INSTITUT SUPÉRIEUR DE GESTION DE TUNIS
Since 1969

Rapport du projet

Remerciements:

Avant d'entamer la présentation de notre expérience de développement, nous souhaitons exprimer notre gratitude envers ceux qui nous ont apporté une aide précieuse tout au long de ce projet. Nous tenons tout particulièrement à remercier Mme Arij Azzabi et Mme Maha Fredj, nos enseignantes de travaux pratiques et travaux dirigés, qui ont été d'une patience et d'une pédagogie exemplaires dans leur accompagnement de notre équipe. Nous sommes reconnaissants pour tout ce que nous avons appris de leur part et nous sommes convaincus que leur expertise et leurs conseils ont grandement contribué à la réussite de notre projet.

Introduction:

Nous sommes heureux de vous présenter aujourd'hui notre projet de programmation en langage C, réalisé par deux étudiants passionnés d'informatique Nader ben Ammar et Halim Kooli .

Notre projet a été développé dans le cadre de notre cursus universitaire et nous sommes ravis de vous en parler aujourd'hui.

Nous avons travaillé en étroite collaboration pour concevoir et développer ce projet, en utilisant des méthodes de développement agile pour assurer la qualité du code et la fiabilité de l'exécution. Nous sommes fiers du résultat final et nous espérons que cette présentation vous donnera un aperçu de notre travail et de notre passion pour la programmation.

Première partie:

En début de l'exécution un ensemble des messages s'affiche en formant une interface de chargement de 0% à 100% en utilisant les '#' comme bar du progrès.

```
                              chargement en cours..
                    ########################################          80%
```

pour ce faire on a eu recours aux bibliothèques "conio.h" et "unistd.h" Ce qui nous a permis de faire l'appel aux fonctions "system("cls"), system ("color xy") et sleep() afin de choisir le couleur de background ainsi que la couleur du texte sur la console, la réalisation des temps d'arrêts durant l'exécution des lignes du code et la remise a zéro de la zone d'affichage du console.

```c
#include <conio.h>
```

```c
#include "unistd.h"
```

```c
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("                                   chargement en cours                                        ");
printf("                         ########                             10%%");
sleep(1);
system("cls");
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("                                   chargement en cours.                                       ");
printf("                         ####################                 30%%");
sleep(1);
system("cls");
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("                                   chargement en cours.                                       ");
printf("                         #######################              35%%");
sleep(1);
system("cls");
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("                                   chargement en cours..                                      ");
printf("                         ########################################          80%%");
sleep(1);
system("cls");
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("                                   chargement en cours...                                     ");
printf("                         #################################################### 100%%");
sleep(1);
system("cls");
```

le programme propose initialement deux choix:

```
                 TAPEZ 1 POUR CONTINUER AVEC LE MAP DE LA TUNISIE
                   TAPEZ 2 POUR CONSTRUIRE VOTRE PROPRE GRAPHE
                                   CHOIX:
```

le premier choix nous mène vers une boucle qui permet de créer un type de données "élément"

plusieurs fois dont chacun désigne une gouvernorat avec une résultat de 24 éléments.

le type de données éléments contient : un nom, une date de création, la population, superficie et le nombre de municipalités.

```
typedef struct{
    char nom[18];
    date date_creation;
    int population;
    float superficie;
    int nombre_municipalites;
}gov,*ELEMENT;
```

ensuite, ces derniers seront reliées entre eux par des arêtes toute en formant un graphe dont les gouvernorats sont les sommets et les arêtes sont les routes.

le graphe entier est réalisé sous la forme d'une liste chainée avec deux structures, la 1er pour les nœuds et la 2eme est pour le nombre de sommets et la liste d'adjacence.

```
typedef struct structNode {
ELEMENT info;
int weight;
struct structNode* next;
} structNode, *Node;
typedef struct {
int V;
Node adjList[26];
} strucGraph, *Graph;
```

après avoir choisis la première alternative une liste des gouvernorats s'affiche avec la possibilité de choisir le point de départ et le point d'arrivée.

```
4: BIZERTE
5: GABES
6: GAFSA
7: JENDOUBA
8: KAIROUAN
9: KASSERINE
10: KEBILI
11: KEF
12: MAHDIA
13: MANOUBA
14: MEDENINE
15: MONASTIR
16: NABEUL
17: SFAX
18: SIDI BOUZID
19: SILIANA
20: SOUSSE
21: TATAOUINE
22: TOZEUR
23: TUNIS
24: ZAGHOUAN
CHOISIR VOTRE POINT DE DEPART : 1

TAPEZ VOTRE POINT D'ARRIVEE : 5

vous allez traverser le chemin suivant :
ARIANA->TUNIS->BEN AROUS->ZAGHOUAN->SOUSSE->MAHDIA->SFAX->GABES
la distance sera de 485 km
```

```
la distance sera de 485 km
TAPEZ 1 POUR CONTINUER
TAPEZ 2 POUR QUITTEZ :
```

l'étape qui suit le choix de position et destination est l'étape principale de ce projet.

