Authors: Toan Nguyen, Sawyer Sieja, Chase Rhubottom

CS311

Assignment 8

Problem Description: In this assignment, we were tasked with designing and implementing an algorithm to find the shortest path between two cities inputted by the user. The program outputs the shortest route between the two cities, and also displays information about the two cities. If there is no path between the two cities, the program will display this info to the user. If the city input does not exist, the program detects that and exits.

Program Design: For our program, we chose Dijkstra's algorithm, mainly because it is faster than Bellman's algorithm but also because this program can't have negative edges, so we didn't have to worry about that weakness of Dijkstra's algorithm. We added two classes in our program, CityMap and CityInfo. CityMap is the class that actually holds our cities and contains all the functions to display and calculate the shortest routes between them, while CityInfo holds the specific information for each city, for example, the population and elevation of the city. We added these classes because they simplify our program, and make it easier to troubleshoot issues. The CityInfo class also makes the collection and storage of data much easier when reading in from a file, as it includes a simple constructor that can be used to make a new city when reading info.

System Implementation: Our implementation uses the Dijkstra algorithm for the reasons stated above. It reads the two arguments the user passes in, then displays information about those two cities and the route to take between them that equates in the shortest distance. This is all handled by functions written into the classes we made, for both information and the graph itself. One of the major problems we faced was figuring out how to actually implement the Dijkstra algorithm properly. We overcame this issue by writing each step of the Dijkstra algorithm into the code and then optimizing what we had when the code finally worked.

Results: After testing, all of our outputs did match the results in the sample_results file.

- **The shortest distance and path from FI to GG:**

  24 through the route: IRWIN->PARKER->GRPVE

- **The shortest distance and path from PD to PM:**

  133 through the route: PARKER->BOSSTOWN->TORRANCE->POMONA

- **The shortest distance and path from PM to PD:**

  357 through the route:

POMONA->EDWIN->ANAHEIM->VICTORVILLE->CHINO->GRPVE->IRWIN->PARKER

- **The Shortest distance and path from SB to PR:**

  152 through the route: BERNADINO->ISABELLA->BREA->CHINO->RIVERA

Conclusion: Our outputs do match the output of the test program, and while we did run into a few problems while building this program, we were able to overcome them. One issue we got was related to tracking the path of the Dijkstra Algorithm. We thought the code would run fine but it didn't, so a lot of time was wasted to track the code step by step to understand it. Another problem we had was when the code should return no route message but it returned the largest integer possible, but we fixed it quickly. We also had issues with the makefile and some discrepancies between each other's code editors, but through troubleshooting we were able to fix all of those issues in a short time.

Appendix

graph.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <limits.h>
#include "graph.h"

using namespace std;


CityMap::CityMap()
{
    fstream roadFile; //Initializing roadFile as read/write type.
    fstream cityFile; //Intializing cityFile as read/write type.

    string line; //Initializing line as a string to get data from files.

    if(!cityFile.is_open()) { //If the city file is not open...
        cityFile.open("city.txt", ios::in); //Open the city file.
    }

    //Get a line from the city file.
    //Process data from line.
    //Repeat until end of file.
    do{
        CityInfo temp;
        //Initializing integers with negative value for error
detection,(program uses all positive integers).
        cityFile >> temp.numID >> temp.code >> temp.name >> temp.pop >>
temp.elev; //Read from city.txt and store file data accordingly.

        cityList.push_back(temp);
        //This line of code prints each line being read from file. TEST
PURPOSES ONLY.
        //(It works!)
    }while(getline(cityFile, line));

    cityFile.close(); //Close cityFile
    totalCity = static_cast<int>(cityList.size());

    //Assign -1 to every vertex in weight
    for (int i = 0; i < totalCity; i++)
    {
        for (int j = 0; j < totalCity; j++)
        {
            weight[i][j] = -1;
        }
    }

    if(!roadFile.is_open()) { //If the road file is not open...
        roadFile.open("road.txt", ios::in); //Open the road file.

        //Initializing integers with negative value for error
detection,(program uses all positive integers).
        int fromCity = -1;
        int toCity = -1;
        int cityDistance = -1;

        //Get a line from the road file.
        //Process data from line.
        //Repeat until end of file.
        do{
            roadFile >> fromCity >> toCity >> cityDistance; //Read from
road.txt and store file data accordingingly.

            //Save data into weight
            weight[fromCity][toCity] = cityDistance;
        }while(getline(roadFile, line));

        roadFile.close(); //Close roadFile
    }
    else {
        cout << "Error opening/closing file." << endl; //Error catcher in
event can't open or close file properly.
    }

}

int CityMap::CityNumber(string cityCode)
{
    //If the cityCode match a CityInfo in cityList, return city name of
```

```
that CityInfo
    for (int i = 0; i < totalCity; i++)
    {
        if (cityList[i].code == cityCode)
            return cityList[i].numID;
    }
    return -1;
}

void CityMap::FindShortestRoute(string a, string b)
{
    //Use CityNumber function to get city number of a and b
    int cityA = CityNumber(a);
    int cityB = CityNumber(b);

    bool visited[cityList.size()];
    int distance[cityList.size()];
    string path[cityList.size()];

