**République Tunisienne

Rapport mini projet C  
**Gestion d’itinéraire**

**élaboRé par:Encadré par:**

**Tlili Hcen Hadded Mohamed Ramzi**

**Khachaa Wafa**

Ministère de l’enseignement supérieur et de la recherche scientifique

Université Tunis El Manar

École National d’Ingénieurs de Tunis

*Département génie informatique*

**Année universitaire**

**2016/2017**

**Sommaire**

Introduction

1. Cahier de charge
2. Conception
   1. Version statique
   2. Version dynamique (avec enregistrement)

Conclusion

**Introduction**

Etant une activité quotidienne, le voyage entre les villes pour les agents de poste ou les livreurs d’un produit présente une perte de temps,d’essence, etc. Afin d’économiser ces ressources, on invite les utilisateurs à essayer l’ »Application permettant la gestion d’itinéraires » servant à donner le chemin le plus cours à choisir lors l’accomplissement du travail.

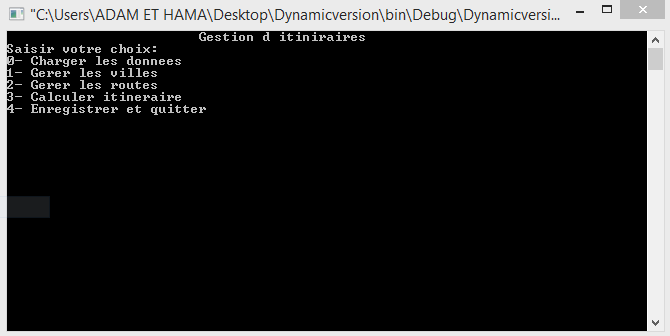
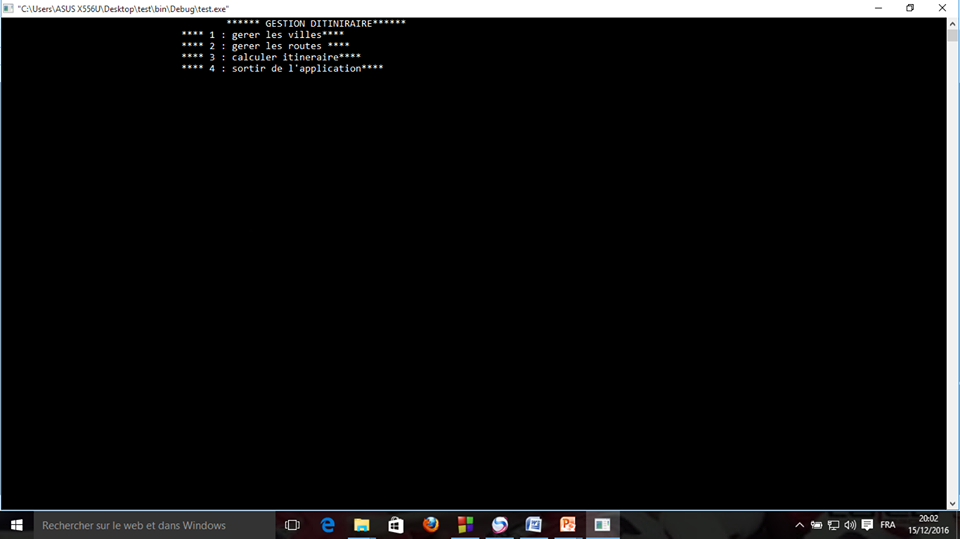
1. **Cahier de charges**

Cette application permet de :

1. **Gérer les villes** 
   1. **Insertion d’une ville**
   2. **Suppression d’une ville**
   3. **Modification d’une ville**
   4. **Affichage des villes**
2. **Gérer les routes** 
   1. **Insertion d’une route**
   2. **Modification d’une distance**
   3. **Suppression d’une route**
   4. **Affichage des routes**
3. **Donner le meilleur itinéraire :**
   1. **Calcul du plus court chemin entre deux villes**
   2. **Calcul du plus courte tournée**
4. **conception**
5. Version statique