il s'agit de la détermination du chemin le plus court possible.

on a donc implémenté une fonction appelée "shortest path" qui prend en paramètre le graphe et le nœud source ainsi que le nœud destination.

La fonction "shortestPath" implémente l'algorithme de Dijkstra pour trouver le plus court chemin dans un graphe pondéré.

La fonction effectue les étapes suivantes :

- **Elle initialise trois tableaux: "dist", "vist" et "pare", qui seront utilisés pour stocker respectivement les distances minimales, les sommets visités et les parents de chaque sommet dans le chemin le plus court.**
- **Elle initialise les tableaux "dist" et "pare" avec des valeurs par défaut (toutes les distances sont initialisées à INT_MAX et tous les parents sont initialisés à -1). Cela signifie que chaque sommet est initialement inaccessible.**
- **Elle initialise la distance de la source à 0.**
- **Elle effectue l'algorithme de Dijkstra en bouclant jusqu'à G->V - 1 fois (le nombre maximum de sommets qu'il peut y avoir dans un chemin le plus court). À chaque itération, elle sélectionne le sommet non visité avec la plus petite distance à la source.**
- **Elle marque le sommet sélectionné comme visité et met à jour les distances des sommets adjacents non visités en vérifiant s'il est possible d'atteindre ces sommets en passant par le sommet sélectionné et en obtenant une distance plus courte.**
- **Elle stocke les parents de chaque sommet dans le chemin le plus court.**
- **Elle appelle la fonction "afficher" pour afficher le chemin le plus court et ses détails.**

```
void shortestPath(Graph G, char source[], char dest[]){
    int dist[G->V],vist[G->V],pare[G->V];
    char ch[25];
    for(int i=1;i<=G->V;i++){
        dist[i]=INT_MAX;
        vist[i]=0;
        pare[i]=-1;
    }
    dist[find_ch(source,G)]=0;
    for (int i = 1; i <=G->V - 1; i++) {
        int u = minimum(dist, vist,G);
        vist[u] = 1;
        for(Node q=G->adjList[u];q!=NULL;q=q->next){
            int v = find_ch(q->info->nom,G);
            if (vist[v] == 0 && dist[u] != INT_MAX &&
                dist[u] + q->weight < dist[v]) {
                dist[v] = dist[u] + q->weight;
                pare[v] = u;
            }
        }
    }
    afficher(pare,source,dest,G->V,G,dist,vist);
}
```

Résultat final :

```
vous allez traverser le chemin suivant :
ARIANA->TUNIS->BEN AROUS->ZAGHOUAN->SOUSSE->MAHDIA->SFAX->GABES
la distance sera de 485 km
```

Deuxième partie:

la deuxième partie consiste à construire un graphe, ensuite le programme offre 3 options:

```
TAPEZ 1 POUR AJOUTER DES ARRET
TAPEZ 2 POUR AFFICHER LE GRAPH
TAPEZ 3 POUR AFFICHER LE PLUS COURT CHEMIN
TAPEZ 0 POUR REVENIR AU MENU PRINCIPALE :
```

1er option : ajouter une arête

l'ajout d'une arête est considérer comme étape complémentaire à la fonction principale de la deuxième partie de ce projet puisqu'elle permet à l'utilisateur d'avoir la structure finale du graphe (sommets + arêtes).

