Semester Project

(Data Structures and Algorithms)

Grigore Roberto Cristian

421G

Introduction

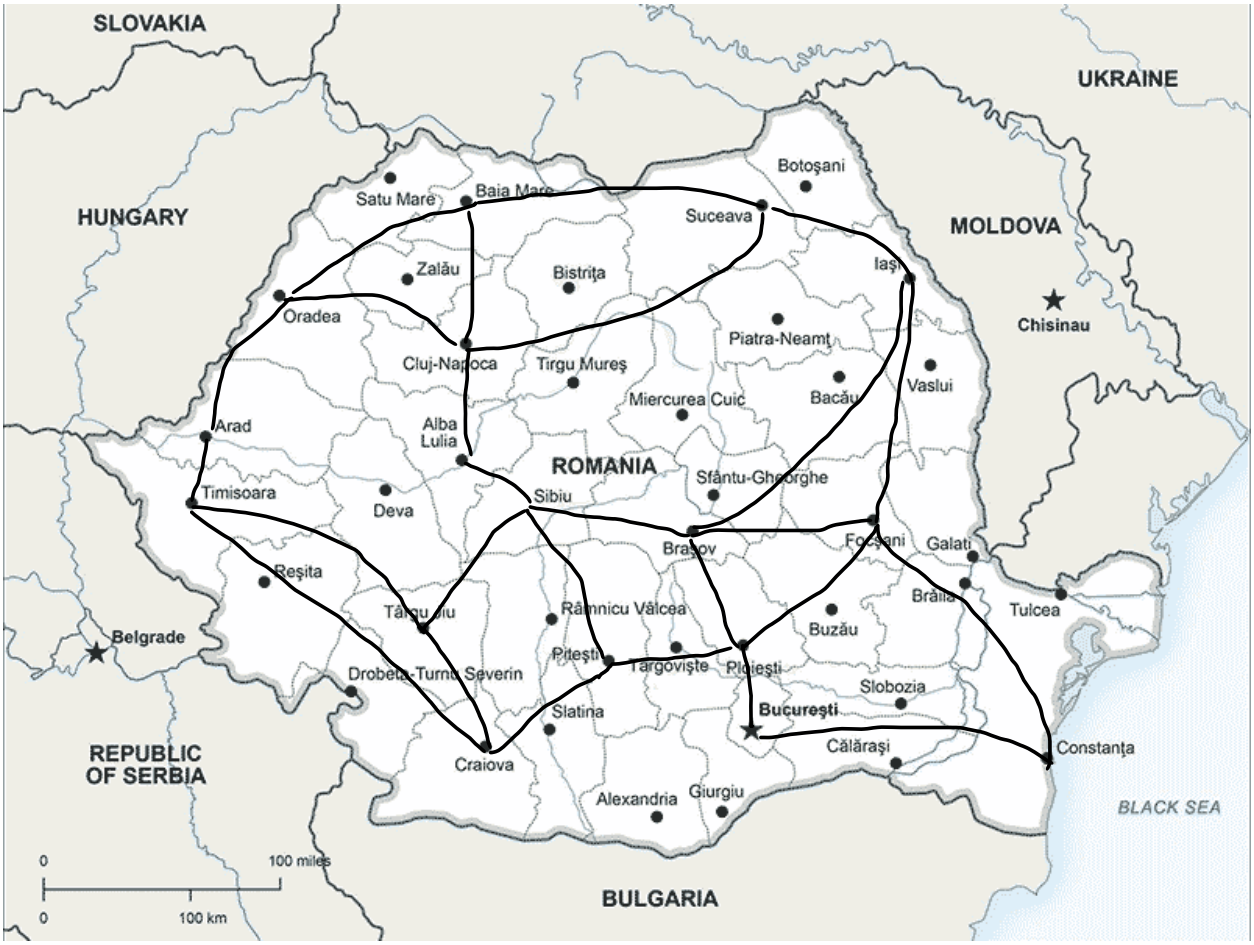
For my semester project at Data Structures and Algorithms I chose to develop a program that can help you find the fastest route between some of the important cities of Romania.

