**2011/2010**

Sommaire

[2) Présentation du problème et principe de notre analyse 3](#__RefHeading__1664_1099940372)

[3) Méthodes de travail 7](#__RefHeading__1666_1099940372)

[→](#__RefHeading__1668_1099940372) GOOGLE DOCUMENT: 7

[→](#__RefHeading__1670_1099940372) GITHUB : 7

[Présentation du code et Implémentation de la solution 8](#__RefHeading__1672_1099940372)

[1) Les structures 8](#__RefHeading__1674_1099940372)

[2) Les fonctions importantes 8](#__RefHeading__1676_1099940372)

[CONCLUSION 13](#__RefHeading__1678_1099940372)

[Nos erreurs ou difficultés 13](#__RefHeading__1680_1099940372)

[Autre solution possible 13](#__RefHeading__1682_1099940372)

[ANNEXE : 14](#__RefHeading__1684_1099940372)

INTRODUCTION  
  
1) Présentation du projet

**L**e projet de C 2010/2011 s’inscrit dans le cadre de données géo localisées et de cartes, permettant de positionner un objet (une personne,...) à l'aide de ses coordonnées géographiques.

**L**a géo localisation (ou géo référencement) permet, plus spécifiquement, à l'aide d'un récepteurGPS (ou à d'autres techniques) de publier en temps réel ou de façon différée ses coordonnées géographiques : latitude et longitude.

Ces données sont généralement diffusées et utilisées via internet.

Elles peuvent être utilisées pour des applications professionnelles (ex : Transport de marchandises ou de passagers - Logistique - Suivi et protection de personnes - Propreté urbaine...) comme pour des applications personnelles (Suivi et protection de personnes - Sécurité des biens personnels (voitures, smart phone, plaisance...) - etc.).

**O**n peut notamment  trouver sur le site <http://www.world-gazetteer.com/> les données de toutes les grandes villes de tous les pays avec leur population et les coordonnées GPS.

## 2) Présentation du problème et principe de notre analyse

**L**e but principal de ce projet est d’adapté des données géo localisées à partir de fichiers de format différents afin de pouvoir les traiter via des programmes C.

**L**es différentes parties de ce projet visent à atteindre ce but.

Nous commençons tout d’abord par définir les différents types de fichier qui seront utilisés dans ce projet afin de bien les cerner.

CSV (Comma-separated values)

* Il s’agit d’un [format informatique](http://fr.wikipedia.org/wiki/Format_informatique) [ouvert](http://fr.wikipedia.org/wiki/Format_ouvert) représentant des données tabulaires sous forme de  valeurs séparées par des [virgules](http://fr.wikipedia.org/wiki/Virgule).
* Il est utilisés autour du [logiciel](http://fr.wikipedia.org/wiki/Logiciel) [Microsoft](http://fr.wikipedia.org/wiki/Microsoft_Excel) EXCEL mais  les séparateurs ne sont pas standardisés (virgules, [points-virgules](http://fr.wikipedia.org/wiki/Point-virgule) ) ce qui rend ce format peu pratique pour une utilisation autre que des échanges de données ponctuels.
* Cependant il demeure assez populaire car il est relativement simple et facile à générer.

HTML (HyperText Mark-Up Language)

* Langage dit de « marquage » (de « structuration » ou de « balisage ») dont le rôle est de formaliser l'écriture d'un document avec des balises de formatage.
* Les balises permettent d'indiquer la façon dont doit être présenté le document et les liens qu'il établit avec d'autres documents.
* Le langage HTML permet notamment la lecture de documents sur [Internet](http://www.commentcamarche.net/contents/internet/internet.php3) à partir de machines différentes, grâce au [protocole HTTP](http://www.commentcamarche.net/contents/internet/http.php3), permettant d'accéder via le réseau à des documents repérés par une adresse unique, appelée [URL](http://www.commentcamarche.net/contents/internet/url.php3).

XML (extensible Markup Langage)

* C’est en quelque sorte un langage [HTML](http://www.commentcamarche.net/contents/html/htmlintro.php3) amélioré permettant de définir de nouvelles balises. Il s'agit effectivement d'un langage permettant de mettre en forme des documents grâce à des balises (markup).
* Par conséquent, pour stocker ou gérer des informations numériques (texte, image, vidéo, son), XML peut être une solution très  intéressante.
* C’est aussi un format  d'échange de données par excellence.

