**Experiment 20: Write a C program to implement the Prim’s Algorithm.**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

printf("\nEnter the number of nodes:");

scanf("%d",&n);

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

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

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

{

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

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

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

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

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

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

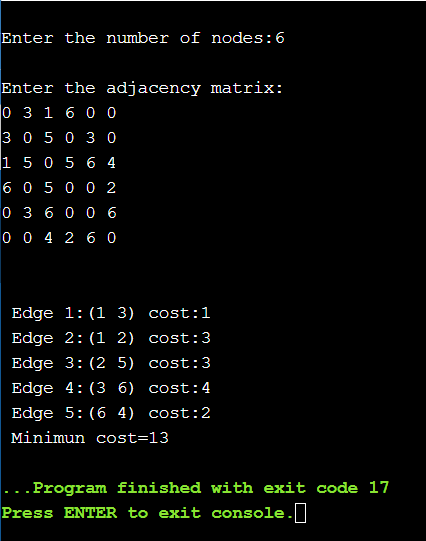
cost[a][b]=cost[b][a]=999;

}

printf("\n Minimun cost=%d",mincost);

}

**OUTPUT-**



**Experiment 21: Write a C program to implement the Kruskal’s Algorithm.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

printf("\n\t Implementation of Kruskal's algorithm\n");

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

scanf("%d",&n);

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

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

{

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

{

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

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

cost[i][j]=999;

}

}

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

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

for(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("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\tMinimum cost = %d\n",mincost);

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

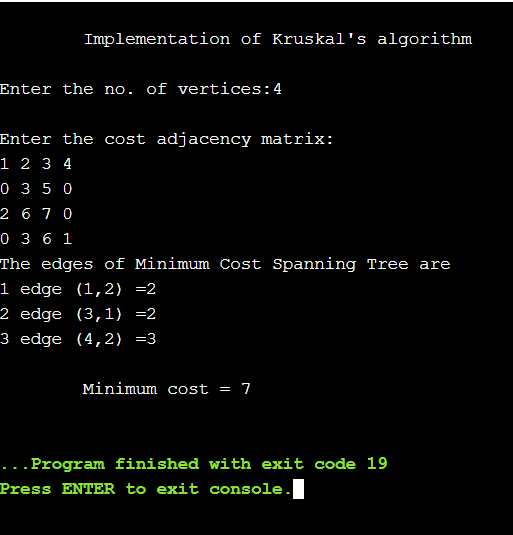
return 1;

}

return 0;

}

**OUTPUT-**

****

**Experiment 22: Write a C program to implement Breadth First Search.**

#include<stdio.h>

#include<conio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {

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

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r) {

visited[q[f]] = 1;

bfs(q[f++]);

}

}

int main() {

int v;

printf("Enter the number of vertices: ");

scanf("%d",&n);

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

q[i] = 0;

visited[i] = 0;

}

printf("\nEnter graph data in matrix form:\n");

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

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

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

}

}

printf("Enter the starting vertex: ");

scanf("%d", &v);

bfs(v);

printf("\nThe node which are reachable are:");

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

if(visited[i])

printf(" %d", i);

else {

printf("\nBFS is not possible. All nodes are not reachable!");

break;

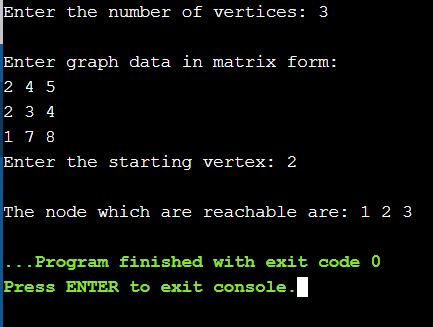
}

}

return 0;

}

**OUTPUT-**



**Experiment 23: Write a C program to implement Depth First Search.**

#include<stdio.h>

#include<stdlib.h>

int DepthFirstSearch(int);

int n;

int graph[20][20], completed[10];

int DepthFirstSearch(int i)