**Fonction main() ;**

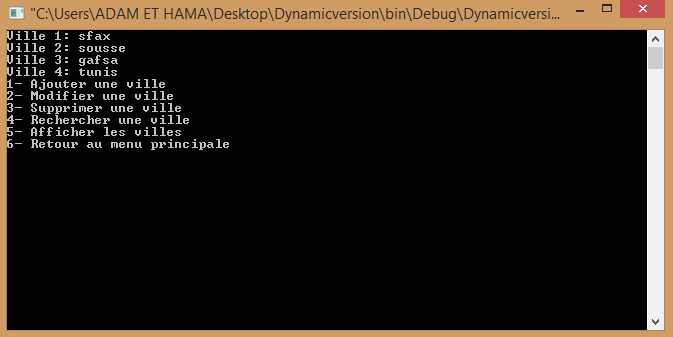
Cette fonction englobe les fonctions principales de l’application , elle contient la déclaration les fonctions de gestion de villes , de routes et de calcul d’itinéraires .



**Fonction void gestion\_des\_villes(int\* nbrvilles,ville tab\_ville[100])**

Elle englobe les fonctions élémentaires de gestion des villes , c'est-à-dire dans cette fonction qu’on fait appel aux fonctions d’ajout , de suppression et d’affichage .. . cette fonction a été basée sur le ‘’ switch’’.

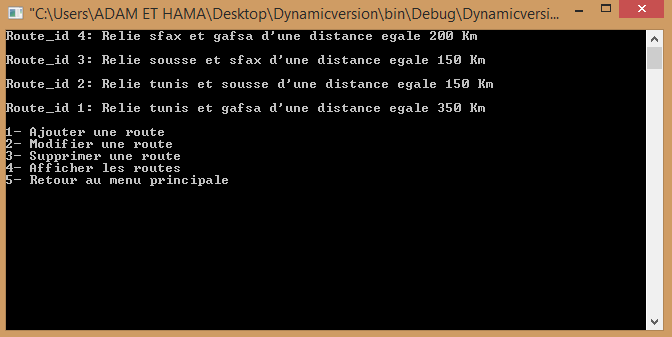
Cette fonction a comme variable le nombre de villes et le tableau des villes.



**Fonction void gestion\_des\_routes(int \*nbrroutes,route \*tab\_route, ville\* tab\_villes,int nbrvilles)**

Elle englobe les fonctions élémentaires de gestion des routes, c'est-à-dire dans cette fonction qu’on fait appel aux fonctions d’ajout , de suppression et d’affichage .. . cette fonction a été basée sur le ‘’ switch’’.

Les variables de cette fonction sont le nombre de routes et le tableaux des routes avec passage par adresse parce qu’ils subissent des changement , et le nombre de villes accompagnés du tableau des villes.



**Fonction void calculs\_itineraires (ville \*tab\_ville,int nbrvilles,route \*tab\_route,int nbrroutes)**

Cette fonction englobe les fonctions calculant l’itinéraires tel que le calcul du mlus court chemin et de la plus courte tournée , ses variables ont un passage par valeur , elle utilise le tableau de villes et des routes et leurs nombres .

**Les fonctions traitant les villes sont :**

**Fonction void ajouter\_ville(ville \*tab\_ville,int\* nbrvilles)**

Celle fonction a pour role d’ajouter une ville en fin du tableau des villes , c’est pourquoi elle a comme variables globaux : le tableau des villes ( passage par adresses) et le nombre des villes.

**Fonction char rechercherville(ville \*tab\_ville,int nbrvilles,char nom[100])**

Cette fonction retourne un message indiquant la position de la ville ,reconnue par son nom, si elle existe dans le tableau des villes . elle a come variables le tableau des villes , le nombre des villes en passage par valeur et une chaine de caracteres à saisir et à rechercher .

**Fonction void afficher\_villes(ville tab\_ville[100],int nbrvilles)**

C’est la fonction qui parcoure le tableau des villes afin d’affiches le nom de chacune , utilisant ainsi le tableau et sa taille nombre des villes .

**Fonction void modifierville (ville \*V, int nbrvilles)**

Cette fonction permet de modifier le nom d’une ville déjà existante dans le tableau des villes . on introduit ainsi un nom , on vérifie qu’il est le nom d’une ville du tableau et on le modifie , en utilisant pour cela le tableau des villes et leurs nombre.

