**Week 7**

**Q1) 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 into location of**

**his house and his friend’s 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 friend’s location to Akshay’s house)**

**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 weight.**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int minDisIndex(int \*dis,bool \*vis,int v)

{

int i;

int minDis=INT\_MAX;

int minIndex=-1;

for(i=0;i<v;i++)

{

if(vis[i]==false && dis[i]<=minDis)

{

minDis=dis[i];

minIndex=i;

}

}

return minIndex;

}

void dijkstra(vector<vector<int>> mat,int v,int s)

{

int dis[v];

bool vis[v];

int parent[v];

int i,j;

for(i=0;i<v;i++)

{

dis[i]=INT\_MAX;

vis[i]=false;

parent[i]=-1;

}

dis[s]=0;

parent[s]=s;

for(i=0;i<v;i++)

{

int m=minDisIndex(dis,vis,v);

vis[m]=true;

for(j=0;j<v;j++)

{

if(dis[m]!=INT\_MAX && !vis[j] && mat[m][j])

{

if(dis[j]>dis[m]+mat[m][j])

{

dis[j]=dis[m]+mat[m][j];

parent[j]=m;

}

}

}

}

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

if(i==s) {

cout<<i+1<<" : "<<dis[i]<<endl;

continue;

}

cout<<i+1;

j=i;

while(parent[j]!=s) {

cout<<" "<<parent[j]+1;

j=parent[j];

}

cout<<" "<<s+1<<" : "<<dis[i]<<endl;

}

}

int main()

{

int i,j;

int v;

cin>>v;

vector<vector<int>> mat(v,vector<int> (v));

for(i=0;i<v;i++)

for(j=0;j<v;j++)

cin>>mat[i][j];

int s;

cin>>s;

dijkstra(mat,v,s-1);

return 0;

}

**OUTPUT**

**A screen shot of numbers

Description automatically generated with low confidence**

**Q2) 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 weight.**

#include <bits/stdc++.h>

using namespace std;

void calulate(vector<int> &pa, int i)

{

cout << i + 1 << " ";

if (pa[i] >= 0)

calulate(pa, pa[i]);

}

void find\_path(int \*\*graph, int m, int sour)

{

vector<int> dis(m, INT\_MAX), pa(m, -1);

dis[sour] = 0;

for (int ki = 0; ki < m - 1; ki++)

{

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

{

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

{

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

{

if (dis[j] > dis[i] + graph[i][j])

{

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

pa[j] = i;

}

}

}

}

}

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

{

calulate(pa, i);

cout << ": " << dis[i] << endl;

}

}

int main()

{

int m, source, ed;

cin >> m;

int \*\*graph = (int \*\*)malloc(m \* sizeof(int \*));

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

graph[i] = (int \*)malloc(m \* sizeof(int));

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

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

cin >> graph[i][j];

}

}

cin >> source;

find\_path(graph, m, source - 1);

}

**OUTPUT**

A screen shot of a calculator

Description automatically generated with low confidence

**Q3) 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.**

**Next input line will obtain source and destination vertex number.**

**Last input line will obtain value k.**

**Output Format:**

**Output will be the weight of shortest path from source to destination having exactly k edges.**

**If no path is available, then print “no path of length k is available”.**

#include <bits/stdc++.h>

using namespace std;

int shortest\_weigt(int \*\*graph, int ver, int u, int v, int k)

{

if (k <= 0)

return INT\_MAX;

if (k == 0 && u == v)

return 0;

if (k == 1 && graph[u][v] != INT\_MAX)

return graph[u][v];

int res = INT\_MAX;

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

if (graph[u][i] != 0 && u != i && v != i) {

int recu = shortest\_weigt(graph, ver, i, v, k - 1);

if (recu != INT\_MAX)

res = min(res, graph[u][i] + recu);

}

}

return res;

}

int main()

{

int ver, u, v, k, ans;

cin >> ver;

int \*\*graph = (int \*\*)malloc(ver \* sizeof(int \*));

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

graph[i] = (int \*)malloc(sizeof(int) \* ver);

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

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

cin >> graph[i][j];

cin >> u >> v >> k;

ans = shortest\_weigt(graph, ver, u - 1, v - 1, k);

cout << "Weight of shortest path from (" << u

<< "," << v << ") with " << k << " edges :" << ans;

}

**OUTPUT**

**Text

Description automatically generated**