**Registration Number: 19BCE2119**

**Name: Gaurav Kumar Singh**

**Course: Network and Communication CSE1004**

**Digital Assignment 4**

**Bellman-Ford Algorithm**

#include<bits/stdc++.h>

using namespace std;

struct edge{

int src,dst,wt;

};

int V,E;

void bellmanFord(vector<edge>& Edges)

{

int parent[V];

int cost\_parent[V];

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

parent[0] = -1;

value[0] = 0;

bool updated;

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

{

updated = false;

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

{

int U = Edges[j].src;

int V = Edges[j].dst;

int wt = Edges[j].wt;

if(value[U]!=INT\_MAX and value[U]+wt<value[V])

{

value[V] = value[U]+wt;

parent[V] = U;

cost\_parent[V] = value[V];

updated = true;

}

}

if(updated==false)

break;

}

for(int j=0;j<E and updated==true;++j)

{

int U = Edges[j].src;

int V = Edges[j].dst;

int wt = Edges[j].wt;

if(value[U]!=INT\_MAX and value[U]+wt<value[V])

{

cout<<"Graph has -VE edge cycle\n";

return;

}

}

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

cout<<"U->V: "<<parent[i]<<"->"<<i<<" Cost to reach "<<i<<" from source 0 = "<<value[i]<<"\n";

}

int main()

{

cout<<"Enter no. of vertices and edges:\n";

cin>>V>>E;

vector<edge> Edges(E);

int src,dst,wt;

cout<<"\nEnter the edges with their weight in the format `Source Destination Weight` in a directed manner: \n";

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

{

cin>>src>>dst>>wt;

Edges[i].src = src;

Edges[i].dst = dst;

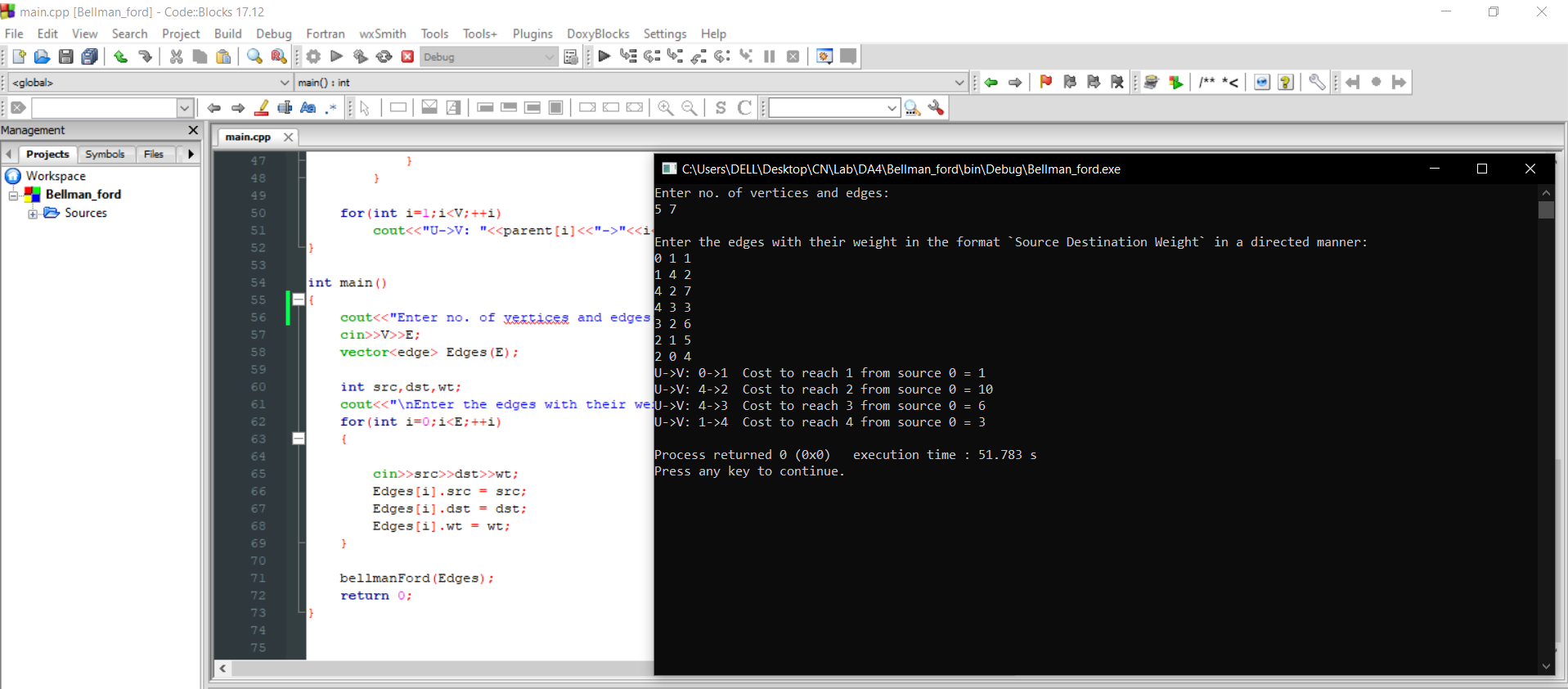
Edges[i].wt = wt;

}

bellmanFord(Edges);

return 0;

}



**Dijkstra Algorithm**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main() {

int G[MAX][MAX], i, j, n, u;

printf("Enter no. of vertices:");

scanf("%d", & n);

printf("\nEnter the adjacency matrix:\n");

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

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

scanf("%d", & G[i][j]);

printf("\nEnter the starting node:");

scanf("%d", & u);

dijkstra(G, n, u);

return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode) {

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

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

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

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

cost[i][j] = INFINITY;

else

cost[i][j] = G[i][j];

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

distance[i] = cost[startnode][i];

pred[i] = startnode;

visited[i] = 0;

}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n - 1) {

mindistance = INFINITY;

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

if (distance[i] < mindistance && !visited[i]) {

mindistance = distance[i];

nextnode = i;

}

visited[nextnode] = 1;

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

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i]) {

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

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

if (i != startnode) {

printf("\nDistance of node%d=%d", i, distance[i]);

printf("\nPath=%d", i);

j = i;

do {

j = pred[j];

printf("<-%d", j);

} while (j != startnode);

}

}

