#### Source Code

#### Code [Console]

#### Header files

#include <iostream>

#include <vector> //storing vertices

#include <list> //represents the edge list of each vertex in the graph

#include <queue> //Used in Dijkstra's algorithm for priority queue operations

#include <algorithm> //Used for the reverse function to reverse the path in Dijkstra's algorithm.

#include <string> //represents place names and comments.

#include <climits>

#include <fstream> //file handling

using namespace std;

#### Edge Class

class edge {

public:

int dest\_vertex\_id;

int weight;

edge() {

dest\_vertex\_id = -1;

weight = 0;

}

edge(int dv, int wt) {

dest\_vertex\_id = dv;

weight = wt;

}

void set\_dest\_vertex\_id(int dvi) { dest\_vertex\_id = dvi; }

void set\_weight(int w) { weight = w; }

int get\_dest\_vertex\_id() { return dest\_vertex\_id; }

int get\_weight() { return weight; }

};

#### Vertex Class

class vertex {

public:

int vertex\_id;

string place\_name;

list<edge> edgelist;

vertex() {

vertex\_id = -1;

place\_name = "";

}

void set\_vertex\_id(int v) { vertex\_id = v; }

void set\_place\_name(const string& name) { place\_name = name; }

string get\_place\_name() const { return place\_name; }

int get\_vertex\_id() { return vertex\_id; }

list<edge>& get\_edge\_list() { return edgelist; }

void print\_edge\_list() {

cout << "[" << place\_name << "]: ";

for (auto it = edgelist.begin(); it != edgelist.end(); it++) {

cout << it->get\_dest\_vertex\_id() << "(" << it->get\_weight() << ") --> ";

}

cout << endl;

}

};

#### Graph Class

class graph {

public:

vector<vertex> mygraph;

void add\_vertex(const vertex& v) {

vertex new\_vertex = v; // Create a new vertex and copy the data from the passed vertex

new\_vertex.set\_vertex\_id(mygraph.size()); // Set the vertex ID based on the current size of mygraph

mygraph.push\_back(new\_vertex); // Add the new vertex to the graph

}

void add\_edge(const string& from, const string& to, int w) {

int from\_id = -1, to\_id = -1;

for (size\_t i = 0; i < mygraph.size(); ++i) {

if (mygraph[i].get\_place\_name() == from) {

from\_id = i;

}

if (mygraph[i].get\_place\_name() == to) {

to\_id = i;

}

}

if (from\_id != -1 && to\_id != -1) {

edge e1(to\_id, w);

mygraph[from\_id].edgelist.push\_back(e1);

edge e2(from\_id, w);

mygraph[to\_id].edgelist.push\_back(e2);

}

else {

cout << "Invalid place name(s) for edge: " << from << " - " << to << endl;

}

}

string findNearestHospital(const string& location) {

int location\_id = -1;

// Finding the location index in the graph

for (size\_t i = 0; i < mygraph.size(); ++i) {

if (mygraph[i].get\_place\_name() == location) {

location\_id = i;

break;

}

}

if (location\_id == -1) {

return "Location not found in the graph.";

}

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

vector<int> distTo(mygraph.size(), INT\_MAX);

distTo[location\_id] = 0;

pq.push({ 0, location\_id });

while (!pq.empty()) {

int node = pq.top().second;

int dis = pq.top().first;

pq.pop();

for (auto it : mygraph[node].get\_edge\_list()) {

int v = it.dest\_vertex\_id;

int w = it.weight;

if (dis + w < distTo[v]) {

distTo[v] = dis + w;

pq.push({ dis + w, v });

}

}

}

int minDist = INT\_MAX;

string nearestHospital;

// Finding the nearest hospital from the calculated distances

for (size\_t i = 0; i < mygraph.size(); ++i) {

if (mygraph[i].get\_place\_name().find("Hospital") != string::npos && distTo[i] < minDist) {

minDist = distTo[i];

nearestHospital = mygraph[i].get\_place\_name();

}

}

if (nearestHospital.empty()) {

return "No hospitals found nearby.";

}

return nearestHospital;

}

};

