RAPPORT DE PROJET

Développement d'une application basée
Sur une base de données

« MY WAY v1.0 »

Par

Adnane Ouahabi Ayaou – Marouan El Hajjaji – Youssef Bourimech

Encadrant : ***Mr. Hassan Badir***

Mai 2013

SOMMAIRE

1. Introduction....3
2. Présentation du projet....3
3. Recherche....4
3.1. Cahier des charges....4
3.1.1. Objectifs....4
3.1.2. Contraintes....4
3.2. Outils....4
3.2.2. Langages de programmation....4
3.2.1. Logiciels....4
4. Développement....5
4.1. Modélisation du graphe....5
4.2. Implémentation de la base de données : MCD, MLD, Schéma relationnel et SQL 6
4.3. Développement de l'application....9
4.3.1. Algorithme du chemin le plus court : Algorithme de Dijkstra....9
4.3.2. Requêtes utilisées....11
4.3.3. Interface graphique....12
4.3.4. Création de l'exécutable....13
5. Conclusion....13

1. Introduction

Afin d'appliquer les méthodologies et les notions enseignées durant notre parcours universitaire, nous devons réaliser un Travail d'application. Celui-ci nous permet à nous, étudiants, de nous initier à la programmation, d'appliquer les connaissances acquises durant notre scolarité et de favoriser le travail en groupe encadré par un enseignant.

Le projet que nous devons réaliser est une application permettant de calculer l'itinéraire le plus court entre deux villes distinctes, en modélisant une carte avec un graphe non orienté, et en se référant a une base de donnée contenant les distances entre deux villes adjacentes.

Afin de comprendre la démarche que nous avons utilisée pour mener ce projet à son terme, notre rapport se structure de la façon suivante :
Tout d'abord, dans une première partie nous présentons le cadre général de notre projet, c'est-à-dire ce qui existe et ce que notre projet va apporter. Puis dans une seconde partie, nous présentons le travail que nous avons effectué, en commençant par définir le cahier des charges. Ensuite dans une troisième partie, nous expliquons comment nous avons choisi les critères de simulations et quels problèmes ont été rencontrés avant que dans une quatrième partie nous décrivions les résultats obtenus. Enfin, dans la dernière partie, nous clôturerons notre rapport avec une conclusion et nous proposerons l'ensemble des extensions qui peuvent être ajoutés à notre projet par la suite.

2. Présentation du projet

Le projet réalisé est une application graphique permettant à l'utilisateur de sélectionner sa ville de départ et sa ville de destination sur la carte et obtenir par la suite l'itinéraire le plus court entre ces deux derniers qui sera tracé par la suite sur la carte par un trait rouge en cliquant sur le bouton 'Générer le chemin' ainsi que la distance en kilomètres. L'utilisateur peut appliquer cette démarche a plusieurs reprise en cliquant sur 'Réinitialiser' après avoir obtenu le résultat souhaité.

3. Recherche

3.1. Cahier des charges

3.1.1. Objectifs du projet

Notre objectif primaire, comme on l'a bien cité dans l'introduction, est d'obtenir le plus court itinéraire entre deux villes distinctes, mais on a aussi visé à offrir à l'utilisateur une interface pratique et simple d'utilisation.

3.1.2. Contraintes

Plusieurs contraintes s'imposent pour la réalisation de ce projet au niveau de la conception mais aussi au niveau du développement :

- Concevoir une base de données solide et optimale.
- Création d'une interface graphique vu qu'on y a jamais travaillé auparavant.
- Exploiter les données de notre base et lancer les requêtes sur notre programme en C++ pour y accéder.
- Créer une application exécutable sur toutes les machines.