**Fonction int id\_ville(ville \*v,int nbrvilles, char \*nom\_ville)**

Cette fonction est pareille a la fonction de rechercherville . la fonction rechercherville déjà décrites retourne un message contenant la position en cas de presence de ville(-1 di elle n’est pas trouvée) , la fonction id\_ville retourne un entier indiquant la position , en utilisant le tableau de ville , le nombre de celles-ci une chaine nom de la ville rechercher .

**Les fonctions traitant les routes sont :**

**Fonction route saisirroute(ville \*tabVilles,int nbrroutes,int nbrvilles)**

C’est la fonction servant au saisie d’une route , elle retourne une route avec toutes les composantes de la structure route . la saisie est controlée , et ici on utilise le tableau des villes duquel on tire nos villes de départ et d’arrivée , le nombre de routes (utilisé pour gérer l’id de la nouvelle route qui sera ‘nbrroutes’ ) et le nombres de villes déjà saisies .

**Fonction void ajouterroute(route\* tab\_rout,int \*nbrroutes,route r)**

Cette fonction se serve de la fonction précédente de saisie de route pour ajouter la route saisie à la fin du tableau des routes , ce tableau est donc utilisé, le nombre de ses cases ainsi que la nouvelle route saisie .

**Fonction int chercherroute(route \*r,int nbrroutes,route rout1)**

Cette fonction retourne la position d’une route rout1 dans le tableau des routes en parcourant celui-ci et on a -1 si la route n’xiste pas dans le tableau . alors elle utilise le tableau des routes , le nombre des routes et la route recherchée.

**Fonction void modifierroute(route \*tab\_route,int \*nbrroutes)**

C’est la fonction qui modifie la distance associée à une route , elle reconnue la route à modifie à partir de son id saisie , et elle accède à la distance pour modifier , cette modification affecte le tableau des routes utilisé comme variable globale el le nombres de routes servant au test de l’id saisi .

**Fonction void supprimerroute(route tab\_route[10],int \*nbrroutes,route r)**

Cette fonction permet la suppression d’une route du tableau des routes , il y a alors décalage des routes pour ne pas avoir de cases vides et pour avoir un nombre de routes correct . les variables sont le tableau des routes , ses cases remlies(nbrroutes : passage par adresse parcequ’il va changer apres suppression)et la route à supprimer , cette route est déjà saisie dans la fonction globale de gestion des routes avec la fonction saisirroute avant l’appel de la foncton de suppression

**Fonction void afficheroutes(int nbrroutes,ville \*tab\_villes,int nbrvilles,route \*tab\_route)**

Cette fonction parcoure le tableau des routes afin de les afficher avec la commande : printf(" %s -->%s:distance en KM =%d, id=%d\n",a,b,tab\_route[i].distance,tab\_route[i].id);

Cette fonction utilise tableau des routes , nbrroutes , le tableau des villes et leurs nombres .

**Les fonctions servant au calcul de l’itinéraire :**

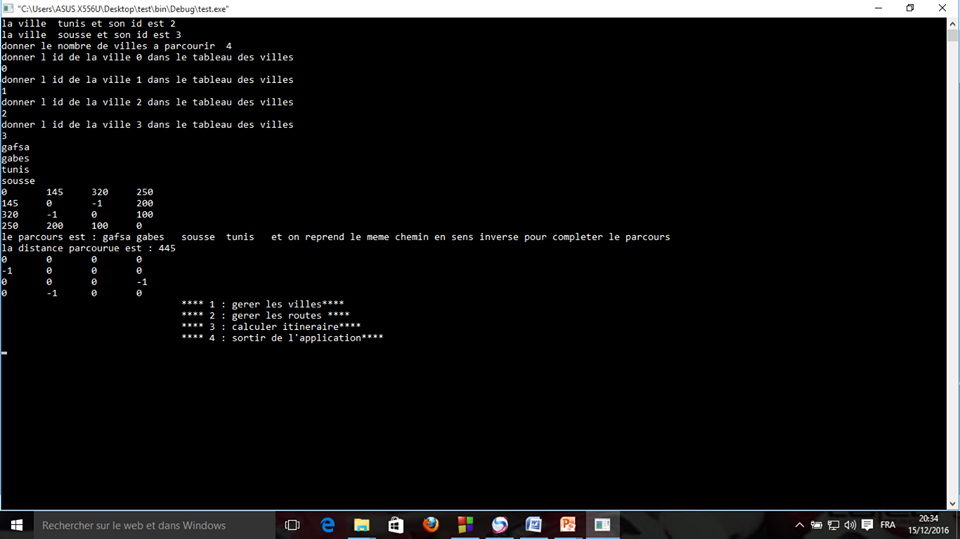
**Fonction int minimum(int tab[100], int a)**