**U**ne fois ces langages définis, nous avons pu étudier les grands principes de ce projet et en faire une rapide analyse que nous vous présentons de manière brève:

→Lire les données d’entrée à partir d’un fichier CSV

→Afficher les X plus grandes villes sur Google Maps

→Calculer le plus court chemin entre deux villes

→Calculer l’arbre recouvrant minimal

→Lire des fichiers d’entrée dans un format plus complexe

## 3) Méthodes de travail

**N**ous vous présentons dans cette partie un petit résumé des diverses méthodes de travail que nous avons établi pour mener à bien notre projet.

Dans un premier temps les réunions :

**N**ous avons établi un planning de réunion auxquels nous avons tenté de nous tenir compte tenu de notre emploi du temps respectif.

**L**ors de ces réunions nous accomplissions les taches communes (discutions autour de notre approche du problème etc.), nous répartissions les taches à accomplir ainsi que l’échéance, et avons tenté de nous y tenir.

Dans un second temps l’utilisation d’outils de travail collaboratif :

### → GOOGLE DOCUMENT:

**N**ous avons utilisé Google document pour des raisons de rapidité et d’ergonomie dans le but de partagé des informations (résumé de réunion, principe d’utilisation de programmes...).

### → GITHUB :

**G**ITHUB est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programmegit.

**N**ous avons utilisé la plateforme [http://github.com](http://github.com/) pour partager nos codes source de manière efficace et  indépendante.

# Présentation du code et Implémentation de la solution

**D**ans cette partie nous allons vous présentez des morceaux de code significatifs, l’implémentation de la solution que nous avons réalisée ainsi que les choix que nous avons fait et les difficultés que nous avons rencontré.

## 1) Les structures

**N**ous vous présentons ici l’ensemble des structures que nous avons élaboré qu’on les ait utilisé ou non.  
  
 **D**ans la première partie de ce projet, il nous été demandé d’enregistrer l’ensemble des informations, composant l’identité d’une ville, présentes dans un fichier en format CSV afin de pouvoir les utiliser de manière efficace dans la suite.

**N**ous avons donc décidé d’utiliser la structure suivante:

**Struct ville**{

char nom\_ ville[100]; // représentant le nom de la ville

int pop; // représentant la population de la ville   
 float latitude; // représentant la latitude de la ville  
 float longitude; // représentant la latitude de la ville  
};  
**typedef struct ville Ville;**  
  
 **L**a structure que nous avons choisi comporte un tableau de caractères de taille 100 contenant le nom de la ville stockée, un entier représentant la population locale et deux flottant stockant la latitude et la longitude.  
  
 **N**ous avons préféré définir un tableau de taille 100 plutôt qu’un pointeur sur un tableau pour des questions de facilité d’enregistrement et d’utilisation dans les fonctions que nous avons écrites.

**N**ous avons du utilisé des flottants pour décrire les coordonnées des villes car pour pouvoir utiliser Google Map et en particulier le format HTML, les données sont enregistrées comme tels. Nous avons utilisé cette structure dans la plupart de nos fonctions par la suite.

**L**e choix de cette structure a donc été primordiale et nous avons du définir nos priorités la concernant. Elle semblait relativement évidente à choisir et ne nous donc pas poser particulièrement de problème dans le choix de sa conception.  
  
 **D**ans un second temps, nous avons été confrontés à la possibilité d’utiliser une autre structure dans la troisième partie du projet, à savoir, le calcul du plus court chemin entre deux villes « en petit avion ».  
En effet, dans cette partie nous avons été amenés à identifier et utiliser nos données comme étant des sommets de graphes.  
Il s’est donc imposé à nous la possibilité de créer une structure de graphe :  
  
**Struct graphe** {  
  
   char X [1000]; //ensemble des sommets du graphe   
   Arc U [10000]; //ensemble des arcs du graphe  
};  
**typedef struct graphe Graphe**;  
  
 **C**ette structure en utilisant une autre, des arcs :  
  
**struct arc** {  
  
   Ville source ; // ville de départ  
   Ville destination ; // ville d'arrivée  
   float distance; //distance entre la source et la destination  
};  
**typedef struct arc Arc**;  
  
  
 **C**ependant nous n’avons finalement pas retenu cette solution ne la trouvant pas optimale en terme de temps comparée à l’utilisation d’une simple matrice pour représenté l’ensemble des sommets ( en l'occurrence des villes) que nous avons traité.