3.2. Outils

3.2.2. Langages de programmation.

Le principal langage de programmation dont notre projet se base est le langage SQL qui nous a permit d'exploiter les informations de notre base avec des requêtes.
Notre idée du départ était d'utiliser le langage PHP pour interpréter les requêtes et offrir une interface a l'utilisateur, mais notre choix s'est tourné finalement vers le langage C++, vu qu'on l'a étudié tout le long de ce semestre, c'est ce qui nous a permis d'avoir des connaissances assez suffisante pour développer une application avec ce langage, son avantage aussi c'est qu'il offre une interface graphique sur un programme exécutable, contrairement au PHP qui doit être obligatoirement lancée sur un navigateur web.

3.2.1. Logiciels.

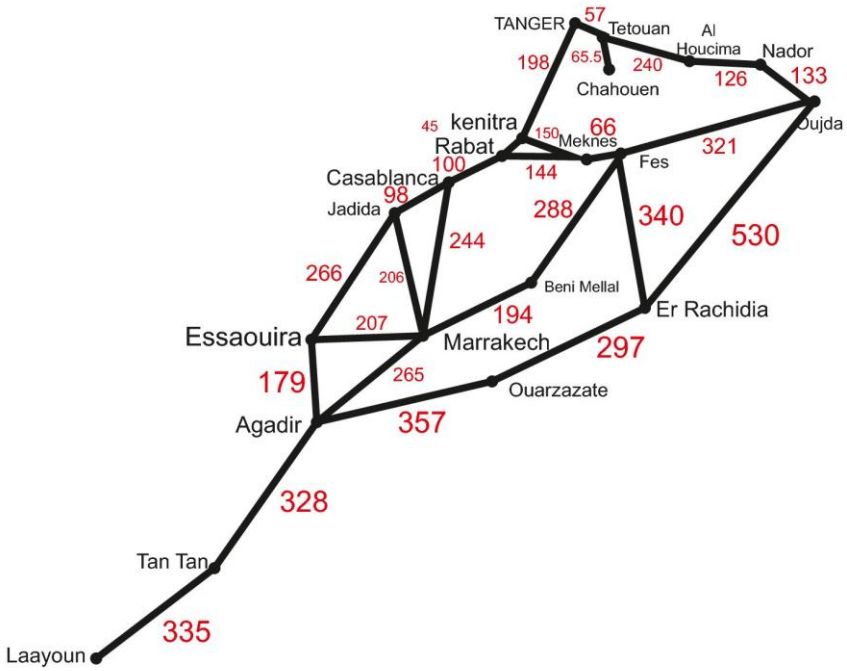
Tout d'abord nous avons besoin d'un SGBD pour stocker notre base de données, notre choix s'est focalisé sur SQLite, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme.

Nous avons besoin ensuite d'un environnement de développement orienté pour la programmation en C++, nous avons choisis Nokia Qt Creator, vu qu'on a pu le pratiquer à plusieurs reprises, aussi simple que pratique, il offre un outil de création d'interfaces graphiques nommé Qt Designer, L'éditeur de texte intégré permet l'autocomplétion ainsi que la coloration syntaxique.