The requirement of the problem:

Write a program that can find the fastest route between two cities. The user will be prompted to select his starting point and the destination. The program will show him the fastest path and the distance that he needs to travel.

Description of the program in natural language

The program will ask for two names and display the required information. The cities represent nodes in a weighted graph, the weight being the distance between two nodes in kilometers. For an easier visualization I realized the following schematic:



```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>
#define numar 100
#define Raza 6371



typedef struct {
    double x;
    double y;
    char *name;
    int number;
} Point;

typedef struct {
    int source;
    int destination;
    double weight;
} Edge;

typedef struct {
    Point nodes[numar ];
    int numNodes;
    double adjacencyMatrix[numar][numar];
} Graph;

double distance(Point* p, Point* q);

Point* addNode(Graph* graph, const char* name, double x, double y, int number) {
    Point* newNode = (Point*)malloc( Size: sizeof(Point));
    newNode->x = x;
    newNode->y = y;
    newNode->number = number;
    newNode->name = strdup( Src: name);
    graph->nodes[number] = *newNode;
    graph->numNodes++;

    return newNode;
}

void addEdge(Graph* graph, Point *p,Point *q) {

    Edge newEdge;
    newEdge.source=p->number;
    newEdge.destination=q->number;
    newEdge.weight=distance(p,q);
    graph->adjacencyMatrix[p->number][q->number] = distance(p,q);
    graph->adjacencyMatrix[q->number][p->number] = distance(p,q);



}

double distance(Point* p, Point* q) {
    double latitude1, longitude1, height1, latitude2, longitude2, height2;

    double x1 = p->y * 0.017;
    double y1 = p->x * 0.017;
    double x2 = q->y * 0.017;
    double y2 = q->x * 0.017;

    height1 = Raza * cos( X: y1);
    latitude1 = Raza * sin( X: y1) * cos( X: x1);
    longitude1 = Raza * sin( X: y1) * sin( X: x1);

    height2 = Raza * cos( X: y2);
    latitude2 = Raza * sin( X: y2) * cos( X: x2);
    longitude2 = Raza * sin( X: y2) * sin( X: x2);

    double distance = (latitude1 - latitude2) * (latitude1 - latitude2) +
                      (longitude1 - longitude2) * (longitude1 - longitude2) +
                      (height1 - height2) * (height1 - height2);

    return sqrt( X: distance);
}
```

```c
void bfs(Graph* graph, Point* start, Point* end) {
    bool visited[numar] = {false};
    int queue[numar];
    int front = 0, rear = 0;
    double distances[numar];
    int previous[numar];  // Keep track of previous nodes for reconstructing the path
    int path[numar];
    int pathIndex = 0;

    for (int i = 0; i < numar; ++i) {
        distances[i] = INFINITY;
        previous[i] = -1;  // Initialize previous nodes to -1
    }

    queue[rear++] = start->number;
    visited[start->number] = true;
    distances[start->number] = 0.0;

    while (front != rear) {
        int current = queue[front++];
        path[pathIndex++] = current;

        if (current == end->number) {
            // Reconstruct the path
            printf( format: "Path: ");
            int node = end->number;
            while (node != -1) {
                printf( format: "%s ", graph->nodes[node].name);
                node = previous[node];
            }
            printf( format: "\nTotal Distance: %.2f km\n", distances[end->number]);
            return;
        }

        for (int i = 0; i < graph->numNodes; ++i) {
            if (!visited[i] && graph->adjacencyMatrix[current][i] != 0) {
                queue[rear++] = i;
                visited[i] = true;
                distances[i] = distances[current] + graph->adjacencyMatrix[current][i];
                previous[i] = current;  // Record the previous node