**E**n effet, nous l’expliquerons par la suite, nous avons  fait le choix de privilégier la vitesse d’exécution de nos programmes et par conséquent la rapidité d’accès aux informations stockées.

## 2) Les fonctions importantes

**D**ans cette partie nous allons nous intéresser aux différents programmes que nous avons implémentés en nous concentrant sur la fonction de ces programmes, les fonctions intéressantes qui les composent, ainsi qu’une explication de nos choix et des performances les concernant.  
  
 **N**ous vous présentons les fonctions principales en bleu et les morceaux de code pertinents *en violet.*  
  
→ Lecture d’un fichier csv et écriture dans un html

Module de manipulation de fichier

**L**e premier module du projet est celui concernant les fichiers CSV et HTML.  
  
  
Dans le module MODULE\_FICHIER

#include"MODULE\_GLOBALE.h"  
#ifndef MODULE\_FICHIER\_H\_INCLUDED  
#define MODULE\_FICHIER\_H\_INCLUDED  
  
FILE\* lire\_fichier(char\* path\_fichier);  
Ville\* read\_ville(FILE\* fp, int\* nombre\_de\_ville);  
void ecrire\_fichier(Ville \*ville, FILE \*fp\_origine\_html, FILE \*fp\_new\_html, int x);  
  
#endif  
  
 **D**ans ce module nous avons utilisé trois fonctions principales:

- Une fonction de changement de format pour traduire les coordonnées des villes du fichier CSV à la version HTML.

float change\_format(char\* str);  
   
*switch (str[i])*  
*{*  
*case 'S':*  
*case 'W':*  
*//printf("je traite les S et W");*  
*finale= (atof(str\_new) \* (-1.0));* // utilisation de la fonction atof(char c)   
*break;*  
*default:*  
 *//printf("Je suis dans default");*  
*finale=atof(str\_new);*  
*break;*  
*}*  
  
 **N**otez l’utilisation de la fonction atof qui transforme les chaines de caractères en flottants.

**E**lle nous a été utile et est très simple d’utilisation.

- une fonction de lecture de fichier csv.

FILE\* lire\_fichier(char\* path\_fichier);  
Ville\* read\_ville(FILE\* fp, int\* nombre\_de\_ville);  
   
*Ville \*ville=(Ville\*)malloc(\*nombre\_de\_ville\*sizeof(Ville));*

//utilisation d’un tableau dynamique pour enregistrer l’ensemble de nos données

*ville[i].pop=atoi(token);*  
*ville[i].latitude=change\_format(token);*   
*token = strtok( NULL, "," );*  
  
 **U**tilisation des fonctions atoi (provenant de la bibliothèque <string.h>) qui transforme une chaîne de caractères en entier, change\_format (décrite précédemment) et strtok qui découpe une chaîne de caractères.  
  
 **L**ors de l’amélioration de ce programme, nous avons tenté d’importer les villes dans un tableau dynamique, qui verrait sa taille incrémenté à chaque fois qu’une ligne décrivant une ville est lue dans le fichier source csv. Cependant la fonction Realloc de la bibliothèque <stdlib.h>, permettant cette méthode, engendrait une quantité considérable d’erreurs dont notamment des fuites mémoires et des erreurs de segmentation. Après consultation auprès des professeurs, ces derniers nous ont conseillé d’éviter la fonction au résultat aléatoire dans la gestion de mémoire et au temps d'exécution important. Comme vous le constaterez, il a été décidé par l’équipe d’utiliser un seul appel à la fonction Malloc et deux parcours du fichier csv: le premier parcours destiné à obtenir la taille du tableau immédiatement suivi de l’allocation correspondante en mémoire avec l’appel au malloc; le deuxième parcours qui affecte chaque ligne lu dans le fichier source à un élément du tableau.

- une fonction d’écriture en format html.

void ecrire\_fichier(Ville \*ville, FILE \*fp\_origine\_html, FILE \*fp\_new\_html, int x);  
  
*fprintf(fp\_new\_html, "var coordinates = [\n");*

// modification du tableau de coordonnées avec utilisation de la structure VILLE  
  
*for(i=0; i<x; i++)*   
*{*  
*fprintf(fp\_new\_html, "[%d, %.2f, %.2f], \n", i+1, ville[i].latitude, ville[i].longitude);*  
*}*  
  
 **N**otez ici l’utilisation de notre structure de manière efficace surtout en terme de récupération des données, d’où notre choix de les conserver dans un tableau dynamique plutôt qu’une liste.