4. Développement

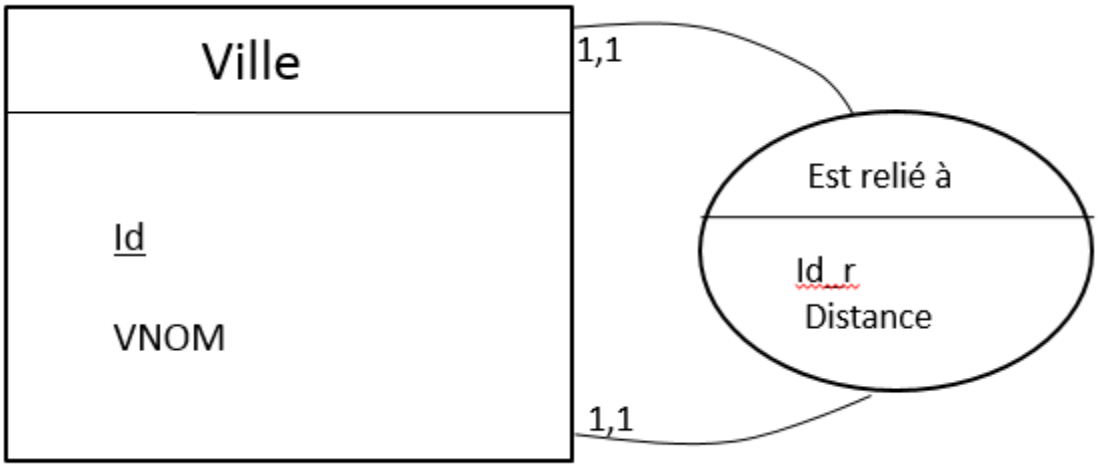
4.1. Modélisation du graphe

Tout d'abord, nous avons choisi de modéliser la carte du Maroc, qui est bien évidemment la mieux connu par nous, on l'a transformé par la suite en un graphe pondéré pour inclure les distances entre les villes adjacentes, et non orienté pour inclure les allers-retours.

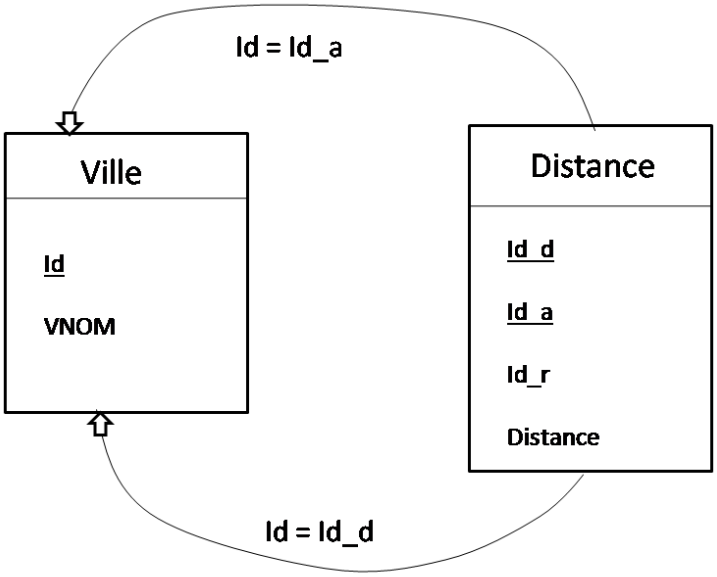


4.2. Implémentation de la base de données : MCD, MLD, Schéma relationnel et SQL

Nous avons besoin par la suite de stocker l'ensemble de ces informations sur une base de données, mais tout d'abord il a fallut qu'on définisse le MCD de notre base.



On peut extraire le MLD suivant :



D'où le schéma relationnel suivant :

Ville (Id, VNOM)

Attribut	Description	Type	Caractéristiques
Id	Numéro d'identification de la ville	Integer	Clé primaire
VNOM	Nom de la ville	Varchar2(10)	

Distance (Id_d, Id_a, Id_r, Distance)

Attribut	Description	Type	Caractéristiques
Id_d	Numéro d'identification de la ville de départ	Integer	Clé primaire
Id_a	Numéro d'identification de la ville d'arrivée	Integer	
Id_r	Numéro d'identification de la route	Integer	
Distance	Distance entre deux villes en Km	Integer	

Ensuite nous avons stocké nos données sur le SGBD **SQLite**.