    //Initialize all value of visited[] to false
    //Initialize all value of distance[] to largest integer possible
    //Initialize all value of path[] to empty string
    for (int i = 0; i < totalCity; i++)
    {
        visited[i] = false;
        distance[i] = INT_MAX;
        path[i] = "";
    }

    //Assign 0 to distance of vertex A, making it the starting point
    distance[cityA] = 0;
    while (!VisitedAll(visited))
    {
        //Assign the city number of the closest vertex to u
        int u = FindClosestUnprocessedVertex(distance,visited);

        //Mark u as visited vertex
        visited[u] = true;

        for (int v = 0; v < totalCity; v++)
        {
            //If v vertex is not marked as visited
            //and having an edge connecting from u to v
            //and the distance from u to v is less than or equal the
current distance of v
            if (!visited[v] && weight[u][v] > -1 && distance[v] >=
distance[u] + weight[u][v])
            {
                //Assign the distance from u to v to the distance of v
                distance[v] = distance[u] + weight[u][v];

                //Assign the path of u to path[v]
                path[v] = path[u] + cityList[u].name + "->";
            }
        }
    }

    //Error message if city distance is less than 0
    if (distance[cityB] == INT_MAX)
    {
        cout << "No route from " << cityList[cityA].name << " to " <<
cityList[cityB].name << endl;
    }
    else
    {
        //Print output
        cout << "The Shortest Distance from " << cityList[cityA].name << "
to " << cityList[cityB].name << " is " << distance[cityB] << endl;

        cout << "through the route: ";
        cout << path[cityB] << cityList[cityB].name << endl;
    }
}

bool CityMap::VisitedAll(bool list[])
{
    //Return false if detect a false inside list[]
    for (int i = 0; i < totalCity; i++)
    {
        if (!list[i])
            return false;
    }
    return true;
}

int CityMap::FindClosestUnprocessedVertex(int distance[], bool visited[])
```

```
{
    //Initialize minimun distance to largest integer possible
    int minDist = INT_MAX;
    int closest = -1;

    for (int i = 0; i < totalCity; i++)
    {
        //Assign the city number with the smallest value in distance[] to 
closest
        if (distance[i] <= minDist && !visited[i])
        {
            minDist = distance[i];
            closest = i;
        }
    }
    return closest;
}

bool CityMap::CityExist(string city)
{
    for (int i = 0; i < totalCity; i++)
    {
        if (cityList[i].code == city)
            return true;
    }
    cout << "Invalid City Code: " << city << endl;
    return false;
}

void CityMap::PrintCityInfo(string cityCode)
{
    int num = CityNumber(cityCode);
    cout << "City: " << cityList[num].name
         << ", population " << cityList[num].pop
         << ". elevation " << cityList[num].elev
         << "\n";
}
```

Graph.h

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

using namespace std;

class CityInfo
{
    public:
        //Infos of the city
        int numID;
        string code;
        string name;
        int pop;
        int elev;

        //Class constructor
        CityInfo() {numID = -1; code = ""; name = ""; pop = -1; elev = -1;};
};

class CityMap
{
public:
    int weight[20][20];
    vector<CityInfo> cityList;
    int totalCity;

    //Construct the city info and road map by reading city.txt and road.txt
file
    CityMap();

    //Print informations of a city
    void PrintCityInfo(string cityCode);

    //Using Dijkstra Algorithm to find the shortest route
    void FindShortestRoute(string a, string b);
    int FindClosestUnprocessedVertex(int distance[], bool visited[]);
    bool VisitedAll(bool list[]);

    //Return city number from city code
    int CityNumber(string cityCode);

    //Check if the city exist in data
    bool CityExist(string city);
};
```

Main.cpp

```
/*
* main.cpp
* Authors: Toan Nguyen, Sawyer Sieja, Chase Rhubottom
* CS311
* Assignment 8
* Description:
*   This program will find the shortest route between cities, using the graph
data structures.
*/
#include <iostream>
#include <string>
#include <fstream>
#include "graph.h"

using namespace std;

void printGroupInfo();

int main(int argc, char** argv) {

    //Print the information of the group
    printGroupInfo();

    //Check if the argument 1 and 2 exist
    if (argv[1] == NULL || argv[2] == NULL)
    {
        cout << "Need More Argument!" << endl;
        return -1;
    }

    //Initialize the city map
    CityMap map;

    //End program if city code in argument 1 doesn't exist in city map
    if (!map.CityExist(argv[1]))
        return -1;
    //End program if city code in argument 2 doesn't exist in city map
    if (!map.CityExist(argv[2]))
        return -1;
    cout << "from ";
    map.PrintCityInfo(argv[1]);
    cout << "to ";
    map.PrintCityInfo(argv[2]);
    cout << 
"----------------------------------------------------------------\n";
    map.FindShortestRoute(argv[1], argv[2]);

    return 0;
}

//This function prints our group and assignment information.
void printGroupInfo() {
    cout << "\nAuthors: Toan Nguyen, Sawyer Sieja and Chase Rhubottom" << endl;
    cout << "Date: 11/29/2023" << endl;
    cout << "Course: CS311 (Data Structures and Algorithms)" << endl;
    cout << "Description: This program finds the shortest route between 
cities." << endl;
    cout << "----------------------------------------------------------------" 
<< endl;
}
```