Celle fonction permet la détermination de l’élément du tableau d’entiers qui le plus petit élément strictement positif , cette fonction serve pour la suite dans les fonctions de calcul d’itinéraires .tab est le tableau dans lequel on cherche , et a est la taille de ce tableau à laquelle on limite la recherche.

**Fonction int existance(int a,int tab[100],int b)**

Cette fonction vérifie l’existance de l’entier a dans le tableau tab , en limitant la vérification à la taille b .cette fonction est utilisée pour d’autres fonctions.

**Fonction void leplusprochevoisin (ville tp[100],int nvp,int nbrvilles,ville \*tab\_ville,int nbrroutes,route \*tab\_route**

****

Cette fonction permet le calcul de la plus courte tournée dans un ensemble de villes saisies ,

Dans cette fonction on saisie le nombre de villes à parcourir .

On remplie un tableau par les id des villes du parcours .

On remplie la la matrice des distances , diagonale nulle , est cette matrice est symétrique .

Si deux villes n’ont pas de route entre eux , la case associé aura -1 et sa symétrique aussi aura -1 pour qu’on ne rebrosse pas le chemin .

Le 1er id saisi est celui de la ville de laquelle on souhaite commencer le parcours.

Cette fonction annule la 1ere colonne pour ne plus revenir à la ville de départ, parcoure la 1ere ligne et prend la position du minimum obtenu par la fonction minimum déjà décrite et en utilisant ses conditions . on récupère l’ordonné de ce minimum qui est le plus proche voisin à la ville de depart , cet ordonné est l’indice de la nouvelle ligne étudiée , une colonne dans laquelle est localisé un minimum va etre nulle pour ne pas revenir à une ville déjà parcourue.et ainsi de suite on obtient une tournée sous la règle du plus proche voisin .

Cette fonction utilise comme variable le tableau de parcours , le nombres de villes du parcours , le tableau des villes , celui des routes et leurs nombres nbrvilles et nbrroutes.

**Fonction egalitevect(int a[100],int b[100],int c)**

Cette fonction permet de verifier si deux tableaux d’entiers(a et b) sont égaux à un indice c.

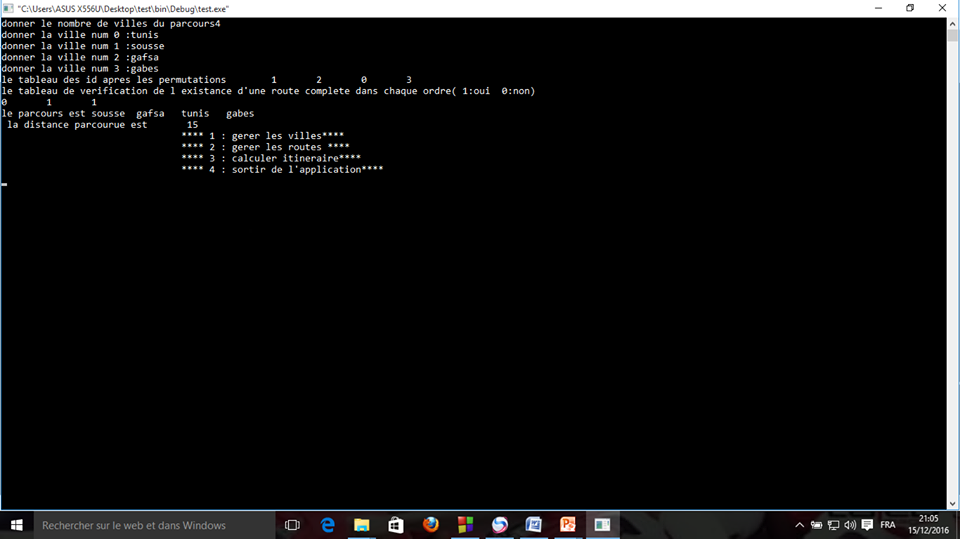
Elle est utilisée dans d’autres fonctions .