{

int j;

printf("%d\n", i);

completed[i] = 1;

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

{

if(!completed[j] && graph[i][j] == 1)

{

DepthFirstSearch(j);

}

}

return 0;

}

int main()

{

int i, j;

printf("\n enter number of vertices:");

scanf("%d",&n);

printf("\n enter adjacency matrix :\n");

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

{

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

{

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

}

}

printf("Depth First Search is:-\n");

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

{

completed[i]=0;

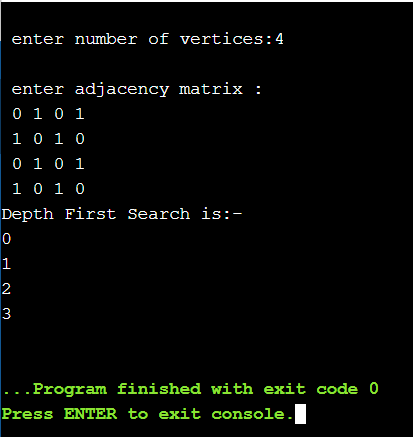
}

DepthFirstSearch(0);

return 0;

}

**OUTPUT-**

****

**Experiment 24: Write a C program to find shortest paths to other vertices from a given vertex in a weighted connected graph using Dijkstra’s algorithm**.

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

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

void main(){

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

printf("\nEnter the 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);

dijikstra(G,n,u);

}

void dijikstra(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 %d = %d", i, distance[i]);

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

j=i;

do

{

j=pred[j];

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

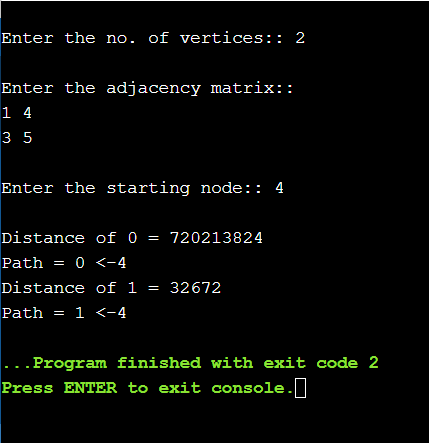
}

while(j!=startnode);

}

}

**OUTPUT-**



**Experiment 25: Write a C program to find shortest paths to other vertices from a given vertex in a weighted connected graph using Bellman-Ford algorithm.**

#include <iostream>

#include <vector>

#include <iomanip>

#include <climits>

using namespace std;

struct Edge

{

int source, dest, weight;

};

void printPath(vector<int> const &parent, int vertex)

{

if (vertex < 0)

return;

printPath(parent, parent[vertex]);

cout << vertex << " ";

}

void bellmanFord(vector<Edge> const &edges, int source, int N)

{

vector<int> distance (N, INT\_MAX);

distance[source] = 0;

vector<int> parent (N, -1);

int u, v, w, k = N;

while (--k)

{

for (Edge edge: edges)

{

u = edge.source;

v = edge.dest;

w = edge.weight;

if (distance[u] != INT\_MAX && distance[u] + w < distance[v])

{

distance[v] = distance[u] + w;

parent[v] = u;

}

}

}

for (Edge edge: edges)

{

u = edge.source;

v = edge.dest;

w = edge.weight;

if (distance[u] != INT\_MAX && distance[u] + w < distance[v])

{

cout << "Negative Weight Cycle Found!!";

return;

}

}

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

{

cout << "Distance of vertex " << i << " from the source is "

<< setw(2) << distance[i] << ". It's path is [ ";

printPath(parent, i); cout << "]" << '\n';

}

}

int main()

{

vector<Edge> edges =

{

{ 0, 1, -1 }, { 0, 2, 4 }, { 1, 2, 3 }, { 1, 3, 2 },

{ 1, 4, 2 }, { 3, 2, 5 }, { 3, 1, 1 }, { 4, 3, -3 }

};

int N = 5;

int source = 0;

bellmanFord(edges, source, N);

return 0;

}

**OUTPUT-**

