```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 40


struct node
{
    int vertex;
    struct node *next;
};

typedef struct node node;

struct Graph
{
    int numVertices;
    struct node **adjLists;
    int *visited;
};

typedef struct Graph Graph;

struct queue
{
    int items[SIZE];
    int front;
    int rear;
};

typedef struct queue queue;

queue *createQueue()
{
    queue* q = malloc(sizeof(struct queue));
    q -> front = -1;
    q -> rear = -1;
    return q;
}

int isEmpty(queue *q)
{
    if(q -> rear == -1)
        return 1;
    else
        return 0;
}

void enqueue(queue *q, int value)
{
    if(q -> rear == SIZE - 1)
```

```
        printf("\nMemory overflow. Queue is full...\n");
    else
    {
        if(q -> front == -1)
            q -> front = 0;
        q -> rear++;
        q -> items[q -> rear] = value;
    }
}

int dequeue(queue *q)
{
    int item;

    if(isEmpty(q))
    {
        printf("\nQueue is empty\n");
        item = -1;
    }
    else
    {
        item = q -> items[q -> front];
        q -> front++;
        if(q -> front > q -> rear)
        {
            q -> front = q -> rear = -1;
        }
    }

    return item;
}

node *createNode(int v)
{
    node *newNode = (node *)malloc(sizeof(node));
    newNode -> vertex = v;
    newNode -> next = NULL;
    return newNode;
}


void addEdge(Graph* graph, int src, int dest)
{
    node *newNode = createNode(dest);
    newNode -> next = graph -> adjLists[src];
    graph -> adjLists[src] = newNode;

    newNode = createNode(src);
    newNode -> next = graph -> adjLists[dest];
    graph -> adjLists[dest] = newNode;
```

```
}

Graph *createGraph(int vertices, int edges)
{
    int i;
    int src,dest;
    Graph *graph =(Graph *) malloc(sizeof(Graph));
    graph -> numVertices = vertices;

    graph -> adjLists = malloc(vertices * sizeof(node*));
    graph -> visited = malloc(vertices * sizeof(int));

    for(i = 0; i < vertices; i++)
    {
        graph -> adjLists[i] = NULL;
        graph -> visited[i] = 0;
    }

    printf("\nEnter Edges...\n");
    printf("\n<source,destination> (Between 0 to %d)", vertices - 1);

    for(i = 0; i < edges; i++)
    {
        printf("\nEnter edge %d:", i+1);
        scanf("%d%d", &src,&dest);
        addEdge(graph,src,dest);
    }

    return graph;
}

void BFS(Graph *graph, int start)
{
    queue *q = createQueue();

    graph -> visited[start] = 1;
    enqueue(q, start);

    while(!isEmpty(q))
    {
        int currentVertex = dequeue(q);
        printf(" %d -> ", currentVertex);

        node *temp = graph -> adjLists[currentVertex];

        while(temp)
        {
            int adjVertex = temp -> vertex;

            if(graph -> visited[adjVertex] == 0)
```

```
            {
                graph -> visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp -> next;
        }
    }
}


void displayGraph(Graph *graph)
{
    int v;
    for(v = 0; v < graph -> numVertices; v++)
    {
        node *temp = graph -> adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while(temp)
        {
            printf("%d -> ", temp -> vertex);
            temp = temp -> next;
        }
        printf("\n");
    }
}

int main()
{
        Graph *graph = NULL;
        int nv, ne;
        int start = 0;
        int e = 1, ch;

        while( e )
        {
                printf( "\n--------------MENU--------------\n" );
                printf( "\n\t1. Create Graph\n\t2. Display\n\t3. Breadth First
Search (BFS) Algorithm\n\t4. Exit\n" );
                printf( "\n--------------------------------\n" );
                printf( "\n Enter your choice:" );
                scanf( "%d", &ch );

                switch( ch )
                {
                        case 1: printf("\nEnter number of verices and edges: ");
                                scanf("%d%d", &nv,&ne);
                                graph = createGraph(nv,ne);
                             break;
                        case 2: displayGraph(graph);
                                 break;
```

```
                        case 3: printf("\nSearched in the order (from the vertex
0): ");
                                BFS(graph,start);
                             break;
                        case 4 : e = 0;
                                 break;
                        default: printf( "\n Invalid choice \n" );
                }

        }
        return 0;
}
```