Name: ANYA AGARWAL

Section: B (13)

Roll no: 2021122

I. After end term examination, Akshay wants to party with his friends. All his friends

are living as paying guest and it has been decided to first gather at Akshay’s house

and then move towards party location. The problem is that no one knows the exact

address of his house in the city. Akshay as a computer science wizard knows how to

apply his theory subjects in his real life and came up with an amazing idea to help

his friends. He draws a graph by looking in to location of his house and his friends’

location (as a node in the graph) on a map. He wishes to find out shortest distance

and path covering that distance from each of his friend’s location to his house and

then whatsapp them this path so that they can reach his house in minimum time.

Akshay has developed the program that implements Dijkstra’s algorithm but not

sure about correctness of results. Can you also implement the same algorithm and

verify the correctness of Akshay’s results? (Hint: Print shortest path and distance

from friends’ location to Akshay’s house)

#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include <unordered\_map>

#include<bits/stdc++.h>

using namespace std;

vector <int> dijkstra(int V,vector<vector<int>> G, int src)

{

unordered\_map<int, list<pair<int,int>>> adj;

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

{

for(int j=0;j<V;j++)

{

if(G[i][j]!=0)

{

adj[i].push\_back({j,G[i][j]});

adj[j].push\_back({i,G[i][j]});

}

}

}

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

dist[src] = 0;

set<pair<int, int>> s;

s.insert({0, src});

while (!s.empty()) {

auto top = \*(s.begin());

int distance = top.first;

int node = top.second;

s.erase(s.begin());

for (auto i : adj[node]) {

if (distance + i.second < dist[i.first]) {

auto record = s.find({dist[i.first], i.first});

if (record != s.end()) {

s.erase(record);

}

dist[i.first] = distance + i.second;

s.insert({dist[i.first], i.first});

}

}

}

return dist;

}

int main(void) {

ifstream inputFile("input.txt");

ofstream outputFile("outputs.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--) {

int n;

inputFile >> n;

// // graph g;

vector<vector<int>> G(n,vector<int>(n,0));

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

inputFile >> G[i][j];

}

}

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

vector<int> ans=dijkstra(n,G,0);

for(int i=0;i<ans.size();i++)

{

outputFile<<ans[i]<<" ";

}

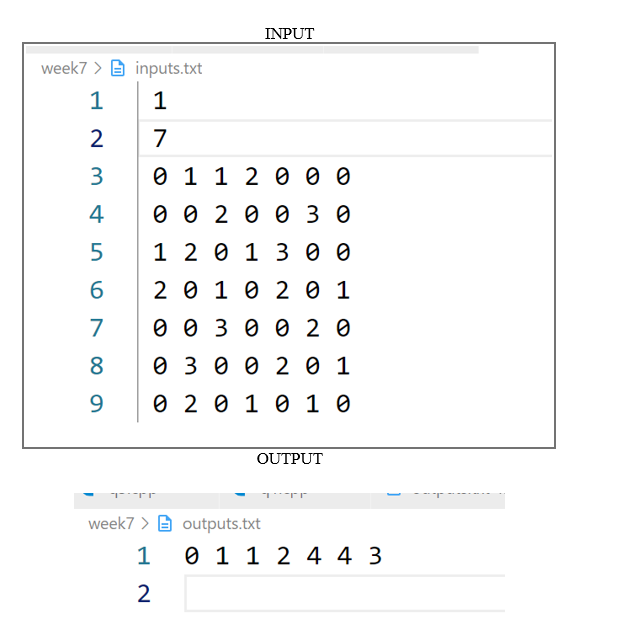
outputFile<<endl;

}

outputFile.close();

return 0;

}



II. Design an algorithm and implement it using a program to solve previous question's

problem using Bellman- Ford's shortest path algorithm.

Input Format:

Input will be the graph in the form of adjacency matrix or

adjacency list. Source vertex number is also provided as an

input.

Output Format:

Output will contain V lines.

Each line will represent the whole path from destination vertex number to source

vertex number along with minimum path weigth.

