Lab - 9 :- BFS

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

#define MAX_SIZE 100

struct Queue
{
    int items[MAX_SIZE];
    int front;
    int rear;
};

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

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

void enqueue(struct Queue* queue, int value)
{
    if (queue->rear == MAX_SIZE-1)
        printf("\n Queue is FULL!! ");
    else
```

```
    {
        if (queue -> front == -1)
            queue -> front = 0;
        queue -> rear ++;
        queue -> items [queue -> rear] = value;
    }
}

int dequeue (struct Queue * queue)
{
    int item;
    if ( isEmpty (queue))
    {
        printf ("Queue is Empty");
        item = -1;
    }
    else
    {
        item = queue -> items [queue -> front];
        queue -> front ++;
        if (queue -> front > queue -> rear)
        {
            queue -> front = queue -> rear = -1;
        }
    }
    return item;
}

Struct Graph
{
    int vertices;
    int ** adjMatrix;
};
```

```
struct Graph* createGraph(int vertices)
{
    struct Graph *graph = (struct Graph*) malloc(sizeof
                                    (struct Graph));
graph->vertices = vertices;

graph->adjMatrix = (int **) malloc(vertices * sizeof(int*));
  for(int i=0; i<vertices; i++)
      {
          graph->adjMatrix[i] = (int *) malloc
                                    (vertices * sizeof(int));
    for(int j=0; j<vertices; j++)
        graph->adjMatrix[i][j] =0;
    }

    return graph;
}


void addEdge(struct Graph*graph, int src, int dest)
  {
        graph->adjMatrix[src][dest] = 1;
        graph->adjMatrix[dest][src] = 1;
  }


void BFS(struct Graph* graph, int startVertex)
  {
        int visited[MAX_SIZE] = {0};
        struct Queue* queue = createQueue();

        visited[startVertex] = 1;
        enqueue(queue, startVertex);
    printf("Breadth first Search Traversal: ");
```

```
while (!isEmpty(queue))
{
    int currentVertex = dequeue(queue);
    printf("%d", currentVertex);

    for(int i=0; i<graph->vertices; i++)
    printf("%d", currentVertex);

    for(int i=0; i<graph->vertices; i++)
    {
        if(graph->adjMatrix[currentVertex][i]==1
            && visited[i]==0)
        {
            visited[i]=1;
            enqueue(queue, i);
        }
    }
}
printf("\n");
}

int main()
{
    int vertices, edges, src, dest;
    printf("Enter the number of vertices:");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);
    printf("Enter the no of edges:");
    scanf("%d", &edges);
```

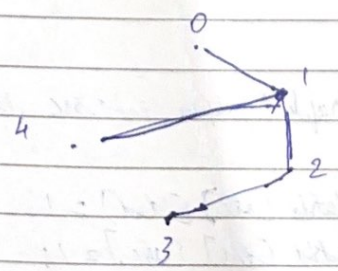
```
for (int i=0; i<edges; i++)
{
    printf ("Enter the edge %d (sour dest): ", i+1);
    scanf ("%d %d", &src, &dest);
    addEdge (graph, src, dest);
}

int startVertex;
printf ("Enter the starting vertex for BFS: ");
scanf ("%d", &startVertex);
BFS (graph, startVertex)
return 0;
}
```

Output :-

```
Enter the number of vertices : 5
Enter the no of edges : 4

Enter edg 1 (src, dest): 0 1
Enter edg 2 (src, dest) : 2 3
Enter edges (src, dest) : 1 4
Enter edg 4 (src, dest) : 1 2
Enter the starting vertex for BFS : 0
Breadth First Search Traversal : 0 1 2 4 3
```



DFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

Struct Graph
{
    int vertices;
    int ** adjMatrix;
};

Struct Graph* Create Graph (int vertices)
{
    struct Graph * graph = (struct Graph *) malloc (sizeof (
                                                struct Graph));
    graph -> vertices = vertices;

    graph -> adjMatrix = (int **) malloc ( vertices * sizeof (int));
    for(int i = 0; i < vertices; i++)
    {
        graph -> adjMatrix [i] = (int*) malloc (vertices * sizeof
                                                (int));
    for (int j = 0; j < vertices ; j++)
        graph -> adjMatrix [i] [j] = 0;
    }
    return graph;
}

void addEdge (struct Graph* graph, int src, int dest)
{
    graph -> adjMatrix [src] [dest] = 1;
    graph -> adjMatrix [dest] [src] = 1;
}
```

```
void DFS ( struct Graph * graph, int src, int dest)
{
    visited [start vertex] = 1;
    for (int i = 0; i < graph -> vertex; i++)
    {
        if (graph -> adjMatrix [start vertex] [i] == 1
                && visited [i] == 0)
            DFS (graph, i, visited);
    }
}

int isConnected (struct Graph * graph)
{
    int * visited = (int *) malloc (graph -> vertex * sizeof (int));
    for (int i = 0; i < graph -> vertex; i++)
        visted [i] = 0;

    DFS (graph, 0, visited);
    for ( int i = 0; i < graph -> vertex; i++)
    {
        if (visited [i] == 0)
            return 0;
    }
    return 1;
}
int main()
{
    int vertex, edges, src, dest;
    printf ("Enter the no of vertices:");
    scanf ("%d", & vertex);
```

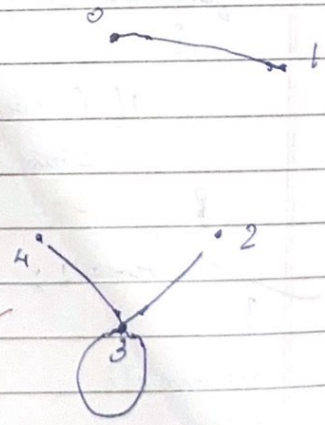
```
struct Graph* graph = createGraph (vertices);

printf ("Enter the no of edges : ");
scanf ("%d", &edges);

for (int i = 0; i < edges; i++)
{
    printf ("Enter edge %d (src dest) : ", i+1);
    scanf ("%d %d", &src, &dest);
    addEdge (graph, src, dest);
}
if (isConnected (graph))
    printf ("The graph is connected.\n");
else
    printf ("The graph is not connected.\n");
return 0;
}
```

Output :-

Enter the no of vertices : 5
Enter the no of edges : 4
Enter edge 1 (src dest) = 0 1
Enter edg 2 (src dest) = 2 3
Enter edg 3 (src dest) = 3 3
Enter edge 4 (src dest) = 3 4



The graph is not connected