RESULTAT:  
  
**** Image représentant  les 11 plus grandes villes du fichier

→ Calculer le plus court chemin entre deux villes.

Module Dijkstra

**D**ans ce module nous nous sommes inspirés de notre cours de graphe afin de comprendre comment réaliser ce programme.

**N**ous avons notamment étudié la possibilité d’utiliser une structure de graphe mais avons finalement rejeté cette idée pour des raisons de rapidité d’exécution et de récupération de données.

**A** la place nous nous sommes concentrés sur l’utilisation d’une matrice d’entiers.

**E**n effet plutôt que d’utiliser directement notre structure de Ville nous avons travaillé exclusivement sur des entiers et récupérons les données relatives aux villes grâce à leur position dans le tableau dynamique de Ville que nous avons créé précédemment.  
  
 **A**insi nous avons créer un programme suivant ce schéma (haut niveau):  
  
**procédure** *Dijkstra*( *M* (matrice des distances) , *dep* , *arv* )  
*Initialisation:*  
  
*c* ← {*dep*} ; *d*[*dep*] ← 0  
**pour** tout sommet *k* **faire**  
   **si** *k≠ dep* **alors** *d*[*k*]← *M*[*dep*,*k*] ; *pred*[*k*]←*dep* **fin si**  
**fin pour**  
  
*Construction de l'ensemble c:*  
**tant que** cardinal(*c*)< *n* **faire**  
   trouver *y* tel que *d*[*y*] = min{*d*[*k*];  *k* sommet tel que *k* n'appartient pas à c}  
**si** ce minimum est ∞ **alors** sortir de la boucle **fin si**  
   *c*← *c union* {*y*}  
 **pour** chaque sommet *k* n'appartient pas à c **faire**  
**si** *d*[*k*] > *d*[*y*] + *M*[*y*,*k*] **alors** *d*[*k*] ← *d*[*y*] + *M*[*y*,*k*]  ; *pred*[*k*] ← *y* **fin si**  
**fin pour**  
**fin tant que**  
  
*Construction du chemin le plus court:*  
*ch*←[];   *k*←*arv*;     
**tant que** *k ≠ dep* **faire**  
    *ch* ← [*k*,*op(ch)*];  *k*←*pred*[*k*]     
**fin tant que**  
*ch* ← [*dep*,*op(ch)*]  
**si** *d*[*arv*]=∞ alors  *pas de chemin possible*  
**sinon**  
    afficher *ch* (*chemin le plus court* ) et  *d*[*arv*] (*la distance correspondante*)  
**fin si**  
**fin procédure**

**V**oici les spécifications fonctionnelles de ce module:  
  
 Dans le module MODULE\_DIJKSTRA

#include"MODULE\_GLOBALE.h"  
#ifndef MODULE\_DIJKSTRA\_H\_INCLUDED  
#define MODULE\_DIJKSTRA\_H\_INCLUDED  
  
float\* creer\_table\_distance(Ville ville\_depart, Ville \*ville, int nombre\_de\_ville);

int\* creer\_table\_marque(int nombre\_de\_ville);

int\* creer\_table\_pere(int nombre\_de\_ville);

int extraire\_min(float \*table\_distance, int \*table\_marque, int nombre\_de\_ville);

//retourne le numéro de ville extraire.

int\* extraire\_update\_table\_marque(float \*table\_distance, int \*table\_marque, int nombre\_de\_ville);

float \*relacher\_table\_distance(float \*table\_distance, int x, int y, Ville \*ville);

int\* relacher\_table\_pere(float \*table\_distance, int \*table\_pere, int X, int Y, Ville \*ville);

int toutes\_villes\_extraites(int \*table\_marque, int nombre\_de\_ville);

int cherche\_chemin\_1(Ville \*ville, FILE \*fp\_new\_html, int \*table\_pere, int No\_de\_depart, int No\_de\_dest, int i);

int cherche\_chemin\_2(FILE \*fp\_new\_html, int \*table\_pere, int No\_de\_depart, int No\_de\_dest, int i);

void ecrire\_fichier\_chemin(Ville \*ville, FILE \*fp\_origine\_html, FILE \*fp\_new\_html\_2, int \*table\_pere, int No\_de\_depart, int No\_de\_dest);  
  