**Fonction void saisir\_tab\_parcours(ville \*tab\_ville,int nbrvilles)**

Cette fonction permet de generer tous les ordes possibles d’un ensembles de villes , elle est si complexe que le compilateur prend beaucoup de temps pour generer ces cas en utilisant des fonctions random .

Chaque element saisi dans la matrice contenant tous les cas est saisie par la fonction random associé aux éléments du tableau , et qu’elle se limite sur les valeurs qui n’ont pas été saisie , et sans saisie d’une colonne deux fois (l’utilisation du test fait par la fonction egalvect , tous les choix ont un nombre de factoriel du nombre de villes dans le parcours , le factoriel est calculer par la fonction int fact(int a)

**Fonction void permutaionspossibles(route tab\_route[100],ville \*tab\_ville,int \*nbrroutes,int nbrvilles)**

****

cette fonction a pour role de saisir un ensembles de villes à parcourir dans un tableaux d’entiers contenant les id des villes

ensuite elle effectue des permutations sur des éléments du tableau ,

elle enregistre à chaque fois l’ordre obtenue après permutation dans une colonne d’une matrice

verifier si en suivant une colonne , on a tous les routes dans les différentes villes dans l’ordre donné dans la colonne .

calculer la distance parcourue si on suit l’ordre donné par une colonne dont toutes les routes existes entre chaque de villes voisines dans la colonne

choisir le minimum de ses distances , afficher l’ordre des villes correspondant , et la distance

cette méthode donne une valeur approchée de la distance voulue puisque les permutations sont faites au hasard.

Elle utilise comme variables les tableaux des routes et des villes et leurs nombres , et d’autres tableaux d’entiers et une matrice , cette fonction appelle d’autres fonctions comme ‘minimum’.

**Les fichiers prototypes contiennent les structures qui sont :**

|  |  |
| --- | --- |
| **typedef struct ville {**  **int idville ;**  **char nomville[100];**  **}ville ;** | **typedef struct route{**  **int distance ;**  **int idvilledepart;**  **int idvillearrivee ;**  **int id; }route ;** |

1. Version dynamique (avec enregistrement)

**Fonction main()**

Cette fonction permet d’afficher le menu principal, et elle englobe la fonction de chargement , d’enregistrement , de gestion de villes , celle des routes , et de calcul d’itineraire .

\*\*\*\*\*

**struct route\* chargerroutes(struct route\*L)**

**struct ville\* chargervilles(struct ville\*L)**

ces deux fonctions permettent de charger des routes et des villes déjà saisies et enregistrées et les mettre dans les listes chainées contenant les villes et les routes .

**struct route\* gererroutes(struct route\* L)**

**struct ville\* gerervilles(struct ville\* L)**

ces deux fonctions permettent d’englober toutes les fonctions de gestion des routes et des villes (ajout , suppression , modification , affichage)

**struct route\* ajouterroute(struct route\* L)**

struct ville\* ajouterville(struct ville\*L)

ces deux fonctions permettent d’ajouter une ville ou une route à la tete de chaque liste , elles prennent comme variables les chaines a laquelles on veut ajouter .

**void affichervilles(struct ville\* L)**

**void afficherroute(struct route\* L)**

ces deux fonctions permettent l’affichage des villes et routes déjà saisies

struct route\* modifierroute(struct route\* L)

struct ville\* modifierville(struct ville\*L)

ces deux fonctions permettent la modification d’une route en modifiant la distance ou la modification d’une ville en modifiant le nom .