            }
        }
    }

    printf( format: "No path found.\n");
}



int main() {
    Graph *graph=malloc( Size: sizeof (Graph));
    Point *c1=addNode(graph, name: "Bucuresti", x: 44.26, y: 26.06, number: 0);
    Point *c2=addNode(graph, name: "Ploiesti", x: 44.94,  y: 26.03, number: 1);
    Point *c3= addNode(graph, name: "Brasov", x: 45.65, y: 25.61, number: 2);
    Point *c4= addNode(graph, name: "Constanta", x: 44.18, y: 28.63, number: 3);
    Point *c5= addNode(graph, name: "Cluj",  x: 46.77, y: 23.62, number: 4);
    Point *c6= addNode(graph, name: "Timisoara", x: 45.74, y: 21.20, number: 5);
    Point *c7= addNode(graph, name: "Iasi",  x: 47.15, y: 27.6, number: 6);
    Point *c8= addNode(graph, name: "Craiova",  x: 44.33, y: 23.8, number: 7);
    Point *c9= addNode(graph, name: "Oradea",  x: 47.04, y: 21.91, number: 8);
    Point *c10= addNode(graph, name: "Arad",  x: 46.18, y: 21.32, number: 9);
    Point *c11= addNode(graph, name: "Pitesti",  x: 44.85, y: 24.86, number: 10);
    Point *c12= addNode(graph, name: "Sibiu",  x: 45.79, y: 24.12, number: 11);
    Point *c13= addNode(graph, name: "Suceava",  x: 47.63, y: 26.25, number: 12);
    Point *c14= addNode(graph, name: "Targu Jiu",  x: 45.04, y: 23.27, number: 13);
    Point *c15= addNode(graph, name: "Baia Mare",  x: 47.65, y: 23.57, number: 14);
    Point *c16= addNode(graph, name: "Alba Iulia", x: 46.4, y: 23.34, number: 15);
    Point *c17= addNode(graph, name: "Focsani", x: 45.41, y: 27.11, number: 16);
    addEdge(graph, p: c1, q: c2);
    addEdge(graph, p: c2, q: c3);
    addEdge(graph, p: c4, q: c1);
    addEdge(graph, p: c3, q: c12);
    addEdge(graph, p: c12, q: c14);
    addEdge(graph, p: c8, q: c6);
    addEdge(graph, p: c10, q: c6);
    addEdge(graph, p: c9, q: c10);
    addEdge(graph, p: c15, q: c9);
    addEdge(graph, p: c2, q: c11);
```

```
    addEdge(graph, p: c10, q: c6);
    addEdge(graph, p: c9, q: c10);
    addEdge(graph, p: c15, q: c9);
    addEdge(graph, p: c2, q: c11);
    addEdge(graph, p: c11, q: c12);
    addEdge(graph,  p: c14, q: c8);
    addEdge(graph, p: c5, q: c9);
    addEdge(graph, p: c5, q: c15);
    addEdge(graph, p: c14, q: c6);
    addEdge(graph, p: c5, q: c13);
    addEdge(graph, p: c13, q: c15);
    addEdge(graph, p: c13, q: c7);
    addEdge(graph, p: c7, q: c3);
    addEdge(graph, p: c11, q: c8);
    addEdge(graph, p: c16, q: c5);
    addEdge(graph, p: c16, q: c12);
    addEdge(graph, p: c17, q: c3);
    addEdge(graph, p: c17, q: c4);
    addEdge(graph, p: c17, q: c2);
    addEdge(graph, p: c17, q: c7);
    Point *start=NULL,*end=NULL;
    char ans[10], city1[50], city2[50];

    do {
            printf( format: "Where do you want your journey to start?\n");
            scanf( format: "%s", city1);
            printf( format: "Where do you want your journey to end?\n");
            scanf( format: "%s", city2);
            for (int i = 0; i < graph->numNodes; i++) {
                if (strcmp(graph->nodes[i].name, city1) == 0) start = &graph->nodes[i];
                else if (strcmp(graph->nodes[i].name, city2) == 0) end = &graph->nodes[i];
                if (start != NULL && end != NULL) {
                    break;
                }
            }
        bfs(graph, start, end);
        printf( format: "Would you like to do something else ['no' to stop]? ");
        scanf( format: "%s", ans);
    } while (strcmp(ans, "no") != 0);



    free( Memory: graph);
    return 0;
```

```
Where do you want your journey to start?
Constanta
Where do you want your journey to end?
Iasi
Path: Iasi Focsani Constanta
Total Distance: 367.22 km
```

```
Where do you want your journey to start?
Timisoara
Where do you want your journey to end?
Bucuresti
Path: Bucuresti Ploiesti Pitesti Craiova Timisoara
Total Distance: 506.53 km
```