**Final Project: Implementing Shortest Algorithm**

CSEN 2328: Data Structures & Algorithms

Texas A&M University- Kingsville

Spring 2019

Dr. Khan

By:

Diego Reyes

**Project description:**

The main objective of our final project was to implement Dijkstra’s shortest path algorithm on the given Navy ship passageway network. Given a source node the algorithm finds the shortest path distance to each exit and which exit is the closest. The algorithm was implemented using the C++ programming language. The given passageway network was comprised of 23 nodes, and 28 edges which were weighted.

**Given project tasks:**

1. Store the adjacency list of the passageway network using a two-dimensional array

2. Implement shortest path algorithm using C++.

3. Run the shortest path algorithm on the passageway network used in Homework 2 and 3.

4. Find the shortest path distances from a given node to all exit nodes and find the nearest exit.

**Given passageway network:**

A close up of a map

Description automatically generated

**Computed distances:**

The distance of the source vertex P from the exits are:

Exit A: 36

Exit F: 54

Exit T: 38   
The closest exit from the source vertex is: A

**Explanation of results:**

The closest exit from vertex P is exit A. The distance is computed by the algorithm which searches the graph for the shortest distance from one vertex to the next.

**Source code:**

**#include<bits/stdc++.h>**

**#include <stdlib.h>**

**using namespace std;**

**#define INF INT\_MAX**

**int G[23][3] = { // Graph array**

**{1,8},**

**{0,22},**

**{3,7,22},**

**{2,15},**

**{6},**

**{6,9},**

**{4,5,7},**

**{2,6,8},**

**{0,7,9},**

**{5,8,10},**

**{9,11,20},**

**{10,12,18},**

**{11,13,19},**

**{12,14,17},**

**{13,15,16},**

**{3,14},**

**{14,17},**

**{13,16,18},**

**{11,17},**

**{12,20},**

**{10,19},**

**{22},**

**{1,2,21}**

**};**

**int W[23][3] = { // Weight array**

**{4,4},**

**{4,4},**

**{8,10,8},**

**{8,12},**

**{8},**

**{16,12},**

**{8,16,8},**

**{10,8,12},**

**{4,12,30},**

**{12,30,12},**

**{12,20,20},**

**{20,8,28},**

**{8,14,12},**

**{14,4,14},**

**{4,8,14},**

**{12,8},**

**{14,4},**

**{14,4,8},**

**{28,8},**

**{12,20},**

**{20,20},**

**{14},**

**{4,8,14}**

**};**

**int Deg[23] = {2,2,3,2,1,2,3,3,3,3,3,3,3,3,3,2,2,3,2,2,2,1,3}; // Degree array**

**int d[23]; // Distance array**

**typedef pair<int, int> node;**

**void ShortestPath(int [][3],char schar) {**

**int s = -1;**

**// Character to Integer Conversion**

**if (schar == 'A') {**

**s = 0;**

**}else if (schar == 'B'){**

**s = 1;**

**}else if (schar == 'C'){**

**s = 2;**

**}else if (schar == 'D'){**

**s = 3;**

**}else if (schar == 'E'){**

**s = 4;**

**}else if (schar == 'F'){**

**s = 5;**

**}else if (schar == 'G'){**

**s = 6;**

**}else if (schar == 'H'){**

**s = 7;**

**}else if (schar == 'I'){**

**s = 8;**

**}else if (schar == 'J'){**

**s = 9;**

**}else if (schar == 'K'){**

**s = 10;**

**}else if (schar == 'L'){**

**s = 11;**

**}else if (schar == 'M'){**

**s = 12;**

**}else if (schar == 'N'){**

**s = 13;**

**}else if (schar == 'O'){**

**s = 14;**

**}else if (schar == 'P'){**

**s = 15;**

**}else if (schar == 'Q'){**

**s = 16;**

**}else if (schar == 'R'){**

**s = 17;**

**}else if (schar == 'S'){**

**s = 18;**

**}else if (schar == 'T'){**

**s = 19;**

**}else if (schar == 'U'){**

**s = 20;**

**}else if (schar == 'V'){**

**s = 21;**

**}else if (schar == 'W'){**

**s = 22;**

**}else{**

**s = -1;**

**}**

**if (s != -1){ // If s does not equal -1 (input is valid) do**

**priority\_queue<node, vector<node>, greater<node> > Q; // Queue gives priority to the node witht he smallest distance from the source node**

**for (int v = 0; v < 23; v++) { // Initialize the distance of the nodes to infinity**

**d[v] = INF;**

**}**

**d[s] = 0; // Initialize the distance of the source node to 0**

**Q.push(make\_pair(s,d[s])); // Enqueue s and d[s]**

**while (!Q.empty()) {**

**int u = Q.top().first; // Set u to the node with the distance closest to the source**

**Q.pop(); // Dequeue that node**

**for (int a = 0; a < Deg[u]; a++) { // For each node adjacent to u do**

**int v = G[u][a]; // Adjacent node to u**

**int weight = W[u][a]; // Weight of the edge between node u and node v**

**if (d[v] > d[u] + weight) { // If the distance of node v is greater than the distance of node u + the weight of the edge between the nodes do**

**d[v] = d[u] + weight; // Set the distance of node v to the distance of node u + the weight of the edge between the nodes**

**Q.push(make\_pair(v,d[v])); // Enqueue the node**

**}**

**}**

**}**

**char min;**

**if (d[0] > d[5] and d[5] > d[19]){ // Compare the distance of the source node to the exits and sets the node with the least distance to min**

**min = 'T';**

**}else if (d[19] > d[0] and d[0] > d[5]){**

**min = 'F';**

**}else{**

**min = 'A';**

**}**

**cout << "The distance of the source vertex " << schar << " from the exits are: " << endl;**

**cout << "Exit A: " << d[0] << endl << "Exit F: " << d[5] << endl << "Exit T: " << d[19] << endl;**

**cout << "The closest exit from the source vertex is: " << min << endl << endl;**

**}else{ // If the input is invalid do**

**cout << "\*\*\*SOURCE VERTEX " << schar <<" DOESN'T EXIST IN THE GIVEN NETWORK\*\*\*" << endl << endl;**

**}**

**}**

**int main() {**

**ShortestPath(G,'A');**

**ShortestPath(G, 'P');**

**ShortestPath(G, 'M');**

**ShortestPath(G, 'G');**

**ShortestPath(G, 'Z');**

**return 0;**

**}**

**Output Example:**

**The distance of the source vertex P from the exits are:**

**Exit A: 36**

**Exit F: 54**

**Exit T: 38**

**The closest exit from the source vertex is: A**