le choix de cette option va engendrer l'exécution du bloc du code suivant :

```
printf("\nVEUILLEZ CHOISIR LE 1er SOMMET : ");
scanf("%i",&choix9);
while(choix9<1 || choix9>G->V)
{
    printf("\nvous avez choisis un sommet qui n'existe pas >:( ! : ");
    scanf("%i",&choix9);
}

printf("\nVEUILLEZ CHOISIR LE 2eme SOMMET : ");
scanf("%i",&choix8);
while(choix8<1 || choix8>24)
{
    printf("\nvous avez choisis un sommet qui n'existe pas >:( ! : ");
    scanf("%i",&choix8);
}
int poids;
printf("\nVEUILLEZ CHOISIR LA LONGEURE DE CETTE ARETE : ");
scanf("%i",&poids);
while(poids<1)
{
    printf("\nla longeure doit etre > a 0 ! : ");
    scanf("%i",&poids);
}
addEdge(G,G->adjList[choix9]->info,G->adjList[choix8]->info,poids);
```

la fonction commence par afficher les noms de tous les sommets du graphe G à l'aide d'une boucle "for". Ensuite, elle demande à l'utilisateur de saisir deux sommets (choix9 et choix8) entre lesquels il souhaite ajouter une arête pondérée. La fonction vérifie que les sommets saisis sont valides

(c'est-à-dire qu'ils sont compris entre 1 et le nombre de sommets du graphe G) en utilisant des boucles "while". Elle demande également à l'utilisateur de saisir le poids de l'arête (poids).

Enfin, la fonction appelle la fonction "addEdge" avec les arguments "G" (le graphe), "G->adjList[choix9]->info" (le premier sommet), "G->adjList[choix8]->info" (le deuxième sommet) et "poids" (le poids de l'arête), pour ajouter l'arête pondérée entre les deux sommets.

La fonction utilise les champs "info" de la structure "Node2" pour accéder aux informations des sommets.

2ème option : afficher le graphe

le choix de cette option fait simplement l'appel à la fonction "printGraph2"

```
{
    printGraph2(G);
}
```

la fonction commence par parcourir tous les sommets du graphe G à l'aide d'une boucle "for". Pour chaque sommet, elle affiche son nom et les noms de tous les sommets voisins en parcourant sa liste d'adjacence à l'aide d'une boucle "for". Pour chaque voisin, la fonction affiche son nom ainsi que le poids de l'arête qui le relie au sommet actuel.

La fonction utilise les champs "info" et "next" de la structure "Node2" pour accéder aux informations du sommet et de ses voisins.

Enfin, la fonction utilise la fonction "getch" pour attendre que l'utilisateur appuie sur une touche avant de terminer l'exécution. Cela permet à l'utilisateur de voir l'affichage du graphe avant que le programme ne se termine.

```
void printGraph2(Graph2 G){
    int i;
    for(i=1;i<=G->V;i++){
        Node2 n;
        printf("\n\t\t%c ",G->adjList[i]->info->nom);printf(": ");
        for(n=G->adjList[i]->next;n!= NULL;n=n->next){
                printf("%c(%i)",n->info->nom,n->weight);
        }
    }
    getch();
}
```

3ème option : (afficher le plus court chemin)

la 3ème est la dernière option pour cette partie, elle s'agit de la détermination du plus court chemin entre deux points, donc le programme va initialement demander ces deux points

(départ / arrivée).

ensuite et en utilisant la fonction ("shortest path") le programme affiche la trajectoire minimale entre les deux points choisis.

```
   char src,dest;
for(int i=1;i<=G->V;i++){
printf("\n %i: %c",i,G->adjList[i]->info->nom);
}
int rep;
printf("\nTAPEZ VOTRE POINT DE DEPART : ");
scanf("%i",&rep);
```

```
while(rep<1 || rep>G->V){
    printf("\nCHOIX INVALIDE !! : ");
    scanf("%i",&rep);
}
src=G->adjList[rep]->info->nom;
printf("\nTAPEZ VOTRE POINT D'ARRIVEE : ");
scanf("%i",&rep);
while(rep<1 || rep>G->V){
    printf("\nCHOIX INVALIDE !! : ");
    scanf("%i",&rep);
}
dest=G->adjList[rep]->info->nom;
shortestPath2(G,src,dest);
```

il commence par la récolte des informations sur le départ et l'arrivée tout en contrôlant la saisie

Elle utilise ensuite la fonction "shortestPath2" pour trouver le chemin le plus court entre ces deux points.

Plus précisément, la fonction commence par afficher les noms de tous les sommets du graphe G. Ensuite, elle demande à l'utilisateur de saisir un point de départ et un point d'arrivée. La fonction vérifie que les points saisis sont valides (c'est-à-dire qu'ils sont compris entre 1 et le nombre de sommets du graphe G) en utilisant une boucle "while". Elle stocke ensuite les noms de ces points dans les variables "src" et "dest".

Enfin, la fonction appelle la fonction "shortestPath2" avec les arguments "G" (le graphe), "src" (le point de départ) et "dest" (le point d'arrivée), pour trouver le chemin le plus court entre ces deux points.