**void rechercherville(struct ville\* L)**

cette fonction permet la recherche d’une ville dans la chaine des villes en retournant un message permettant d’indique la presence et l’emplacement ou la non existance

**struct ville\* supprimerville(struct ville\* L)**

**struct route\* supprimerroute(struct route\* L)**

ces deux fonction permettent de supprimer une ville ou une route de la chaine associée .

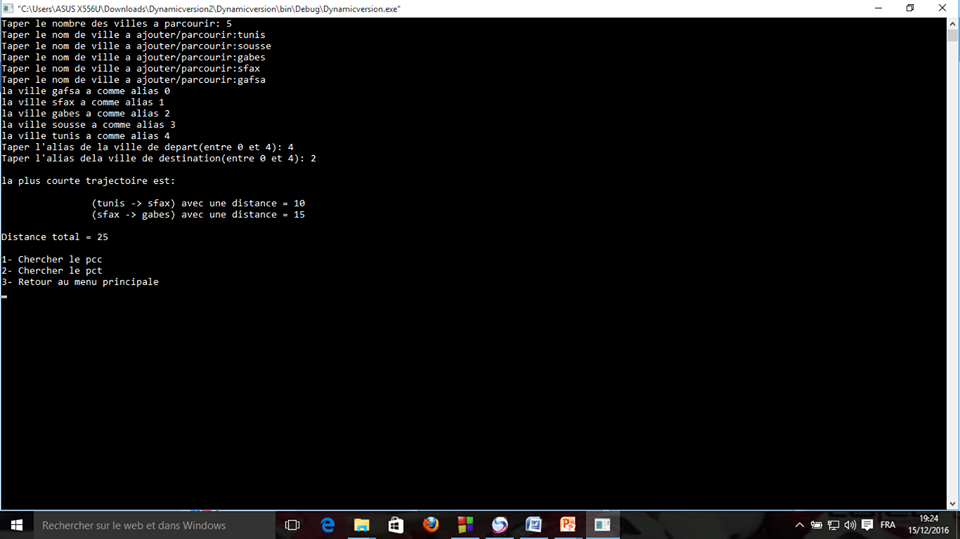
**int verifier(int i,struct route\* L)**

elle permet de verifier si un entier i est egale à un id d’une route dans la liste des routes .

**void enregistrer(struct ville\* L,struct route\* K)**

catte fonction permet d’enregistrer les villes et les routes saisies dans un fichier texte , elle utilise donc les lestes des villes et des routes .

**les fonctions traitant la gestion d’itinéraire par le théoreme dijkstra :**



**int fctdistance(struct route\* L,char a[],char b[])**

permet de retourner la distance entre deux villes si ils existent et si elle ont une route entre eux , les deux villes sont définies ici par leurs noms et on utilise aussi la liste des routes .

**char\* aliasfinder(int i,struct ville\* K)**

permet de déterminer le nom de la ville existante dans la liste des villes et que son id est égal à i .

**void Display\_Result(int s,int d,int final,int precede[],int distance[],int weight[][MAXNODES],struct ville\* K)**

permet d’afficher les résultats , elle est insérée dans la fonction chercherpcourtchemin.

**void chercherpcourtchemin(struct route\* L)**

cette fonction a pour role d’utiliser les fonctions élémentaires afin d’appliquer le fameux théorème dejkstra et de afficher les résultats avec la fonction Display\_Result.

les prototypes de cette version sont :

|  |  |
| --- | --- |
| **struct ville{**  **char nomville[50];**  **int alias;**  **struct ville\* suivant;**  **};** | **struct route{**  **int distance;**  **char villedepart[50];**  **char villearrivee[50];**  **int id;**  **struct route\* suivant;**  **};** |
| La liste des villes à comme contenu dans chaque cellule : le nom de la ville, sont id ; un pointeur de type ville sur la cellule suivante. | La liste des routes à comme contenu dans chaque cellule : le nom de la ville de départ, celui de la ville d’arrivée, la distance, et l’id de la route. |

**Conclusion**

La réalisation de ce projet nous a permis de mieux maîtriser le langage C et d’approfondir nos connaissances sur la théorie des graphes et la théorie de complexité.   
Cette expérience a été également une occasion pour promouvoir notre esprit d’équipe.