**Graph traversals**

**R.Harini**

**18BCE1010**

**Prims Algorithm:**

#include<stdio.h>

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

int visited[10]= {0}

,min,mincost=0,cost[10][10];

int main() {

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter 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);

}

**DFS:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int adj[100][100];

int n, state[100];

int stack[100], top=-1;

void push(int vertex){

if (top==MAX-1){

printf("Stack is full\n");

}

else{

top++;

stack[top]=vertex;

}

}

int is\_empty(){

if (top==-1)

return 1;

else

return 0;

}

int pop(){

int d;

if (top==-1){

printf("Stack is empty\n");

exit(1);

}

d=stack[top];

top--;

return d;

}

void create\_graph(){

int count, max\_edge, origin,dest;

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

scanf("%d",&n);

max\_edge=n\*(n-1);

for (count=1;count<max\_edge;count++){

printf("Enter edge %d(-1 -1 to quit):", count);

scanf("%d %d",&origin, &dest);

if ((origin==-1)&&(dest==-1))break;

if (origin>=n||dest>=n||origin<0||dest<0){

printf("Invalid edge!\n");

count--;

}

else{

adj[origin][dest]=1;

}

}

}

void DFS(int v){

int i;

push(v);

state[v]=waiting;

while(!is\_empty()){

v=pop();

printf("%d",v);

state[v]=visited;

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

if (adj[v][i]==1 && state[i]==initial){

push(i);

state[i]=waiting;

}

}

}

printf("\n");

}

void DF\_Traversal()

{

int v;

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

state[v] = initial;

printf("Enter Start Vertex for DFS: \n");

scanf("%d", &v);

DFS(v);

}

int main()

{

create\_graph();

DF\_Traversal();

return 0;

}

**BFS:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int adj[100][100];

int n, state[100];

int queue[100], front=-1, rear=-1;

void enqueue(int vertex){

if (rear==MAX-1){

printf("Queue is full\n");

}

else{

if (front==-1){

front=0;}

rear++;

queue[rear]=vertex;

}

}

int is\_empty(){

if (front==-1||front>rear)

return 1;

else

return 0;

}

int dequeue(){

int delete\_item;

if (front==-1||front>rear){

printf("Queue is empty\n");

exit(1);

}

delete\_item=queue[front];

front=front+1;

return delete\_item;

}

void create\_graph(){

int count, max\_edge, origin,dest;

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

scanf("%d",&n);

max\_edge=n\*(n-1);

for (count=1;count<max\_edge;count++){

printf("Enter edge %d(-1 -1 to quit):", count);

scanf("%d %d",&origin, &dest);

if ((origin==-1)&&(dest==-1))break;

if (origin>=n||dest>=n||origin<0||dest<0){

printf("Invalid edge!\n");

count--;

}

else{

adj[origin][dest]=1;

}

}

}

void BFS(int v){

int i;

enqueue(v);

state[v]=waiting;

while(!is\_empty()){

v=dequeue();

printf("%d",v);

state[v]=visited;

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

if (adj[v][i]==1 && state[i]==initial){

enqueue(i);

state[i]=waiting;

}

}

}

printf("\n");

}

void BF\_Traversal()

{

int v;

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

state[v] = initial;

printf("Enter Start Vertex for BFS: \n");

scanf("%d", &v);

BFS(v);

}

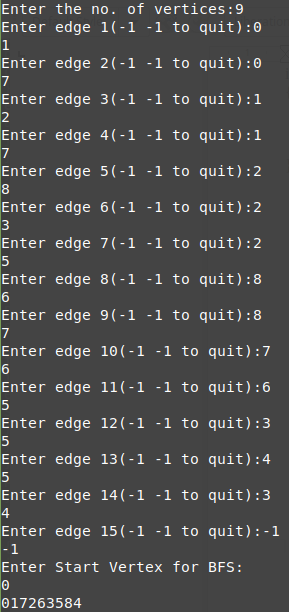
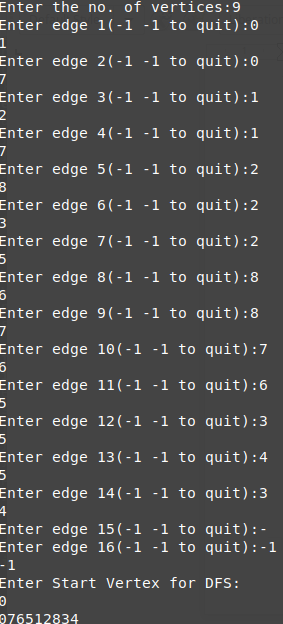
int main()

{

create\_graph();

BF\_Traversal();

return 0;

}