La table Ville :

```
CREATE TABLE Ville (id number,VNOM varchar2(20), constraint Ville_PK PRIMARY KEY (id));
```

id	vnom
0	Tanger
1	Tetouan
19	Chefchaouen
10	Alhoceima
11	Nador
15	Oujda
2	Kenitra
18	Meknes
12	Fes
3	Rabat
4	Casablanca
5	Eljadida
6	Essaouira
7	Marrakech
13	Beni-mellal
8	Agadir
16	Ouarzazate
14	Errachidia
9	Tantan
17	Laayoune

La table Distance:

```
CREATE TABLE Distance(id_d number, id_a number, id_r number,distance number,
constraint Distance_PK PRIMARY KEY (id_d,id_a));
```

id_r	id_d	id_a	distance
0	0	1	57
1	0	2	198
0	1	0	57
2	1	10	240
3	1	19	65
1	2	0	198
4	2	3	45
5	2	18	150
4	3	2	45
6	3	4	100
7	3	18	145
6	4	3	100
8	4	5	98
9	4	7	244
8	5	4	98
10	5	6	266
11	5	7	206
10	6	5	266
12	6	7	207
13	6	8	179
9	7	4	244
11	7	5	206
12	7	6	207
14	7	8	265
15	7	13	194
16	7	16	198

4.3. Développement de l'application

4.3.1 Algorithme du chemin le plus court : Algorithme de Dijkstra

Nous nous sommes trouvé dans l'obligation d'implémenter l'algorithme qui calcule le chemin le plus court entre deux points du graphe, notre choix s'est orienté finalement vers l'algorithme de Dijkstra en C++.

D'abord on a créé la matrice d'adjacence pour tester l'algorithme:

	TANGER	TETOUAN	KENITRA	RABAT	CASA	JADIDA	ESSAOUIRA	MARRAKECH	AGADIR	TAN TAN	HOUCEIMA	NADOR	FES	BENIMELLAL	ER RACHIDIA	OUJDA	OUARZAZAT	LAAYOUN	MEKNES	CHAOUEN
TANGER	0	57	198	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini
TETOUAN	57	0	infini	infini	infini	infini	infini	infini	infini	infini	240	infini	infini	infini	infini	infini	infini	infini	infini	65
KENITRA	198	infini	0	45	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	150	infini
RABAT	infini	infini	45	0	100	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	144	infini
CASA	infini	infini	infini	100	0	98	infini	244	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini
JADIDA	infini	infini	infini	infini	98	0	266	206	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini
ESSAOUIRA	infini	infini	infini	infini	infini	266	0	207	179	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini
MARRAKECH	infini	infini	infini	infini	infini	206	207	0	265	infini	infini	infini	infini	194	infini	infini	198	infini	infini	infini
AGADIR	infini	infini	infini	infini	infini	infini	179	265	0	328	infini	infini	infini	infini	infini	infini	357	infini	infini	infini
TAN TAN	infini	infini	infini	infini	infini	infini	infini	infini	328	0	infini	infini	infini	infini	infini	infini	infini	335	infini	infini
HOUCEIMA	infini	240	infini	infini	infini	infini	infini	infini	infini	infini	0	126	infini	infini	infini	infini	infini	infini	infini	infini
NADOR	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	126	0	infini	infini	infini	133	infini	infini	infini	infini
FES	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	0	288	340	321	infini	infini	66	infini
BENIMELLAL	infini	infini	infini	infini	infini	infini	infini	194	infini	infini	infini	infini	288	0	infini	infini	infini	infini	infini	infini
ER RACHIDIA	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	340	infini	0	530	297	infini	infini	infini
OUJDA	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	133	321	infini	530	0	infini	infini	infini	infini
OUARZAZAT	infini	infini	infini	infini	infini	infini	infini	198	357	infini	infini	infini	infini	infini	297	infini	0	infini	infini	infini
LAAYOUN	infini	infini	infini	infini	infini	infini	infini	infini	infini	335	infini	infini	infini	infini	infini	infini	infini	0	infini	infini
MEKNES	infini	infini	150	144	infini	infini	infini	infini	infini	infini	infini	infini	66	infini	infini	infini	infini	infini	0	infini
CHAOUEN	infini	65	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	infini	0