#endif

**N**ous vous présentons maintenant des bouts de code pertinents à expliquer:

* **float distance(Ville ville1, Ville ville2);**
  + ---pré: 2 structures villes: ville1 et ville 2
  + ---post: donnant la distance entre 2 villes

*float d = acos(sin(lat1\*PI/180.0)\*sin(lat2\*PI/180.0) + cos(lat1\*PI/180.0)\*cos(lat2\*PI/180.0) \*cos((lon2-lon1)\*PI/180.0)) \* R;*  
*return d;*  
*…*  
  
 **C**ette formule a été établie à partir des renseignements récoltés sur le site : http://www.movable-type.co.uk/scripts/latlong.html

* **float\* creer\_table\_distance(Ville ville\_depart, Ville \*ville, int nombre\_de\_ville);**
  + ---post: créer un tableau table distance qui enregistre les distances entre la ville\_depart et tous les autres villes. Exemple: table\_distance[10] est la distance entre ville\_départ est ville[10]

*float \*table\_distance=malloc(sizeof(float)\*nombre\_de\_ville);*  
*for(i=0; i<nombre\_de\_ville; i++)*  
*{*  
*if( (ville[i].latitude == ville\_depart.latitude) &&(ville[i].longitude == ville\_depart.longitude) )*  
*table\_distance[i]=0.0;* // on intialise la ville de départ a 0  
*else*  
*table\_distance[i]=99999.0;* // on initialise toutes les autres villes a 99999

*}*

**O**n alloue dynamique de l’espace mémoire pour créer un tableau.  
  
 **L**’un des soucis de notre programme est l’allocation dynamique de tableaux (marque, distance et père) dont l’espace mémoire ne peut être libéré qu’à la fin complète du programme de calcul.  
  
 **O**n utilise comme prévu les données sur les villes contenues dans notre tableau à partir de leur position donc on travaille sur des nombres plutôt que sur la structure elle même pour remplir notre matrice. Cela nous a permis de concevoir des fonctions relativement simples car manipulant des réels.

* **int extraire\_min(float \*table\_distance, int \*table\_marque, Ville \*ville, int nombre\_de\_ville)**;

//retourne le numéro de la ville à extraire…

* + ---post: retourne le numéro de ville qui est le minimum dans table\_ distance.

*for(i=0; i<nombre\_de\_ville; i++)*  
*{*

*if(table\_marque[i] != -2)* // si la ville est non visitée

*{*  
*if(table\_distance[i]<=min)*  // si il y a une distance plus petite  
*{*  
*min=table\_distance[i];*//on remplace le min  
*num\_ville\_min=i;* // on enregistre le numéro de la ville la plus proche  
*}*  
*}*  
*}*  
  
 **U**n simple calcul avec des réels moins complexe qu’avec la manipulation de struct.

* **float\* extraire\_update\_table\_marque(float \*table\_distance, int \*table\_marque, int nombre\_de\_ville);**
  + ---pré: avoir déjà utilisé extraire\_min;
  + ---post: modifie  la table marque après extraire\_min

**A**lgorithme avec le même principe que la fonction de recherche du minimum.

* **float \*relacher\_table\_distance(float \*table\_distance, int x, int y, Ville \*ville);**
  + ---pré : mise à jour de table\_père une première fois avant cette fonction.
  + ---post:mise à jour de table\_distance après la fonction “relacher”;

//s’il existe une distance dans la table des successeurs qui soit supérieure a celle de la ville précédente + la distance entre X et ses successeur en question  
  
  
*if(table\_distance[Y]>table\_distance[X]+distance(ville[X], ville[Y]))*  
*{*  
*table\_distance[Y]=table\_distance[X]+distance(ville[X], ville[Y]);   // on modifie la table distance*  
*}*  
*return table\_distance;*  
  
  
 **C**alcul simplifié encore une fois par l’utilisation d’entier et rapide d’exécution car la récupération des données se fait rapidement.

