**Agnim Gupta**

**2028083**

**A-23,CSSE**

**Question 1**

#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;

    //pred[] stores the predecessor of each node

    //count gives the number of nodes seen so far

    //create the cost matrix

    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];

    //initialize pred[],distance[] and visited[]

    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;

        //nextnode gives the node at minimum distance

        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++;

    }

    //print the path and distance of each node 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);

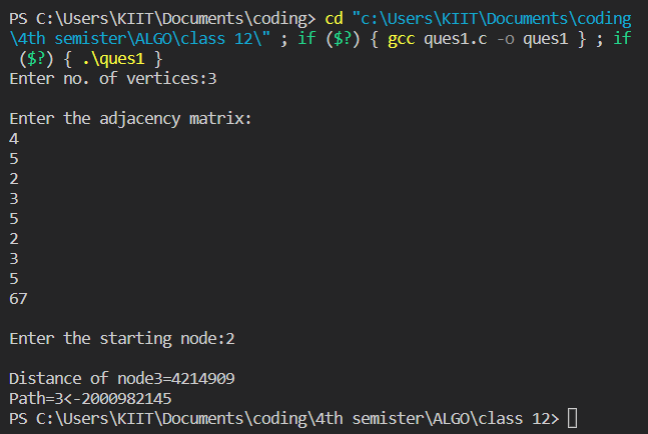
        }

        while(j!=startnode);

    }

}

**Output**

****

**Question 2**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

struct Edge

{

    int source, destination, weight;

};

struct Graph

{

    int V, E;

    // V is number of vertices and E is number of edges

    struct Edge\* edge;

    // This structure contain another structure which we already created edge.

};

struct Graph\* createGraph(int V, int E)

{

    struct Graph\* graph = (struct Graph\*) malloc( sizeof(struct Graph));

    //Allocating space to structure graph

    graph->V = V; //assigning values to structure elements that taken form user. graph->E = E;

    graph->edge = (struct Edge\*) malloc( graph->E \* sizeof( struct Edge ) );

    //Creating "Edge" type structures inside "Graph" structure, the number of edge type structures are equal to number of edges

    return graph;

}

void FinalSolution(int dist[], int n)

{

// This function prints the final solution printf("\nVertex\tDistance from Source Vertex\n"); int i;

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

    printf("%d \t\t %d\n", i, dist[i]);

}

}

void BellmanFord(struct Graph\* graph, int source)

{

    int V = graph->V; int E = graph->E;

    int StoreDistance[V]; int i,j;

    // This is initial step that we know , we initialize all distance to infinity except source.

    // We assign source distance as 0(zero)

    for (i = 0; i < V; i++) StoreDistance[i] = INT\_MAX;

    StoreDistance[source] = 0;

    //The shortest path of graph that contain V vertices, never contain "V-1" edges. So we do here "V- 1" relaxations

    for (i = 1; i <= V-1; i++)

    {

        for (j = 0; j < E; j++)

        {

            int u = graph->edge[j].source;

            int v = graph->edge[j].destination; int weight = graph->edge[j].weight;

            if (StoreDistance[u] + weight < StoreDistance[v]) StoreDistance[v] = StoreDistance[u] + weight;

        }

    }

    // Actually upto now shortest path found. But BellmanFord checks for negative edge cycle. In this step we check for that

    // shortest distances if graph doesn't contain negative weight cycle.

    // If we get a shorter path, then there is a negative edge cycle.

    for (i = 0; i < E; i++)

    {

    int u = graph->edge[i].source;

    int v = graph->edge[i].destination;

    int weight = graph->edge[i].weight;

    if (StoreDistance[u] + weight < StoreDistance[v])

    printf("This graph contains negative edge cycle\n");

    }

    FinalSolution(StoreDistance, V);

    return;

}

int main()

{

    int V,E,S; //V = no.of Vertices, E = no.of Edges, S is source vertex

    printf("Enter number of vertices in graph\n");

    scanf("%d",&V);

    printf("Enter number of edges in graph\n");

    scanf("%d",&E);

    printf("Enter your source vertex number\n");

    scanf("%d",&S);

    struct Graph\* graph = createGraph(V, E); //calling the function to allocate space to these many vertices and edges

    int i; for(i=0;i<E;i++){

        printf("\nEnter edge %d properties Source, destination, weight respectively\n",i+1);

        scanf("%d",&graph->edge[i].source);

        scanf("%d",&graph->edge[i].destination);

        scanf("%d",&graph->edge[i].weight);

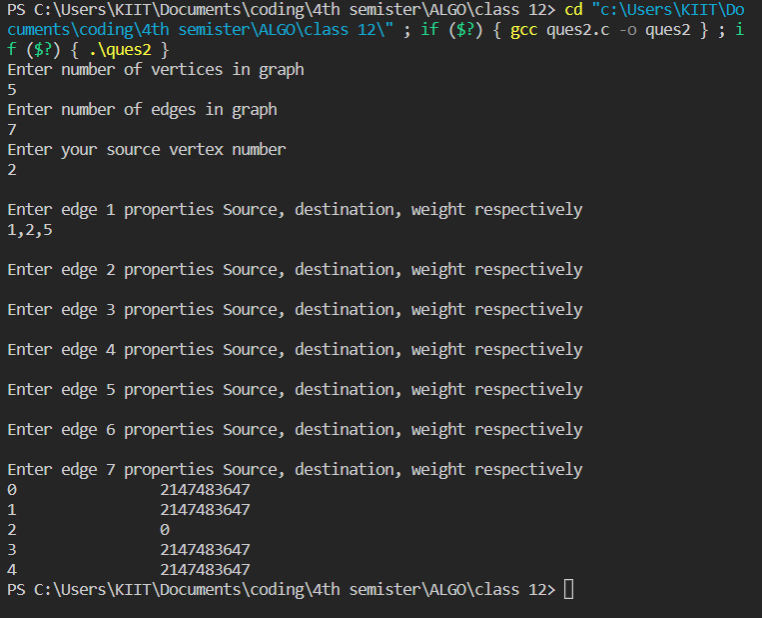
    }

    BellmanFord(graph, S);

    //passing created graph and source vertex to BellmanFord Algorithm function return 0;

}

**Output**

****