#### Solution Class

class sol {

public:

int checkmain;

sol() : checkmain(0) {}

pair<vector<int>, vector<int>> dijkstra(int V, graph& g, int S, int dv) {

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

// Use a value that is less likely to cause overflow

vector<int> distTo(V, INT\_MAX);

vector<int> parent(V);

for (int i = 0; i < V; i++) {

parent[i] = i;

}

parent[S] = S;

distTo[S] = 0;

pq.push({ 0, S });

while (!pq.empty()) {

int node = pq.top().second;

int dis = pq.top().first;

pq.pop();

for (auto it : g.mygraph[node].get\_edge\_list()) {

int v = it.dest\_vertex\_id;

int w = it.weight;

if (dis + w < distTo[v]) {

distTo[v] = dis + w;

pq.push({ dis + w, v });

parent[v] = node;

}

}

}

// Debug print statements

cout << "Parents: ";

for (int i : parent) {

cout << i << " ";

}

cout << endl;

vector<int> path;

int mynode = dv;

while (parent[mynode] != mynode) {

path.push\_back(mynode); //function using vector headerfile

mynode = parent[mynode];

}

path.push\_back(S); //function using vector headerfile

reverse(path.begin(), path.end()); //reverse function using algorithm headerfile

// Debug print statement

cout << "Path: ";

for (int i : path) {

cout << i << " ";

}

cout << endl;

return { path, parent };

}

};

#### Feedback Class

class Feedback {

public:

int rating;

string comments;

Feedback() {

rating = 0;

comments = "";

}

void setRating(int r)

{

rating = r;

}

void setComments(const string& c)

{

comments = c;

}

int getRating() const

{

return rating;

}

const string& getComments() const

{

return comments;

}

};

void saveFeedback(const Feedback& feedback) {

ofstream file("feedback.txt", ios::app); // Open file in append mode

if (file.is\_open()) {

file << "Rating: " << feedback.getRating() << "\n";

file << "Comments: " << feedback.getComments() << "\n";

file << "------------------------\n"; // Separation for multiple feedback entries

file.close();

cout << "Feedback saved successfully!\n";

}

else {

cout << "Error: Unable to open the feedback file.\n";

}

}

void displayFeedback() {

ifstream file("feedback.txt");

if (file.is\_open()) {

string line;

while (getline(file, line)) {

cout << line << endl;

}

file.close();

}

else {

cout << "Error: Unable to open the feedback file.\n";

}

}

#### Main Function

int main() {

cout << "\t\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "\t\t\t\t| \t\t\t\t\t\t\t\t\t\t\t | \n";

cout << "\t\t\t\t|\t\t\t\t\t WELCOME TO\t\t\t | \n";

cout << "\t\t\t\t| \t\t\t\t\t\t\t\t\t\t\t | \n";

cout << "\t\t\t\t|\t\t\t\t\t'SWIFT ROUTE FINDER'\t\t\t\t | \n";

cout << "\t\t\t\t| \t\t\t\t\t\t\t\t\t\t\t | \n";

cout << "\t\t\t\t|\t\t\t\tBetter Way to Reach Your Destination \t\t\t | \n";

cout << "\t\t\t\t| \t\t\t\t\t\t\t\t\t\t\t | \n";

cout << "\t\t\t\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| \n\n\n\n\n";

graph g1;

vertex v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11;

v1.set\_vertex\_id(0);

v1.set\_place\_name("Pakistan Monument");

v2.set\_vertex\_id(1);

v2.set\_place\_name("Faisal Mosque");

v3.set\_vertex\_id(2);

v3.set\_place\_name("Daman-e-Koh");

v4.set\_vertex\_id(3);

v4.set\_place\_name("Rawal Lake");

v5.set\_vertex\_id(4);

v5.set\_place\_name("Lok Virsa Museum");

v6.set\_vertex\_id(5);

v6.set\_place\_name("Islamabad Zoo");

v7.set\_vertex\_id(6);

v7.set\_place\_name("Centaurus Mall");

v8.set\_vertex\_id(7);

v8.set\_place\_name("Saidpur Village");

v9.set\_vertex\_id(8);

v9.set\_place\_name("Pir Sohawa");

v10.set\_vertex\_id(9);

v10.set\_place\_name("Margalla Hills");

v11.set\_vertex\_id(10);

v11.set\_place\_name("Rose and Jasmine Garden");

g1.add\_vertex(v1);