* **int cherche\_chemin\_1(Ville \*ville, FILE \*fp\_new\_html, int \*table\_pere, int No\_de\_depart, int No\_de\_dest, int i);**
  + ---post: fonction récursive qui cherche récursivement les pères en donnant le numéro de la ville de départ et numéro de la ville de destination  (et on écrire sûr le fp\_new\_html)

*int No\_de\_pere=table\_pere[No\_de\_dest];*  
  
*fprintf(fp\_new\_html, "[%d, %.2f, %.2f], \n", i, ville[No\_de\_pere].latitude, ville[No\_de\_pere].longitude);*  
*i++;*  
*while(No\_de\_pere!=No\_de\_depart)*  
 *{*  
*No\_de\_pere=cherche\_chemin\_1(ville, fp\_new\_html, table\_pere, No\_de\_depart, No\_de\_pere, i);*  
 *}*  
*return No\_de\_pere;*  
  
 **U**tilisation du tableau dynamique avec la latitude et la longitude de la structure Ville.

**L**a récursivité est utilisée pour parcourir efficacement le tableau de Ville.

* **void ecrire\_fichier\_chemin(Ville \*ville, FILE \*fp\_origine\_html, FILE \*fp\_new\_html, int \*table\_pere, int No\_de\_depart, int No\_de\_dest);**
  + -- post: fonction qui écrit le fichier qui affiche le plus court chemin (i.e map\_out\_2.html )

**C**ette fonction est écrite sur le même principe que la fonction d’écriture sur un fichier de format HTML vu dans le premier module.  
  
  
  
  
  
  
RESULTAT:  
  
****Plus court chemin de Paris à Marseille avec une limite de vol pour l’avion de 100km

****Plus court chemin de Marseille à Pau avec une limite de vol pour l’avion de 100km

→ Calcul de l’arbre aérien couvrant minimal

Module Prim  
  
  
 **P**our calculer l’arbre aérien couvant minimal nous avons décidé d’utiliser l’algorithme de Prim pour base:  
  
Procédure PRIM  
  
Paramètres locaux : entier s, graphe G  
Paramètres globaux : graphe T  
entier i, m, y  
réel : v  
ensemble : M  
TvectNent : pp  
TvectNReel : d  
  
Début  
  
T ← graphe\_vide  
M ← ensemble\_vide  
Pour i ← 0 jusqu'à N Faire  
  d[i] ←coût(s, i, G)  
  pp[i] ← s  
  M ← Ajouter (i,M)  
Fin pour  
M ← Supprimer (s,M)  
Tant que M <> Ensemble\_vide Faire  
 m ← Choisir (M,d)  
 M ← Supprimer (m,M)  
 z ← pp[m]  
 v ← coût (m,z,G)  
 T ← Ajout arête <m,z> de coût v à T  
Pour i ← 1 jusqu'à d° m dans G Faire  
   y ← i ième\_succ\_de m dans G  
   Si y   
      d[y] ← coût(m,y,G)  
     pp[y] ← m  
   Fin Si  
Fin Pour  
Fin Tant que  
  
Fin algo  
  
  
 **N**ous sommes partit sur un programme ressemblant beaucoup à celui du module de Dijkstra.

**D**e la même manière on alloue des tableaux dynamiques (marque, distance et père) qu’on va initialiser. Puis suite aux calculs du minimum on va les mettre à jour, puis les relâcher. Enfin on réécrit les données récoltées dans un fichier HTML.  
  
 **É**tant donné que le principe est pratiquement le même et que les fonctions sont écrites sur le même principe on ne vous présente que leurs spécifications fonctionnelles:  
  
Dans le module MODULE\_ARBRE  
  
#include"MODULE\_GLOBALE.h"  
#ifndef MODULE\_ARBRE\_H\_INCLUDED  
#define MODULE\_ARBRE\_H\_INCLUDED  
  
//----------Algorithm\_Prim--------------------  
float\* creer\_table\_prim\_distance(Ville \*ville, int nombre\_de\_ville);  
int\* creer\_table\_prim\_marque(int nombre\_de\_ville);  
int\* creer\_table\_prim\_pere(int nombre\_de\_ville);  
int extraire\_prim\_min(float \*table\_distance, int \*table\_marque, int nombre\_de\_ville);  
int\* extraire\_update\_table\_prim\_marque(float \*table\_prim\_distance, int \*table\_prim\_marque, int nombre\_de\_ville);  
int tous\_villes\_extraire\_prim(int \*table\_prim\_marque, int nombre\_de\_ville);  
float\* relacher\_table\_prim\_distance(float \*table\_prim\_distance, int \*table\_prim\_marque, int X, int Y, Ville \*ville);  
int\* relacher\_table\_prim\_pere(float \*table\_prim\_distance, int \*table\_prim\_marque, int \*table\_prim\_pere, int X, int Y, Ville \*ville);  
void ecrire\_fichier\_couvrant\_minimal(FILE \*fp\_origine\_html, FILE \*fp\_new\_html\_3, int \*table\_prim\_pere, int N, Ville \*x\_plus\_grand\_villes);  
  
