1: Write a C Program to implement graphs using adjacency matrix

* Insert vertex
* Insert edge
* Delete vertex
* Delete edge
* Display graph

Program:

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int adj[MAX][MAX];

int n;

void create\_graph();

void display();

void insert\_vertex(int x);

void del\_vertex(int x);

void insert\_edge(int origin,int destin);

void del\_edge(int origin,int destin);

int main()

{

int choice,origin,destin,x;

create\_graph();

printf("\n");

while(1)

{

printf("1:insert vertex\n");

printf("2:insert edge\n");

printf("3:delete vertex\n");

printf("4:delete edge\n");

printf("5:display\n");

printf("6:Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the vertex to be inserted: ");

scanf("%d",&x);

insert\_vertex(x);

printf("\n");

break;

case 2:

printf("Enter an edge to be inserted: ");

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

insert\_edge(origin,destin);

printf("\n");

break;

case 3:

printf("Enter the vertex to be deleted: ");

scanf("%d",&x);

del\_vertex(x);

printf("\n");

break;

case 4:

printf("Enter an edge to be deleted: ");

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

del\_edge(origin,destin);

printf("\n");

break;

case 5:

display();

printf("\n");

break;

case 6:

exit(1);

default:

printf("wrong choice\n");

break;

}

}

}

void create\_graph()

{

int max\_edges,i,j,origin,destin;

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

scanf("%d",&n);

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

for(i=1;i<=max\_edges;i++)

{

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

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

if(origin==-1 && destin==-1)

break;

if(origin>=n || destin>=n || origin<0 || destin<0)

{

printf("Invalid vertex\n");

i--;

}

else

{

adj[origin][destin]=1;

}

}

}

void insert\_edge(int origin,int destin)

{

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

{

printf("origin vertex does not exist\n");

return;

}

if(destin<0 || destin>=n)

{

printf("destin vertex does not exist\n");

return;

}

adj[origin][destin]=1;

}

void del\_edge(int origin,int destin)

{

if(origin>=n || destin>=n || origin<0 || destin<0 || adj[origin][destin]==0)

{

printf("Edge does not exist\n");

return;

}

adj[origin][destin]=0;

}

void display()

{

int i,j;

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

{

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

printf("%d ",adj[i][j]);

printf("\n");

}

}

void insert\_vertex(int x)

{

n++;

int i;

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

{

adj[i][n-1]=0;

adj[n-1][i]=0;

}

}

void del\_vertex(int x)

{

int i;

if(x>n)

{

printf("Vertex not present\n");

return;

}

while(x<n)

{

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

{

adj[i][x]=adj[i][x+1];

}

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

{

adj[x][i]=adj[x+1][i];

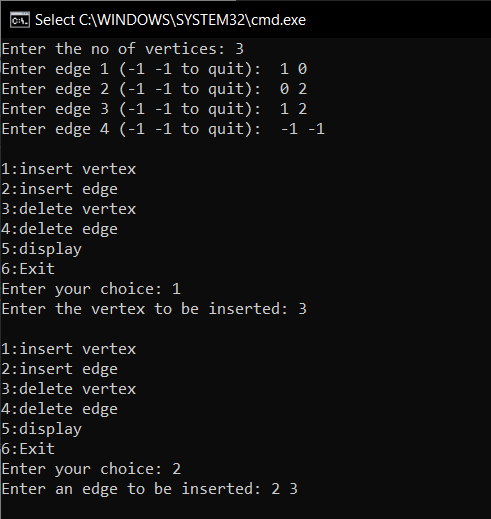
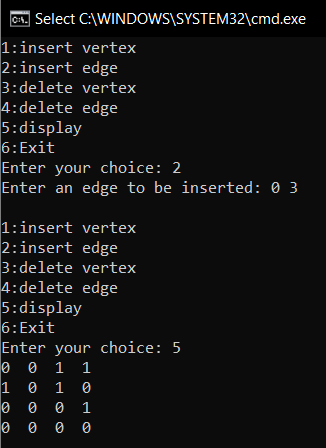
}

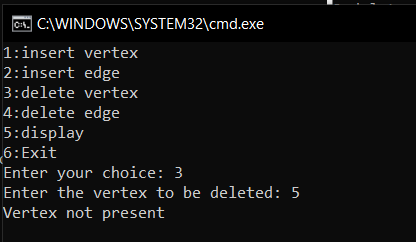
}

n--;

}

Output:



2: Write a C Program to traverse a directed graph using DFS and BFS

Program:

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

#define DFSvisited 4

int n;

int adj[MAX][MAX];

int state[MAX];

void create\_graph();

void BF\_traversal();

void BFS(int v);

void DF\_traversal();

void DFS(int v);

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

int stack[MAX], top=-1;

void insert(int vertex);

int del();

int isEmpty\_queue();

void push(int vertex);

int pop();

int isEmpty\_stack();

int main()

{

create\_graph();

BF\_traversal();

DF\_traversal();

}

void create\_graph()

{

int max\_edges,i,j,origin,destin;

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

scanf("%d",&n);

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

for(i=1;i<=max\_edges;i++)

{

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

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

if(origin==-1 && destin==-1)

break;

if(origin>=n || destin>=n || origin<0 || destin<0)

{

printf("Invalid vertex\n");

i--;

}

else

{

adj[origin][destin]=1;

}

}

printf("\n");

}

void BF\_traversal()

{

int v;

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

state[v]=initial;

printf("Enter starting vertex for BF traversal: ");

scanf("%d",&v);

printf("BFS traversal is: \n");

BFS(v);

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

if(state[v]==initial)

BFS(v);

}

void BFS(int v)

{

int i;

insert(v);

state[v]=waiting;

while(!isEmpty\_queue())

{

v=del();

printf("%d ",v);

state[v]=visited;

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

{

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

{

insert(i);

state[i]=waiting;

}

}

}

printf("\n");

}

void insert(int vertex)

{

if(rear==MAX-1)

printf("Queue overflow\n");

else

{

if(front==-1)

front=0;

rear=rear+1;

queue[rear]=vertex;

}

}

int del()

{

int item;

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

{

printf("Queue underflow\n");

return;

}

item=queue[front];

front=front+1;

return item;

}

int isEmpty\_queue()

{

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

return 1;

else

return 0;

}

void DF\_traversal()

{

int v;

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

state[v]=initial;

printf("enter starting vertex for DF traversal: ");

scanf("%d",&v);

printf("DFS traversal is: \n");

DFS(v);

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

{

if(state[v]==initial)

DFS(v);

}

}

void DFS(int v)

{

int i;

push(v);

while(!isEmpty\_stack())

{

v=pop();

if(state[v]==initial)

{

printf("%d ",v);

state[v]= DFSvisited;

}

for(i=n-1;i>=0;i--)

{

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

push(i);

}

}

}

void push(int v)

{

if(top==MAX-1)

printf("Stack overflow\n");

else

{

top=top+1;

stack[top]=v;

}

}

int pop()

{

int v;

if(top==-1)

{

printf("stack underflow\n");

return;

}

v=stack[top];

top=top-1;

return v;

}

int isEmpty\_stack()

{

if(top==-1)

return 1;

else

return 0;

}