L'algorithme de Dijkstra :

```
#ifndef DIJKSTRA_H
#define DIJKSTRA_H
#include <QVector>

class Dijkstra{
    private:
        int** adjMatrix;  // Matrice d adjacence
        int* predecessor, *distance; // Tableau des distances
        bool* mark;  //keep track of visited node
        int source;  //Id de la ville source
        int numOfVertices;  // Nombre des villes
    public:
        Dijkstra(int=0);
        void read();  /* Lire les distances, ainsi que la matrice d'adjacence depuis la base de donnée.
                         Tous les membres de la matrice d'adjacence doivent être positives */
        void initialize();
        /*
            Initialiser tous les membres au debut de l'execution. La distance entre deux sources est zero
            et les distances entre deux points non adjacentes est infinie
        */
        int getClosestUnmarkedNode();
        void calculateDistance();   /* Calculer la distance minimum de la source vers la destination depuis le noeud source
                                       vers les autres noeud */
        void printPath(int,QVector<int> &);  // Enregiste le chemin depuis la source vers la destination dans un vecteur
        int getdistance(int);  // Renvoie la distance parcouru
};
#endif // DIJKSTRA_H
```

```
#include <Dijkstra.h>
#include <QVector>
//#define q2c(string) string.toStdString()

Dijkstra::Dijkstra(int a)
{
    QSqlQuery qry;
    qry.exec("select COUNT(id) from ville");
    qry.next();
    numOfVertices=qry.value(0).toInt();
    predecessor=new int[numOfVertices];
    distance=new int[numOfVertices];
    mark=new bool[numOfVertices];
    adjMatrix=new int*[numOfVertices];
    for(int i=0;i<numOfVertices;i++)
    {
        adjMatrix[i]=new int[numOfVertices];
        for(int j=0;j<numOfVertices;j++)
            adjMatrix[i][j]=9999;
    }
    source=a;
}
void Dijkstra::read(){
    QSqlQuery qry;
    qry.exec("select id_d,id_a,distance from distance");
    while(qry.next())
    {
        adjMatrix[qry.value(0).toInt()][qry.value(1).toInt()]=qry.value(2).toInt();
    }
}
void Dijkstra::initialize(){
    for(int i=0;i<numOfVertices;i++) {
        mark[i] = false;
        predecessor[i] = -1;
        distance[i] = INFINITY;
    }
    distance[source]= 0;
}
```

```
int Dijkstra::getClosestUnmarkedNode(){
    int minDistance = INFINITY;
    int closestUnmarkedNode=0;
    for(int i=0;i<numOfVertices;i++) {
        if((!mark[i]) && ( minDistance >= distance[i])) {
            minDistance = distance[i];
            closestUnmarkedNode = i;
        }
    }
    return closestUnmarkedNode;
}
void Dijkstra::calculateDistance(){
    initialize();
    int closestUnmarkedNode;
    int count = 0;
    while(count < numOfVertices) {
        closestUnmarkedNode = getClosestUnmarkedNode();
        mark[closestUnmarkedNode] = true;
        for(int i=0;i<numOfVertices;i++) {
            if((!mark[i]) && (adjMatrix[closestUnmarkedNode][i]>0) ) {
                if(distance[i] > distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i]) {
                    distance[i] = distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i];
                    predecessor[i] = closestUnmarkedNode;
                }
            }
        }
        count++;
    }
}
void Dijkstra::printPath(int node,QVector<int> &V)
{

    if(node == source)
        V.push_back(node);
    else
    {
        printPath(predecessor[node],V);
        V.push_back(node);
    }
}
int Dijkstra::getdistance(int i)
{
    return  distance[i];
}
```

4.3.2. Requêtes utilisées

```
Qry.exec("select COUNT(id) from ville");
```

Calculer le nombre de villes depuis la table 'Ville'

```
qry.exec("select id_d,id_a,distance from distance");
  while(qry.next())
  {
    adjMatrix[ qry.value(0).toInt()][qry.value(1).toInt() ]=qry.value(2).toInt();
  }
```

