ARNOLIN Aurélie

BOURSE Romain

THIEBAULT Adrien

THIBAULT Raphael

RAPPORT

Projet de langage C

POLYTECH - 2010 / 2011

Table des matières

[Introduction 2](#_Toc293876175)

[1. Définition et Choix des Structures 3](#_Toc293876176)

[1.1 Structure Ville 3](#_Toc293876177)

1[.2 Structure Noeud](#_Toc293876177) 4

1[.3 Structure Graphe](#_Toc293876177) 5

1[.4 Structure Liste\_ville](#_Toc293876177) 6

1[.5 Structure Arbre](#_Toc293876177) 7

2[. Organisation des dépendances](#_Toc293876177) 8

3. [Présentation des fonctions importantes (avec compléxités)](#_Toc293876177) 9

3[.1 Fonctions importantes présentes dans VILLE.C](#_Toc293876177) 9

3[.2 Fonctions importantes présentes dans NOEUD.C](#_Toc293876177) 10

3[.3 Fonctions importantes présentes dans GRAPHE.C](#_Toc293876177) 11

3[.4 Fonctions importantes présentes dans LISTE\_VILLE.C](#_Toc293876177) 12

3.5 Fonctions importantes présentes dans KRUSKAL.C…………………………………………….13

4[. Difficultées rencontrées et solutions envisagées](#_Toc293876177) .14

1. **Définition et choix des structures**
   1. **Structure Ville**

Etant donné que nous allons travailler avec un grand nombre de données (au moins 1000 villes) il nous faut donc une structure de données simple, qui permette de stocker un grand nombre d’informations et de travailler sur un large panel de données.

Ainsi nous avons décidé d’utiliser un tableau pour représenter une ville. Une ville est caractérisée par son nom, son nombre d’habitants, ses deux coordonnées GPS (latitude et longitude) et son indice (numéro dans le classement décroissant des villes par nombre d’habitants). Ce tableau contient donc 5lignes.

STRUCTURE VILLE :

|  |
| --- |
| NOM de la ville (stringchar) |
| NOMBRE D’HABITANTS de la ville ( int ) |
| COORDONNEE 1 de la ville ( float ) |
| COORDONNEE 2 de la ville ( float ) |
| INDICE de la ville ( int ) |

* 1. **Structure Noeud**

La structure Nœud est une liste simplement chainée dont chaque élément est composé d’une structure Ville, d’un pointeur sur Liste\_Ville (définie plus loin) et d’un pointeur sur un autre Noeud. Nous avons utilisé cette structure car elle permettait de trier directement le fichier d’entrée et donc de répondre plus facilement au moins aux deux premières questions du projet.

Elément d’un Noeud

Ville

Ville

Ville

Liste\_Ville

* 1. **Structure Graphe**

La structure Graphe est un pointeur sur un Nœud. Et comme chaque nœud est composé d’un pointeur sur le Nœud suivant, donc le graphe représente l’ensemble des Nœuds et chaque Nœud représente un des sommets du graphe.

Noeud

Ville

Graphe

* 1. **Structure Liste\_Ville**

La structure Liste\_Ville est une liste simplement chainée dont chaque élément est composé d’une structure Ville, d’un float représentant la distance entre la ville et son voisin et d’un pointeur sur un autre élément.

Ville Float

Ville Float

Ville Float

* 1. **Structure Arbre**

La structure Arbre est une liste simplement chainée dont chaque élément est composé de deux structures Ville, d’un float représentant la distance entre les deux villes et d’un pointeur sur un autre élement.

Ville 1 Ville 2 Float

Ville 1 Ville 2 Float

Ville 1 Ville 2 Float

1. **Organisation des dépendances**

**Schéma des dépendances**

Certains programmes font appel à certaines fonctions présentes dans d’autres fichiers. Ainsi les fichiers sont dépendants entre eux. Ainsi on remarque que le programme principal main.c dépend de tous les autres programmes.