#endif

RESULTAT:  
  
**A**rbre aérien couvrant minimal (exemple, les 15 plus grandes villes)

→ Création d’une grammaire et utilisation du logiciel ANTLR

Module ANTLR  
  
 **L**es productions de la grammaire que nous avons réalisée et utilisée pour générer un code C via le logiciel ANTLR est en Annexe.

Nous avons rencontré un problème lié à l’exécution du fichier parser.out. Malgré tout nous avons testé notre grammaire via l’outil interpréteur d’ANTLR et en lui donnant en entrée une copie du fichier XML, le module dessine correctement l’arbre syntaxique.

# CONCLUSION

## Nos erreurs ou difficultés

Nous avons implémenté notre code, pour réaliser une fonction de tri de notre tableau de ville, avec un tri Bulle qui est peu performant au niveau de la réalisation. Ceci est principalement du à un manque de temps de notre part. Nous aurions du utilisé un tri plus efficace comme un tri par tas ou un Quick sort.

Nous avons également rencontré des difficultés lors de la réalisation de certaines fonctions du projet. En effet, si l’utilisateur tape un caractère à la place d’un entier le programme boucle jusqu’à interruption de ce dernier. Nous avons réglé certains de ces problèmes mais pas tous.

De plus nous avons éprouvé des difficultés dans la gestion de mémoire de nos programmes. En effet, nous avons rencontré plusieurs soucis de segmentation ou de gestion des malloc. Ils n'entravent néanmoins pas l'exécution de nos programmes.

## Autre solution possible

Notre programme aurait pu être amélioré avec l’utilisation de structures plus complexes telles que des tas, des tas de fibonacci ou des fonctions plus efficaces telles qu’un algorithme de géométrie algorithmique ou de Kruskal à la place de celui de Prim.

# ANNEXE :

Grammaire ANTLR : productions(les tokens sont présents dans le fichier ANTLR des sources)

s :'<'bin'>' '<'bin'>'

'<Document>''<name>'bin'</name>'

styletag '</Document>''</kml>' EOF

;

Styletag: '<Style' bin '>''<IconStyle>''<scale>'coordinate'</scale>''<Icon>''<href>'bin'</href>''</Icon>''</IconStyle>''<LabelStyle>'

'<color>' name '</color>'

'<scale>'coordinate'</scale>''</LabelStyle>'

'</Style>' styletag

| liste

;

liste : '<Placemark>'ville'</Placemark>'liste

|

;

ville :’<name>'nom\_ville=name'</name>'

'<description>''<![CDATA[' '<' bin '>population:' pop=population '<br>'bin '<br>' bin '<a' bin '>' name '</a><' name '>' '</font>' ']]>'

'</description>''<styleUrl>'bin'</styleUrl>'

'<Point>'coordonnees'</Point>'

{printf("\%s,\%d", ($nom\_ville.text->chars),($pop.text->toInt32($pop.text)));}

;

coordonnees returns [coord c]

:'<coordinates>' b=bin '</coordinates>'

{int i=0;int j=0;

while(($b.text->chars[i]) != ',') {

$c.longi[i]=($b.text->chars[i]); i++;

}

$c.longi[i]='\0'; j=i+1;

while(($b.text->chars[j]) != ',') {

$c.lat[j-i+1]=$b.text->chars[j]; j++;

}

$c.lat[j]='\0';

if (($c.longi[0])=='-') ($c.longi[strlen($c.longi)])='W'; else ($c.longi[strlen($c.longi)])='E';

if (($c.lat[0])=='-') ($c.lat[strlen($c.lat)])='S'; else ($c.lat[strlen($c.lat)])='N';

printf("\%s,\%s\n",$c.lat,$c.longi); }

;