Q-1: Given a weighted directed graph, find the shortest distance between every pair of vertices using Floyd-Warshall's algorithm.

Sample test case:

|  |
| --- |
| Input:  0, 5, INF, 10  INF, 0, 3, INF  INF, INF, 0, 1  INF, INF, INF, 0  Output:  The following matrix shows the shortest distances between every pair of vertices:  0 5 8 9  INF 0 3 4  INF INF 0 1  INF INF INF 0 |

Solution:

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 4

#define INF 99999

// A function to print the solution matrix

void printSolution(int dist[][V]);

void floydWarshall(int dist[][V])

{

int i, j, k;

/\* Add all vertices one by one to

the set of intermediate vertices.

---> Before start of an iteration,

we have shortest distances between all

pairs of vertices such that the

shortest distances consider only the

vertices in set {0, 1, 2, .. k-1} as

intermediate vertices.

----> After the end of an iteration,

vertex no. k is added to the set of

intermediate vertices and the set becomes {0, 1, 2, ..

k} \*/

for (k = 0; k < V; k++) {

// Pick all vertices as source one by one

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

// Pick all vertices as destination for the

// above picked source

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

// If vertex k is on the shortest path from

// i to j, then update the value of

// dist[i][j]

if (dist[i][j] > (dist[i][k] + dist[k][j])

&& (dist[k][j] != INF

&& dist[i][k] != INF))

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

// Print the shortest distance matrix

printSolution(dist);

}

/\* A utility function to print solution \*/

void printSolution(int dist[][V])

{

cout << "The following matrix shows the shortest distances between every pair of vertices \n";

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

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

if (dist[i][j] == INF){

cout << "INF"<< " ";

}

else{

cout << dist[i][j] << " ";

}

}

cout << endl;

}

}

int main()

{

int graph[V][V] = { { 0, 5, INF, 10 },

{ INF, 0, 3, INF },

{ INF, INF, 0, 1 },

{ INF, INF, INF, 0 } };

// Function call

floydWarshall(graph);

return 0;

}

Q-2: Given a graph and two nodes the task is to print the shortest path between two nodes using the Floyd Warshall algorithm.

Sample test case:

|  |
| --- |
| Input: node1=1, node2=3  Output: Shortest path from 1 to 3: 1 -> 2 -> 3  Input: node1=0, node2=2  Output: Shortest path from 0 to 2: 0 -> 1 -> 2 |

Solution:

#include <bits/stdc++.h>

using namespace std;

#define MAXN 100

// Infinite value for array

const int INF = 1e7;

int dis[MAXN][MAXN];

int Next[MAXN][MAXN];

// Initializing the distance and Next array

void initialise(int V,

vector<vector<int> >& graph)

{

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

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

dis[i][j] = graph[i][j];

// No edge between node i and j

if (graph[i][j] == INF)

Next[i][j] = -1;

else

Next[i][j] = j;

}

}

}

// Function construct the shortest path between u and v

vector<int> constructPath(int u,

int v)

{

// If there's no path between node u and v, simply return an empty array

if (Next[u][v] == -1)

return {};

// Storing the path in a vector

vector<int> path = { u };

while (u != v) {

u = Next[u][v];

path.push\_back(u);

}

return path;

}

// Standard Floyd Warshall Algorithm with little modification Now if we find

// that dis[i][j] > dis[i][k] + dis[k][j] then we modify next[i][j] = next[i][k]

void floydWarshall(int V)

{

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

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

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

// We cannot travel through

// edge that doesn't exist

if (dis[i][k] == INF|| dis[k][j] == INF){

continue;

}

if (dis[i][j] > dis[i][k] + dis[k][j]) {

dis[i][j] = dis[i][k]+ dis[k][j];

Next[i][j] = Next[i][k];

}

}

}

}

}

// Print the shortest path

void printPath(vector<int>& path)

{

int n = path.size();

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

cout << path[i] << " -> ";

cout << path[n - 1] << endl;

}

int main()

{

int V = 4;

vector<vector<int> > graph

= { { 0, 3, INF, 7 },

{ 8, 0, 2, INF },

{ 5, INF, 0, 1 },

{ 2, INF, INF, 0 } };

// Function to initialise the distance and Next array

initialise(V, graph);

// Calling Floyd Warshall Algorithm, this will update the shortest distance as well as Next array

floydWarshall(V);

vector<int> path;

// Path from node 1 to 3

cout << "Shortest path from 1 to 3: ";

path = constructPath(1, 3);

printPath(path);

// Path from node 0 to 2

cout << "Shortest path from 0 to 2: ";

path = constructPath(0, 2);

printPath(path);

// path from node 3 to 2

cout << "Shortest path from 3 to 2: ";

path = constructPath(3, 2);

printPath(path);

return 0;

}

Q-3: You are in a city that consists of n intersections numbered from 0 to n - 1 with bi-directional roads between some intersections.

The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between

any two intersections.

You are given an integer n and a 2D integer array roads where roads[i] = [ui, vi, timei] means that there is a road between intersections

ui and vi that takes timei minutes to travel. You want to know in how many ways you can travel from intersection 0 to intersection

n - 1 in the shortest amount of time.

Return the number of ways you can arrive at your destination in the shortest amount of time. Since the answer may be large,

return it modulo 109 + 7.

Sample test case:

|  |
| --- |
| Input: n = 7, roads = [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],[2,5,1],[0,4,5],[4,6,2]]  Output: 4 |

Solution:

#include <bits/stdc++.h>

#define ll long long

#define pll pair<ll, ll>

using namespace std;

const int mod = 1e9 + 7;

class Solution {

public:

// Function to count the number of ways to reach from 0 to n-1 in the shortest time

int countPaths(int n, vector<vector<int>>& roads) {

vector<vector<ll> >d (n,vector<ll>(n,LONG\_MAX));

vector<vector<ll> >w (n,vector<ll>(n,0));

// Initialize distance and weight arrays with road information

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

d[roads[i][0]][roads[i][1]]=roads[i][2];

d[roads[i][1]][roads[i][0]]=roads[i][2];

w[roads[i][0]][roads[i][1]]=1;

w[roads[i][1]][roads[i][0]]=1;

}

for(int i=0;i<n;i++) {w[i][i]=1;d[i][i]=0;}

// Floyd-Warshall algorithm

for(int k=0;k<n;k++){

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

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

if(i!=j and j!=k and k!=i and d[i][k]!=LONG\_MAX and d[k][j]!=LONG\_MAX){

if(d[i][j]>d[i][k]+d[k][j]){

w[i][j]=w[i][k];

d[i][j]=d[i][k]+d[k][j];

}

else if(d[i][j]==d[i][k]+d[k][j]){

w[i][j]=(w[i][j]+w[i][k])%mod;

}

}

}

}

}

// Return the number of ways to reach n-1 from 0 in shortest time

return w[0][n-1];

}

};

int main() {

Solution obj;

int n = 7;

vector<vector<int>> roads = {{0,6,7},{0,1,2},{1,2,3},{1,3,3},{6,3,3},{3,5,1},{6,5,1},{2,5,1},{0,4,5},{4,6,2}};

int result = obj.countPaths(n, roads);

cout << "Number of ways: " << result << endl;

return 0;

}