g1.add\_vertex(v2);

g1.add\_vertex(v3);

g1.add\_vertex(v4);

g1.add\_vertex(v5);

g1.add\_vertex(v6);

g1.add\_vertex(v7);

g1.add\_vertex(v8);

g1.add\_vertex(v9);

g1.add\_vertex(v10);

g1.add\_vertex(v11);

g1.add\_edge("Faisal Mosque", "Pakistan Monument", 5);

g1.add\_edge("Faisal Mosque", "Daman-e-Koh", 3);

g1.add\_edge("Faisal Mosque", "Saidpur Village", 3);

g1.add\_edge("Daman-e-Koh", "Pir Sohawa", 4);

g1.add\_edge("Daman-e-Koh", "Centaurus Mall", 4);

g1.add\_edge("Daman-e-Koh", "Saidpur Village", 4);

g1.add\_edge("Rawal Lake", "Rose and Jasmine Garden", 8);

g1.add\_edge("Islamabad Zoo", "Rose and Jasmine Garden", 3);

g1.add\_edge("Centaurus Mall", "Saidpur Village", 2);

g1.add\_edge("Pakistan Monument", "Margalla Hills", 4);

g1.add\_edge("Saidpur Village", "Rose and Jasmine Garden", 3);

g1.add\_edge("Daman-e-Koh", "Rose and Jasmine Garden", 3);

g1.add\_edge("Pakistan Monument", "Rose and Jasmine Garden", 3);

g1.add\_edge("Pakistan Monument", "Centaurus Mall", 3);

sol obj;

int V = g1.mygraph.size() + 12;

string start, end;

// Display available places

cout << "\n======================================================================================================================================================================== ";

cout << "\t\t\t\t\t\t\t\t\tAvailable Places\n";

cout << "======================================================================================================================================================================== ";

cout << endl;

for (const vertex& v : g1.mygraph) {

cout << v.get\_place\_name() << endl;

}

cout << "\n======================================================================================================================================================================== ";

cout << "\t\t\t\t\t\t\t\t Obtaining Shortest Path" << endl;

cout << "======================================================================================================================================================================== ";

cout << "Enter the starting place name: ";

getline(cin, start);

cout << "Enter the ending place name: ";

getline(cin, end);

int start\_index = -1, end\_index = -1;

// Find indices of start and end places

for (size\_t i = 0; i < g1.mygraph.size(); ++i) {

if (g1.mygraph[i].get\_place\_name() == start) {

start\_index = i;

}

if (g1.mygraph[i].get\_place\_name() == end) {

end\_index = i;

}

// Break out of the loop if both indices are found

if (start\_index != -1 && end\_index != -1) {

break;

}

}

if (start\_index != -1 && end\_index != -1) {

pair<vector<int>, vector<int>> result = obj.dijkstra(V, g1, start\_index, end\_index);

vector<int> shortestPath = result.first;

vector<int> parent = result.second;

cout << "Size of shortestPath vector: " << result.first.size() << endl;

cout << "Size of parent vector: " << result.second.size() << endl;

// Display the shortest path

cout << "Shortest Path from " << start << " to " << end << ":\n";

for (int node : shortestPath) {

cout << " -> " << g1.mygraph[node].get\_place\_name();

}

cout << endl;

// Display the path user will take (from end to start)

cout << "Path to return back from " << end << " to " << start << ":\n";

int currentNode = end\_index;

while (currentNode != start\_index) {

cout << g1.mygraph[currentNode].get\_place\_name() << " -> ";

currentNode = parent[currentNode];

}

cout << g1.mygraph[currentNode].get\_place\_name() << endl;

}

else {

cout << "Invalid place name(s) entered.\n";

}

cout << "\n======================================================================================================================================================================== ";

cout << "\t\t\t\t\t\t\t\t\tEmergancy Case" << endl;

cout << "======================================================================================================================================================================== ";

