FERCHE Adelin-Flaviu

COULIBALY Mariame

Grp 4

PROJET FINAL

Dans ce projet, nous nous interresions à la réorganisation d'un réseau de fibres optiques d'une agglomérations. Le projet se divise en deux étapes: la reconstitution du réseau et la réorganisation de celui-ci. De plus, pour la reconstitution, on a utilisé trois méthodes, à savoir , une reconstitution par liste chainée, par table de hachage et par arbre quaternaire, puis en comparant laquelle de ces méthodes est la plus adapter.

SOMMAIRE DES DOCUMENTS ENVOYES :

- Chaine.h **:**

    typedef struct cellPoint

    type def struct cellChaine

    type def struct Chaines

- Chaine.c :

    CellPoint* creer_point(double pos_x, double pos_y)
    CellChaine* creer_chaine(int num)
    Chaines* creer_ListeChaine(int nb_chaines, int gamma)
    void libererPoint(CellPoint* P)
    void libererChaine(CellChaine* C)
    void libererListeChaine(Chaines* LC)
    Chaines* lectureChaines(FILE *f)
    void ecrireChaines(Chaines *C, FILE *f)
    void afficheChainesSVG(Chaines *C, char* nomInstance)
    double longueurTotale(Chaines *C)
    int comptePointsTotal(Chaines *C)

- ChaineMain.c :

    int main(int argc, char** argv)

- Reseau.h :

    typedef struct Reseau

    typedef struct cellCommodite

typedef struct cellNoeud

typedef struct Noeud

- Reseau.c :

CellNoeud* creerCellNoeud(Noeud* N)
CellCommodite* creerCommodite(Noeud* A, Noeud* B)
void libererNoeud(Noeud* N)
void libererCellNoeud(CellNoeud* CN)
void libererCommodite(CellCommodite* CC)
void libererReseau(Reseau* R)
Noeud* rechercheCreeNoeudListe(Reseau *R, double x, double y)
Reseau* reconstitueReseauListe(Chaines *C)
void ecrireReseau(Reseau *R, FILE *f)
int nbLiaisons(Reseau *R)
int nbCommodites(Reseau *R)
void afficheReseauSVG(Reseau *R, char* nomInstance)

- Hachage.h :

typedef struct TableH

typedef struct CellNoeud

- Hachage.c :

CellNoeud* creerCellNoeudH(Noeud* N)

CellCommodite* creerCommoditeH(Noeud* A, Noeud* B)

TableH* creerTableHachage(int M)

void libererNoeudH(Noeud* N)

void libererCellNoeudH(CellNoeud* C)

void libererTableH(TableH* T)

double fonctionClef(double x, double y)

int fonctionHachage(double clef, int M)

Noeud* rechercheCreeNoeudHachage(Reseau* R, TableH* H, double x, double y)

Reseau* reconstitueReseauHachage(Chaines *C, int M)

- ReconstitueReseau.c :

void menu()

int main (int argc, char** argv)

- ArbreQuat.h :

    typedef struct ArbreQuat

- ArbreQuat.c :

    void chaineCoordMinMax(Chaines* C, double* xmin, double* ymin, double* xmax, double* ymax)
    ArbreQuat* creerArbreQuat(double xc, double yc, double coteX, double coteY)

- Makefile

- SVGwriter.h

- SVGwriter.c

Exercice 1 :

1.1) Il était question de créer un fichier Chaine.c dans laquelle on implémentait la fonction Chaines* lectureChaine(FILE *f); qui permettait d ‘allouer, remplir ainsi que de retourner une instance de la structure à partir de la lecture d’un fichier. Ainsi, nous avons créer des fonctions auxiliaires d’allocation pour les differente struct c’est-à-dire : CellPoint* creer_point(double pos_x, double pos_y), pour les points , CellChaine* creer_chaine(int num), pour les cellules contenant le point et le numero de la chaine , et Chaine* creer_reseau(int nb_chaines, int gamma), pour la liste de chaine. Enfin, nous avons pu implémenter la fonction Chaines lectureChaines(FILE *f) demandée.

1.2) Ici, il nous fallait d’abord implémenter une fonction qui permettait d’écrire dans un fichier la liste de chaine. Nous avons donc implémenter la fonction void ecrireChaine(Chaines *C, FILE *f), cette dernière parcourt la liste des chaînes et pour chacune on récupère leur numéro et on parcours la liste des points en récupérant les coordonnées. Puis, nous avons tester ces fonctions, en créant un fichier ChaineMain.c qui contient une fonction main qui quand elle est exécutée demande deux fichier , un pour la lecture et l’autre pour l’écriture.

