/\* C Program to implement DFS Algorithm for Connected Graph \*/

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define visited 2

int n; /\* Number of nodes in the graph \*/

int adj[MAX][MAX]; /\*Adjacency Matrix\*/

int state[MAX]; /\*Can be initial or visited \*/

void DF\_Traversal();

void DFS(int v);

void create\_graph();

int stack[MAX];

int top = -1;

void push(int v);

int pop();

int isEmpty\_stack();

main()

{

create\_graph();

DF\_Traversal();

}/\*End of main()\*/

void DF\_Traversal()

{

int v;

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

state[v]=initial;

printf("\nEnter starting node for Depth First Search : ");

scanf("%d",&v);

DFS(v);

printf("\n");

}/\*End of DF\_Traversal( )\*/

void DFS(int v)

{

int i;

push(v);

while(!isEmpty\_stack())

{

v = pop();

if(state[v]==initial)

{

printf("%d ",v);

state[v]=visited;

}

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

{

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

push(i);

}

}

}/\*End of DFS( )\*/

void push(int v)

{

if(top == (MAX-1))

{

printf("\nStack Overflow\n");

return;

}

top=top+1;

stack[top] = v;

}/\*End of push()\*/

int pop()

{

int v;

if(top == -1)

{

printf("\nStack Underflow\n");

exit(1);

}

else

{

v = stack[top];

top=top-1;

return v;

}

}/\*End of pop()\*/

int isEmpty\_stack( )

{

if(top == -1)

return 1;

else

return 0;

}/\*End if isEmpty\_stack()\*/

void create\_graph()

{

int i,max\_edges,origin,destin;

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

scanf("%d",&n);

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

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

{

printf("\nEnter 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("\nInvalid edge!\n");

i--;

}

else

{

adj[origin][destin] = 1;

}

}

}

OUTPUT : :

Enter number of nodes : 6

Enter edge 1( -1 -1 to quit ) : 0 1

Enter edge 2( -1 -1 to quit ) : 0 2

Enter edge 3( -1 -1 to quit ) : 0 3

Enter edge 4( -1 -1 to quit ) : 1 3

Enter edge 5( -1 -1 to quit ) : 3 4

Enter edge 6( -1 -1 to quit ) : 4 2

Enter edge 7( -1 -1 to quit ) : 5 5

Enter edge 8( -1 -1 to quit ) : -1 -1

Enter starting node for Depth First Search : 0

0 1 3 4 2

Process returned 0

/\* C Program to implement BFS Algorithm for Disconnected Graph \*/

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n; /\*Number of vertices in the graph\*/

int adj[MAX][MAX]; /\*Adjacency Matrix\*/

int state[MAX]; /\*can be initial, waiting or visited\*/

void create\_graph();

void BF\_Traversal();

void BFS(int v);

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

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main()

{

create\_graph();

BF\_Traversal();

return 0;

}/\*End of main()\*/

void BF\_Traversal()

{

int v;

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

state[v]=initial;

printf("\nEnter starting vertex for Breadth First Search : ");

scanf("%d", &v);

BFS(v);

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

if(state[v] == initial)

BFS(v);

}/\*End of BF\_Traversal()\*/

void BFS(int v)

{

int i;

insert\_queue(v);

state[v]=waiting;

while( !isEmpty\_queue() )

{

v = delete\_queue( );

printf("%d ",v);

state[v] = visited;

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

{

/\* Check for adjacent unvisited vertices \*/

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

{

insert\_queue(i);

state[i] = waiting;

}

}

}

printf("\n");

}/\*End of BFS()\*/

void insert\_queue(int vertex)

{

if (rear == MAX-1)

printf("Queue Overflow\n");

else

{

if (front == -1) /\*If queue is initially empty \*/

front = 0;

rear = rear+1;

queue[rear] = vertex ;

}

}/\*End of insert\_queue()\*/

int isEmpty\_queue()

{

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

return 1;

else

return 0;

}/\*End of isEmpty\_queue()\*/

int delete\_queue()

{

int del\_item;

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

{

printf("\nQueue Underflow\n");

exit(1);

}

del\_item = queue[front];

front = front+1;

return del\_item;

}/\*End of delete\_queue() \*/

void create\_graph()

{

int i,max\_edges,origin,destin;

printf("\nEnter number of vertices : ");

scanf("%d",&n);

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

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

{

printf("\nEnter 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("\nInvalid edge!\n");

i--;

}

else

{

adj[origin][destin]=1;

}

}/\*End of for\*/

}/\*End of create\_graph()\*/

OUTPUT : :

Enter number of vertices : 5

Enter edge 1( -1 -1 to quit ) : 0 1

Enter edge 2( -1 -1 to quit ) : 0 2

Enter edge 3( -1 -1 to quit ) : 0 3

Enter edge 4( -1 -1 to quit ) : 1 3

Enter edge 5( -1 -1 to quit ) : 3 2

Enter edge 6( -1 -1 to quit ) : 4 4

Enter edge 7( -1 -1 to quit ) : -1 -1

Enter starting vertex for Breadth First Search : 0

0 1 2 3

4

Process returned 0