BFS

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100
int n;
int adj[MAX][MAX];
int visited[MAX];
int level[MAX];

void create_graph();
void represent_graph();
void BFS();
void printlevel();

int queue[MAX], front = -1,rear = -1;
void enQueue(int vertex);
int deQueue();
int isEmpty_queue();

int main()
{
   create_graph();
   represent_graph();
   BFS();
   printlevel();
   return 0;
}


void create_graph()
{
   int count,max_edge,origin,destin;

   printf("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,&destin);

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

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

```
        {           printf("Invalid edge!\n");
                    count--;
        }
        else
        {
            adj[origin][destin] = 1;
           // adj[destin][origin] = 1;
        }
    }
}


void represent_graph()
{
    printf("\nAdjacent matrix:\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            printf("%d ",adj[i][j] );
        printf("\n");

    }
    printf("\n");
}


void BFS()
{
    int i=0;
    for(i=0; i<n; i++)
    {
        visited[i] = 0;
        level[i]=-1;
    }

    int src, u;
    printf("Start Vertex: \n");
    scanf("%d", &src);

    enQueue(src);
    level[src]=0;
    while(!isEmpty_queue())
    {
        u= deQueue( );
        if(visited[u])
            continue;

        printf("%d->",u);
        visited[u] = 1;
```

```
        for(i=0; i<n; i++)
        {
            if(adj[u][i] == 1 && visited[i] == 0)
            {
                enQueue(i);
                level[i]=level[u]+1;
            }
        }
    }
    printf("\n");
}


void printlevel()
{
    printf("vertex Level\n");
    for(int i=0; i<n; i++)
        printf("%d\t%d\n", i,level[i]);
}


void enQueue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}


int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}



int deQueue()
{
    int delete_item;
```

```
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }

    delete_item = queue[front];
    front = front+1;
    return delete_item;
}
```

DFS

```
#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 DFS();
void create_graph();
void represent_graph();

int stack[MAX];
int top = -1;
void push(int v);
int pop();
int isEmpty_stack();


int main()
{
   create_graph();
   represent_graph();
   DFS();
}



void create_graph()
{
   int i,max_edges,origin,destin;

   printf("No. of nodes : ");
   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 edge!\n");
            i--;
        }
        else
            adj[origin][destin] = 1;

    }
}

void represent_graph()
{
    printf("\nAdjacent matrix:\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            printf("%d ",adj[i][j] );
        printf("\n");
    }
    printf("\n");
}


void DFS()
{
    int i;
    for(i=0; i<n; i++)
        state[i]=initial;

    printf("\nStarting node: ");
    int src,u;
    scanf("%d",&src);
    push(src);

    while(!isEmpty_stack())
    {
        u = pop();
        if(state[u]==initial)
        {
            printf("%d-> ",u);
            state[u]=visited;
        }
        for(i=n-1; i>=0; i--)
        {
            if(adj[u][i]==1 && state[i]==initial)
                push(i);
```

```
        }
    }
}


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()*/
```