//HOSPITAL FEATURE

vertex hospital1, hospital2, hospital3, hospital4, hospital5;

hospital1.set\_vertex\_id(12);

hospital1.set\_place\_name("Maroof International Hospital");

hospital2.set\_vertex\_id(13);

hospital2.set\_place\_name("Max Health Hospital");

hospital3.set\_vertex\_id(14);

hospital3.set\_place\_name("HBS General Hospital");

hospital4.set\_vertex\_id(15);

hospital4.set\_place\_name("Hanif Medical Complex");

hospital5.set\_vertex\_id(16);

hospital5.set\_place\_name("PAF Hospital Unit 2");

g1.add\_vertex(hospital1);

g1.add\_vertex(hospital2);

g1.add\_vertex(hospital3);

g1.add\_vertex(hospital4);

g1.add\_vertex(hospital5);

// Connect hospital vertices with edges

g1.add\_edge("Maroof International Hospital", "Faisal Mosque", 8);

g1.add\_edge("Daman-e-Koh", "Pir Sohawa", 8);

g1.add\_edge("Centaurus Mall", "Islamabad Zoo", 8);

g1.add\_edge("Pakistan Monument", "Rose and Jasmine Garden", 8);

g1.add\_edge("Max Health Hospital", "Pakistan Monument", 9);

g1.add\_edge("Centaurus Mall", "Max Health Hospital", 8);

g1.add\_edge("Pir Sohawa", "Centaurus Mall", 8);

g1.add\_edge("HBS General Hospital", "Rawal Lake", 10);

g1.add\_edge("Pakistan Monument", "Lok Virsa Museum", 8);

g1.add\_edge("Hanif Medical Complex", "Saidpur Village", 8);

g1.add\_edge("Max Health Hospital", "Daman-e-Koh", 8);

g1.add\_edge("PAF Hospital Unit 2", "Margalla Hills", 8);

string emergencyOption;

cout << "Do you have an emergency? (yes/no): ";

cin >> emergencyOption;

if (emergencyOption == "yes") {

string userLocation;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Enter your current location: ";

getline(cin, userLocation);

// Find the nearest hospital

string nearestHospital = g1.findNearestHospital(userLocation);

cout << "The nearest hospital from " << userLocation << " is: " << nearestHospital << endl;

// Find the shortest path from user's location to the nearest hospital

int userLocationIndex = -1;

int nearestHospitalIndex = -1;

// Find the index of the user's location and the nearest hospital in the graph

for (size\_t i = 0; i < g1.mygraph.size(); ++i) {

if (g1.mygraph[i].get\_place\_name() == userLocation) {

userLocationIndex = i;

}

if (g1.mygraph[i].get\_place\_name() == nearestHospital) {

nearestHospitalIndex = i;

}

// Break out of the loop if both indices are found

if (userLocationIndex != -1 && nearestHospitalIndex != -1) {

break;

}

}

if (userLocationIndex != -1 && nearestHospitalIndex != -1) {

pair<vector<int>, vector<int>> result = obj.dijkstra(V, g1, userLocationIndex, nearestHospitalIndex);

vector<int> shortestPath = result.first;

vector<int> parent = result.second;

// Display the shortest path

cout << "Shortest Path from " << userLocation << " to " << nearestHospital << ":\n";

for (int node : shortestPath) {

cout << " -> " << g1.mygraph[node].get\_place\_name();

}

cout << endl;

// Display the path from user's location to the nearest hospital

cout << "Path to return back from " << nearestHospital << " to " << userLocation << ":\n";

int currentNode = nearestHospitalIndex;

while (currentNode != userLocationIndex) {

cout << g1.mygraph[currentNode].get\_place\_name() << " -> ";

currentNode = parent[currentNode];

}

cout << g1.mygraph[currentNode].get\_place\_name() << endl;

}

else {

cout << "User location or nearest hospital not found in the graph.\n";

}

}

cout << "\n======================================================================================================================================================================== ";

cout << endl;

Feedback userFeedback;

cout << "Please provide feedback on your experience:" << endl;

// Get rating

cout << "Rating (1 to 5, 5 being the best): ";

cin >> userFeedback.rating;

// Clear input buffer

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

// Get comments

cout << "Comments (press Enter to skip): ";

getline(cin, userFeedback.comments);

// Display feedback

cout << "\nThank you for your feedback!\n";

cout << "Rating: " << userFeedback.rating << "\n";

cout << "Comments: " << userFeedback.comments << "\n";

// Save feedback to file

saveFeedback(userFeedback);

// Display all feedback

cout << "\nAll Feedback:\n";

displayFeedback();

return 0;

}