1.3) Nous avons rajouté la fonction contenu dans le fichier texte dans le fichier Chaine.c, en n’oubliant pas d’inclure SVGwriter.h. La fonction void afficheChaineSVG(Chaines* C, char* nomInstance) nous permet de créer un fichier html contenant le graphique de la liste de chaîne. On la tester dans le main de ChaineMain.c

1.4) Les fonction à implémenter étaient : double longueurChaine(CellChaine *c); et double longueurTotale(Chaines *C); qui effectuait respectivement le calcul de la longueur d’une chaîne ainsi que la longueur totale de la liste de chaînes. La longueur d’une chaîne est la somme des distance entre deux points, donc la longueur de la liste de chaîne est la somme des longueur des chaines.

1.5) Pour celle-ci il a fallut implémenter une fonction qui renvoyait le nombre total d’occurence de points c’est à dire un compteur qui augmenté a chaque fois qu’un même point apparaissait plusieurs fois ; ce compteur étant renvoyés à chaque fois c’est la fonction int comptePointTotal(Chaine* C).

Exercice 2 :

2.1) Dans celle-ci il était question de l'implémentation d'une fonction Noeud* rechercheCreerNoeudListe(Reseau *R, double x, double y) qui pour un réseau R retournait un Noeud correspondant à un point de coordonnée (x,y) s'il existe ou qui le créer, le cas échéant, tout en l'ajoutant à la liste des nœuds du réseau. Ainsi, s'il le nœuds n'existait pas, on a crée des fonctions d'auxiliaires d'allocation pour créer le nœud et la cellule qui le contiendrait puis en l'insérant en tête de la liste des nœuds du réseau.

2.2) Ici la fonction Reseau* reconstitueReseauListe(Chaines *C) construit un réseau de zéro, à l'aide d'une liste de chaine,

2.3) La fonction main dans ReconstitueReseau.c demande à l'utilisateur lors de l'execution, le nom du fichier permettant de créer une liste de chaine. Puis, elle demandera à l'utilisateur de choisir entre 3 méthodes, liste chaîne, table de hachage et arbre, pour reconstituer un réseau. Ce fichier sera notre exécutable pour ce projet, on le mettra petit à petit à jour.

Exercice 3 :

3.1 On commence par implémenter les fonctions nbCommodites(Reseau *R); et int nbLiaisons(Reseau *R); qui respectivement compte le nombre de commodités et de liaisons dans le réseau qu'elles prennent en argument. A savoir qu' une liaison est la conséquence de deux points qui sont reliées par un câble, donc pour déterminer le nombre totale de liaison, on parcourra la liste des voisins de chaque nœuds.

3.2) La fonction void ecrireReseau(Reseau *R, FILE *f) écrit dans un fichier f le contenu du réseau en respectant la norme du fichier donnée en annexe.

3.3) De même que affichageChaineSVG, afficheReseauSVG permettra de créer un fichier html contenant le graphique du réseau.

Exercice 4 :

4.1) Dans un fichier Hachage.h, on définie une structure typedef struct TableH, constitué du nombre d'éléments totale dans la table, de la taille de la table et d'une liste de pointeur pointant sur des cellules contenant des nœuds. Cela nous permet d'implémenter une table de hachage.

4.2) Pour se faire, la table de hachage va parcourir la liste des points la fonction clé génère les clés des points en fonction des coordonnées (x,y). Ainsi, en fonction de la clé, la table saura si des nœuds ont déjà été stockés. Oui, la formule clef semble appropriée.

4.3) En développant la formule, h(k) = (int) (m*k*A – m*((int) k*A)). Cette première formule pour la fonction de hachage, nous permet d'avoir une table de hachage assez efficace.

4.4) La fonction Noeud* rechercheCreeNoeudHachage(Reseau* R, TableHachage* H, double x, double y); ainsi implémentée va renvoyer le noeud du réseau donné en argument qui correspond au coordonnée rentrée dans la table de hachage (s'il existe et le crée sinon).

4.5) Enfin la fonction Reseau* reconstitueReseauHachage(Chaines *C, int M) va reconstruire le réseau R, à l'aide d'une table de hachage de taille M, en fonction d'une liste de chaines C.

Exercice 5 :

5.1) On parcoure une liste de chaine, dont on parcoure pour chaque chaine la liste des points tout en mettant à jour les coordonnées minimales et maximales, grâce à des teste, des points de la liste de chaine.

5.2) Création d'un arbre quaternaire, en allouant et initialisant les struct qu'il contient.