Remplir la matrice d adjacence à partir de notre base de données

```
for(int i=0;i<A.size()-1;i++)
  {
    QSqlQuery qry;
    QString B="select id_R from distance where id_d="+QString::number(A[i])+" and
id_a="+QString::number(A[i+1]);
```

```
    qry.exec(B);
    while(qry.next())
    {
       R[qry.value(0).toInt()]->show();
    }
  }
```

Extraire les id des routes du chemin génère à partir du A[ ]
qui contient l'ensemble des nœuds d'où passe le chemin, et afficher les chemins équivalents dans l'interface graphique.

```
qry1.exec("select vnom from ville where id="+QString::number(id_depart));
```

Extraire le nom de ville de départ depuis la table 'Ville'

```
qry2.exec("select vnom from ville where id="+QString::number(id_arrive));
```

Extraire le nom de ville de d'arrivée depuis la table 'Ville'

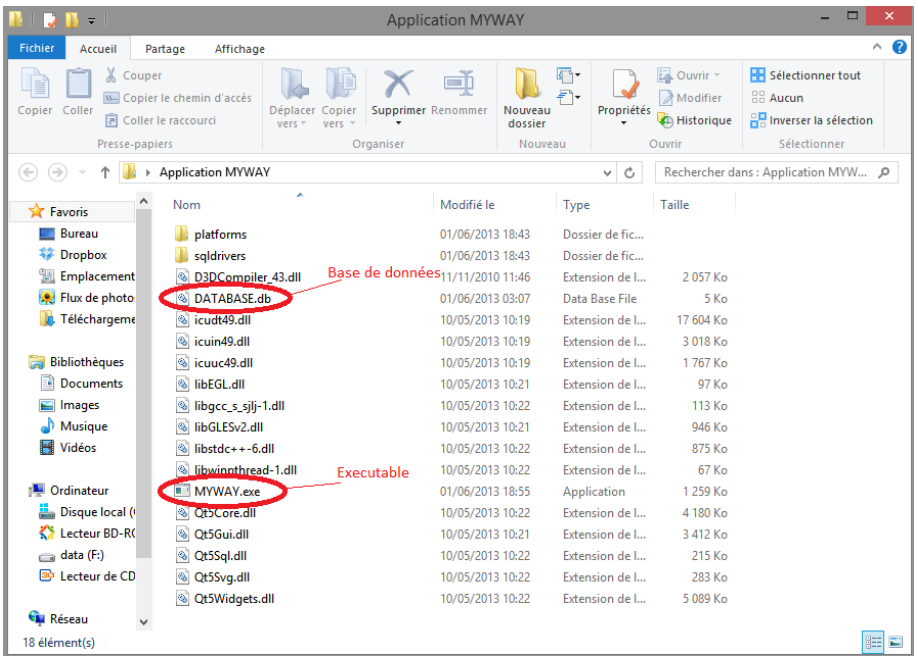
```
L'itinéraire le plus court de "+qry1.value(0).toString()+" à "+qry2.value(0).toString()+" est de
"+QString::number(G.getdistance(id_arrive))+" Km");
```

Afficher le message
« L'itinéraire le plus court de ville départ à 'ville d arrive' est de nombre Km »

4.3.3. Interface graphique

Nous avons créé l'interface graphique de notre application grâce à l'outil Nokia Qt Designer.

4.4.3. Création de l'exécutable



5. Conclusion

Bien évidemment, notre projet peut s'étendre sur plusieurs niveaux tout en offrant beaucoup plus de possibilités et d'options, tout d'abord en incluant d'autres villes sur notre base de données, on pourrait aussi prendre en considération le type de transport (Automobile, Train, Avion) et aussi le type de trajet automobile (Autoroute, national), et en incluant aussi quelques informations sur les routes (Sécurité, embouteillages, Vitesse maximale,..).

L'application peut être développée sous forme d'un guide touristique, en lui offrant davantage d'informations sur les villes, les endroits touristiques qui spécifient chaque ville, l'emplacement des hôtels, des restaurants