Cette analyse permet donc de créer un makefile correct (ci-joint dans le dossier)

**Main.c**

**Nœud.h**

**Ville.h**

**Menu.h**

**Graphe.h**

**Ville.h**

**Nœud.h**

**Kruskal.h**

**Liste\_ville.h**

**Graphe.h**

**Graphe.h**

**Kruskal.h**

**Liste\_ville.h**

**3. Présentation des fonctions importantes**

* 1. **Fonctions importantes présentes dans VILLE.C**
* **Fonction Afficher\_Ville**

-- prend une ville en paramètre et affiche ses différentes caractéristiques à savoir son nom, son nombre d'habitants, ses coordonnées GPS, son indice

* **Fonction conversion\_coordonnee**

-- convertit une coordonnée de type caractère (00.00000A) en une coordonnée de type float

-- renvoie un flottant (négatif si la chaine en entrée finit par ou S)

* **Fonction distance**

-- prend en paramètre deux villes et renvoie un float

-- calcule la distance entre deux villes données

* 1. **Fonctions importantes présentes dans NOEUD.C**
* **Fonction Afficher\_Noeud**

-- prend un pointeur sur Noeud en paramètre et affiche la ville courante

* **Fonction Ajouter\_Fin -- O(n)**

-- insère les villes les unes après les autres dans le graphe, triées par ordre décroissant du nombre d’habitants (ajout+tri)

* **Fonction Completion**

-- prend en paramètre un pointeur sur caractères et un pointeur sur Noeud

-- affiche les noms des villes commençant par les caractères donnés en paramètre

* **Fonction Rechercher\_Noeud**

-- prend en paramètre un nom de ville et renvoie un booléen

* 1. **Fonctions importantes présentes dans GRAPHE.C**
* **Fonction Afficher\_Graphe**

-- prend un pointeur sur Noeud en paramètre et affiche le graphe

* **Fonction Generer\_Graphe -- en O(n²)**

-- prend un pointeur sur Noeud en paramètre et génère la liste des voisins de chaque ville

* **Fonction PCC (Plus Court Chemin) --en O(nlog(n))**

-- calcule le chemin le plus court entre deux villes données

* **Fonction indice**

-- affiche l'indice associé au nom de la ville

-- Les villes sont classées par ordre décroissant du nombre d'habitants et on leur attribue un indice en conséquence.

Ex : Paris 3345565565 habitants indice 0

Marseille 213553323 habitants indice 1

Lyon 56556556 habitants indice 2

* **Fonction Villesnecessaires**

-- renvoie un tableau contenant les villes nécessaires c'est à dire les villes dans lesquelles le petit avion doit absolument s'arrêter pour faire le plein de kérosène

* 1. **Fonctions importantes présentes dans LISTE\_VILLE.C**
* **Fonction Afficher\_voisins**

-- prend en paramètre un pointeur sur Liste\_Ville et renvoie les voisins de cette Liste\_Ville

* **Fonction Ajouter\_debut\_voisins**

-- ajoute un voisin au début de la liste des voisins

* 1. **Fonctions importantes présentes dans KRUSKAL.C**

**Fonction Arbre\* RemplirArbre(Arbre\* A,Noeud\* N,int M)**

--remplit un arbre (liste) en fonction des distances croissantes entre les N plus grandes villes

**Fonction Arbre\* Ligne(Arbre\* A,int N, Noeud\* No)**

--Transforme l’arbre généré par RemplirArbre à fin qu’il ne présente pas de cycle

**Fonction int Est\_sans\_circuit(int N, float Mat[N][N])**

--Teste s’il y a un circuit dans la matrice d’adjacence

**Fonction Arbre\* quick\_sort (Arbre\* list) –en O(nlog(n))**

--est utilisé dans RemplirArbre lors du tri des distances

**4. Difficultés rencontrées et solutions envisagées**

Concernant la question 1 c’est-à-dire la lecture des données d’entrée à partir d’un fichier CSV, nous avons eu un problème au niveau des sauts de ligne : des sauts de ligne apparaissaient au début du nom de chaque ville. Ainsi nous avons été obligés de comparer manuellement caractère par caractère tout au long du projet.

Concernant la question 3, nous avons rencontré des difficultés concernant la génération du graphe. En effet, il fallait générer des listes de listes et nous avons dû chercher des algorithmes plus rapides et plus efficaces.

Concernant la question 4, c’est-à-dire calculer l’arbre aérien couvrant minimal, nous avons eu des problèmes concernant la rapidité de l’algorithme utilisé. En effet l’algorithme fonctionne bien pour 300 villes environ mais est trop long pour un nombre plus important de villes.