#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include <unordered\_map>

#include<bits/stdc++.h>

using namespace std;

vector<int> bellman\_ford(int V, vector<vector<int>>& edges, int S) {

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

dist[S] = 0;

for (int i = 0; i < V - 1; i++) {

for (auto it : edges) {

int u = it[0];

int v = it[1];

int wt = it[2];

if (dist[u] != INT\_MAX && dist[u] + wt < dist[v]) {

dist[v] = dist[u] + wt;

}

}

}

// Nth relaxation to check negative cycle

for (auto it : edges) {

int u = it[0];

int v = it[1];

int wt = it[2];

if (dist[u] != INT\_MAX && dist[u] + wt < dist[v]) {

return { -1};

}

}

return dist;

}

int main(void) {

ifstream inputFile("inputs.txt");

ofstream outputFile("outputs.txt");

if (!inputFile) {

cerr << "Error opening input file." << endl;

return 1;

}

int T;

inputFile >> T;

while (T--) {

int V;

inputFile >> V;

// // graph g;

vector<vector<int>> edges(V+1,vector<int>(3,0));

for(int i=0;i<V+1;i++)

{

for(int j=0;j<3;j++)

{

inputFile >> edges[i][j];

}

}

if (!outputFile) {

cerr << "Error opening output file." << endl;

return 1;

}

vector<int> ans =bellman\_ford(V ,edges,0);

for(int i=0;i<ans.size();i++)

{

outputFile<<ans[i]<<" ";

}

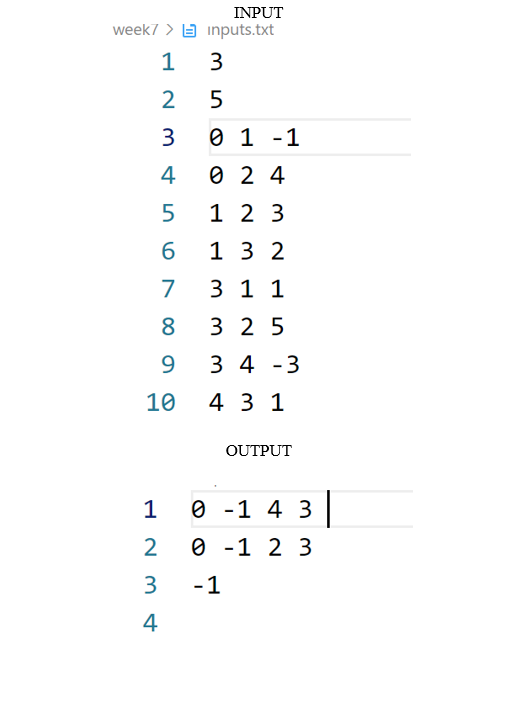
outputFile<<endl;

}

outputFile.close();

return 0;

}



III. Given a directed graph with two vertices ( source and destination). Design an

algorithm and implement it using a program to find the weight of the shortest path

from source to destination with exactly k edges on the path.

Input Format:

First input line will obtain number of vertices V present in the graph.

Graph in the form of adjacency matrix or adjacency list is taken as an input in next V

lines.

#include<bits/stdc++.h>

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

// Function to find the weight of the shortest path from source to destination with exactly k edges

int shortestPathWithKEdges(vector<vector<int>>& graph, int source, int destination, int k) {

int V = graph.size();

// Initialize 3D DP array

vector<vector<vector<int>>> dp(V, vector<vector<int>>(V, vector<int>(k + 1, INT\_MAX)));

// Initialize base cases

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

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

if (i == j) dp[i][j][0] = 0;

if (graph[i][j] != INT\_MAX) dp[i][j][1] = graph[i][j];

}

}

// Dynamic programming to find shortest path with exactly k edges

for (int e = 2; e <= k; ++e) {

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

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

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

if (dp[i][m][e - 1] != INT\_MAX && graph[m][j] != INT\_MAX) {

dp[i][j][e] = min(dp[i][j][e], dp[i][m][e - 1] + graph[m][j]);

}

}

}

}

}

// Return the weight of the shortest path from source to destination with exactly k edges

return dp[source][destination][k];

}

int main() {

// Example graph represented as an adjacency matrix with INT\_MAX indicating absence of edge

vector<vector<int>> graph = {

{0, 3, INT\_MAX, 7},

{INT\_MAX, 0, 1, 2},

{INT\_MAX, INT\_MAX, 0, INT\_MAX},

{INT\_MAX, INT\_MAX, 4, 0}

};

int source = 0; // Source vertex

int destination = 3; // Destination vertex

int k = 2; // Number of edges on the path

// Find the weight of the shortest path with exactly k edges

int weight = shortestPathWithKEdges(graph, source, destination, k);

if (weight != INT\_MAX) {

cout << "Weight of the shortest path from source to destination with exactly " << k << " edges: " << weight << endl;

} else {

cout << "No path with exactly " << k << " edges exists from source to destination." << endl;

}

return 0;

}

ss