**Assignment: 8**

**Name: Shishu**

**Reg.No:2020CA089**

**1.Write a C program to implement Kruskal's algorithm to find Minimum Spanning Tree.**

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define max 10000

int cost[100][100],parent[100];

int find(int x)

{

while(parent[x])

x=parent[x];

return x;

}

int uni(int x,int y)

{

if(x!=y)

{

parent[y]=x;

return 1;

}

return 0;

}

void main()

{

printf("\nKruskal's algorithm\n");

int n;

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

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

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

{

for(int j=1;j<=n;j++)

{

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

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

cost[i][j]=max;

}

}

int a,b,u,v;

int min,min\_cost=0,t=1;

printf("The edges of Minimum Cost Spanning Tree are\n");

while(t < n)

{

min=max;

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

{

for(int j=1;j <= n;j++)

{

if(cost[i][j] < min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("edge (%d,%d) =%d\n",a,b,min);

min\_cost +=min;

}

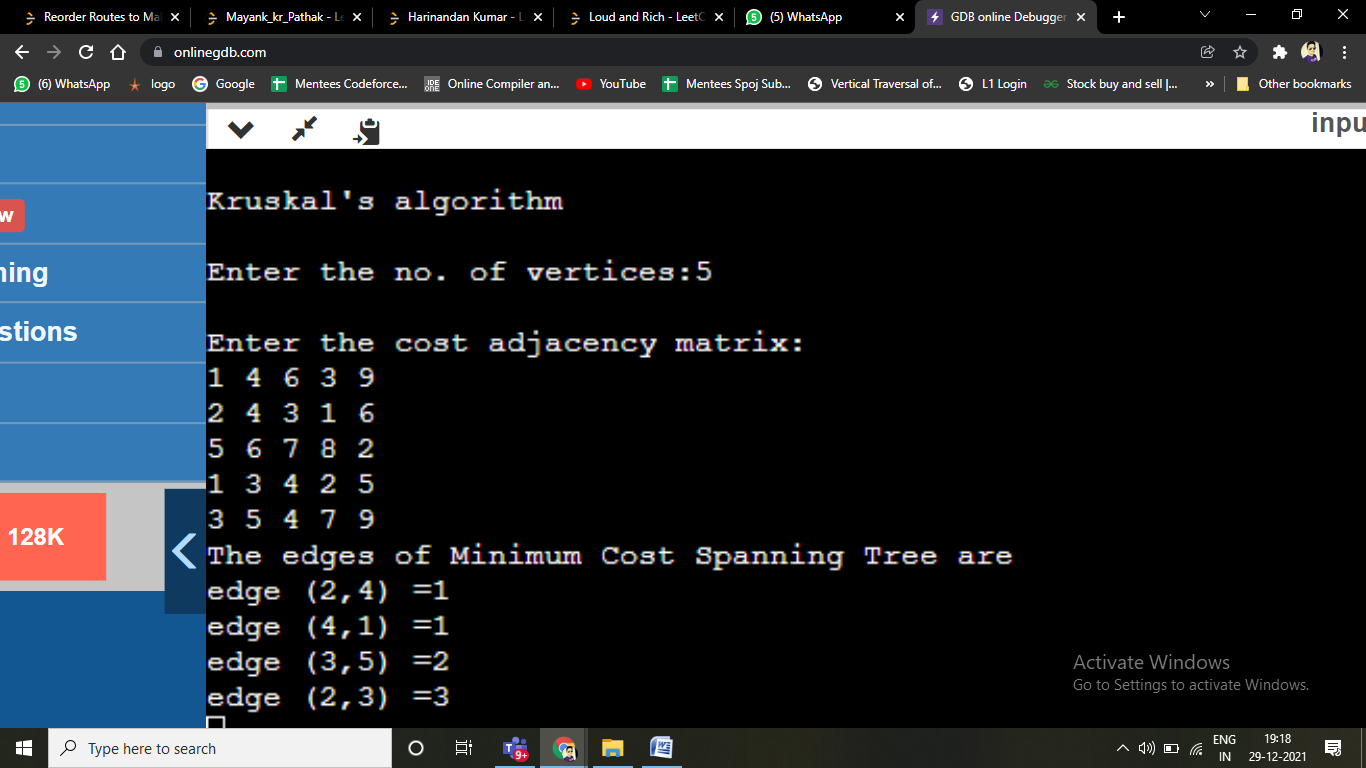
cost[a][b]=cost[b][a]=max;

}

printf("\nMinimum cost = %d\n",min\_cost);

}

**Output:**

****

**2.Write a c program to implement DFS.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

struct Vertex {

char label;

bool visited;

};

//stack variables

int stack[MAX];

int top = -1;

//graph variables

//array of vertices

struct Vertex\* lstVertices[MAX];

//adjacency matrix

int adjMatrix[MAX][MAX];

//vertex count

int vertexCount = 0;

//stack functions

void push(int item) {

stack[++top] = item;

}

int pop() {

return stack[top--];

}

int peek() {

return stack[top];

}

bool isStackEmpty() {

return top == -1;

