke toutes ces chaînes dans un tableau de liste d'entiers. Ensuite, on stocke dans une matrice carrée le nombre de chaînes qui passe par chaque arête. Si le nombre de chaînes qui passe par une des arêtes de notre réseau est supérieur à gamma, alors notre réseau est surexploité sinon si pour toutes les arêtes, le nombre de chaînes qui passent par celles-ci est inférieur ou égale à gamma, alors notre réseau est bien exploité.

Question 7.5

Nos tests pour la réorganisation de réseaux de fibres optiques sont effectués dans le fichier ReconstitueReseau.c. Après l'analyse du code et des résultat, on peut proposer afin d'améliorer notre fonction de réorganisation de réseaux d'utiliser une matrice triangulaire au

lieu d’une matrice carrée pour stocker le nombre de chaînes passant par chaque arête , sachant que le nombre est le même pour les deux extrémités de chaque arête ce qui permet de faire des économies en termes d’espace mémoire et de temps de calcul.