22-2-24

Week - 09

1) Write a program to traverse a graph using BFS method

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

struct Queue {
    int items[MAX];
    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;
}
int isFull(struct Queue *queue) {
    if(queue -> rear == MAX-1)
        return 1;
    else
        return 0;
}
void enqueue(struct Queue * queue, int value) {
    if(isFull(queue))
        printf("\n Queue is Full ");
    else {
        if(queue -> front == -1)
            queue -> front = 0;
```

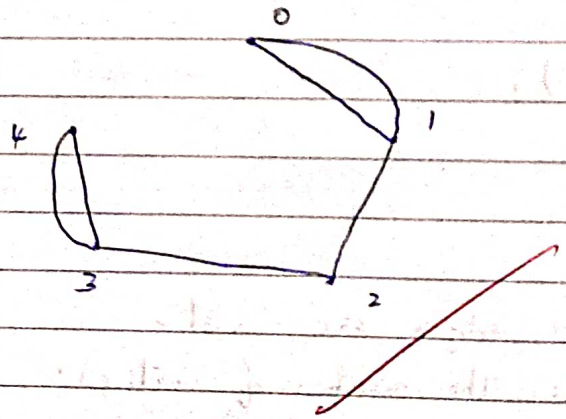
```
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

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] = {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 ++) {
            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 number of edges : ");
    scanf ("%d", &edges);
    for (int i=0; i< edges; i++) {
        printf (" Enter edge %d (source destination): ", 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 number of edges: 6
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destination): 3 4
Enter edge 6 (source destination): 4 4
Enter the starting vertex for BFS: 0
Breadth first search Traversal: 0 1 2 3 4
```



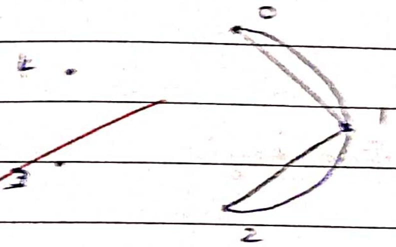
2) Write a program to check whether given graph is connected or not using DFS method.

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

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 DFS(struct Graph *graph, int startVertex, int visited[])
{

    visited[startVertex] = 1;
    for(int i=0; i<graph -> vertices; i++) {
        if(graph -> adjMatrix[startVertex][i] == 1 && visited[i]==0)
            DFS(graph, i, visited);
    }
}
```

```
int isConnected(struct Graph * graph)
{
    int * visited = (int*) malloc (graph -> vertices * sizeof(int));
    for (int i=0; i< graph -> vertices ; i++)
        visited[i] = 0;
    DFS (graph, 0, visited);
    for(int i = 0; i< graph -> vertices ; i++)
    {
        if (visited[i] == 0)
            return 0;
    }
    return 1;
}

int main()
{
    int vertices, edges, src, dest;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    struct Graph * graph = createGraph (vertices);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
    for(int i =0; i<edges ; i++)
    {
        printf("Enter edge %d (source destination): ", 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 number of vertices: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 1 2
The graph is not connected



22/2/24