}

void addVertex(char label) {

struct Vertex\* vertex = (struct Vertex\*) malloc(sizeof(struct Vertex));

vertex->label = label;

vertex->visited = false;

lstVertices[vertexCount++] = vertex;

}

void addEdge(int start,int end) {

adjMatrix[start][end] = 1;

adjMatrix[end][start] = 1;

}

void displayVertex(int vertexIndex) {

printf("%c ",lstVertices[vertexIndex]->label);

}

//get the adjacent unvisited vertex

int getAdjUnvisitedVertex(int vertexIndex) {

int i;

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

if(adjMatrix[vertexIndex][i] == 1 &&lstVertices[i]->visited == false) {

return i;

}

}

return -1;

}

void dfs() {

int i;

lstVertices[0]->visited = true;

displayVertex(0);

push(0);

while(!isStackEmpty()) {

int unvisitedVertex = getAdjUnvisitedVertex(peek());

if(unvisitedVertex == -1) {

pop();

} else {

lstVertices[unvisitedVertex]->visited = true;

displayVertex(unvisitedVertex);

push(unvisitedVertex);

}

}

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

lstVertices[i]->visited = false;

}

}

int main() {

int i, j;

for(i = 0; i< MAX; i++){ // set adjacency (problem with braces !!)

for(j = 0; j < MAX; j++) // matrix to 0

adjMatrix[i][j] = 0;

}

addVertex('X');

addVertex('Y');

addVertex('Z');

addVertex('W');

addVertex('A');

addEdge(0, 1);

addEdge(0, 2);

addEdge(0, 3);

addEdge(1, 4);

addEdge(2, 4);

addEdge(3, 4);

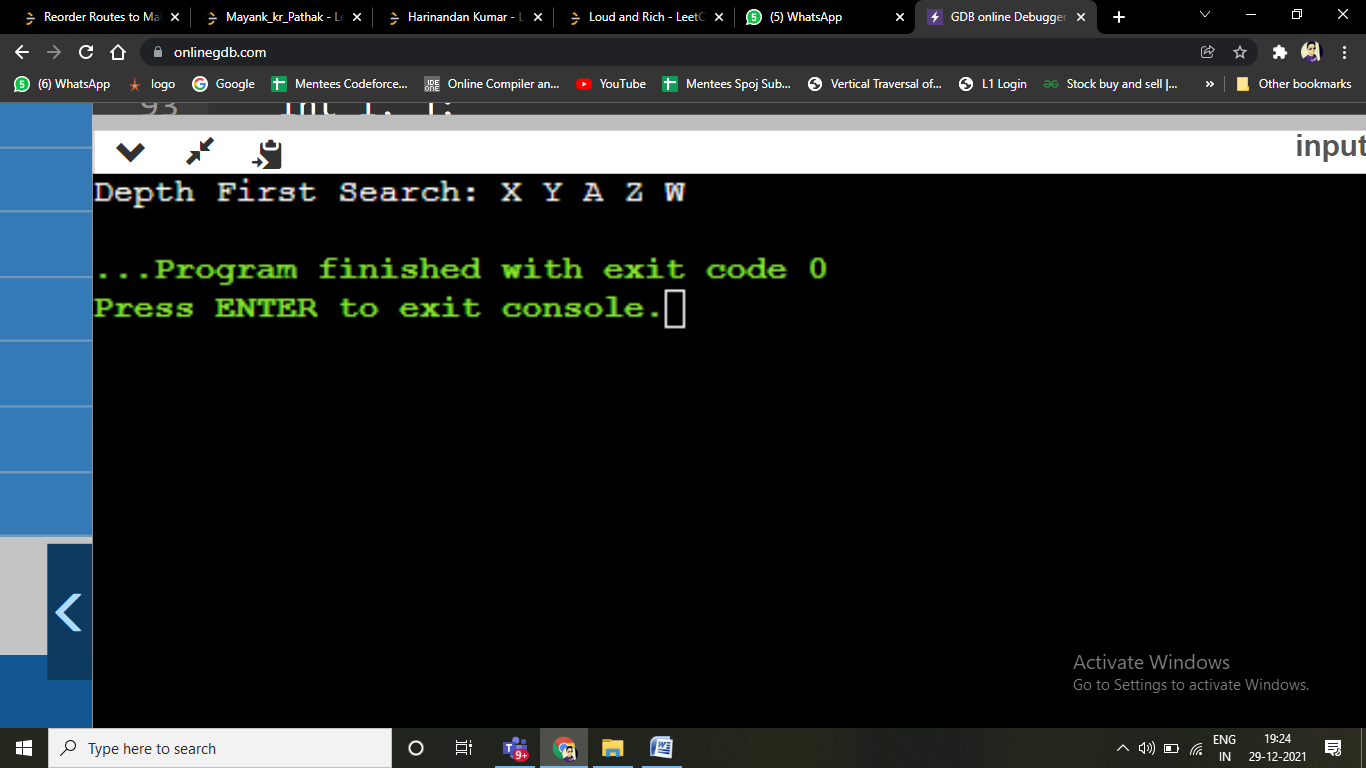
printf("Depth First Search: ");

dfs();

return 0;

}

**Output:**

****