LAB-9 DAA LAB

Shubham Singh
1906587

LE 8.1: WAP to Implement Breadth First Search.

Program:

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f=0, r=-1;
void bfs(int v) {
    for(i=1; i<=n; i++)
        if(a[v][i] && !visited[i])
            q[++r]=i;
    if(f<=r) {
     visited[q[f]]=1;
    }bfs(q[f++]);
}

void main() {
    int v;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    for(i=1; i<=n; i++) {
        q[i]=0;
        visited[i]=0;
    }
    printf("Enter Graph data: \n");
     for(i=1; i<=n; i++) {
         for(j=1; j<=n; j++) {
             scanf("%d", &a[i][j]);
         }
     }
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);
    Printf("The node which are reachable are: \n");
```

```
    for(i=1;i<=n;i++){
        if(visited[i])
          printf("%d \t", i);
        else
          printf("Bfs is not possible.\n");
          break;
    }
}
```

INPUT/OUTPUT:-

```
Enter number of vertices: 4
Enter graph data:
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1
Enter the string vertex: 1
The nodes which are reachable are:
1   2   3   4
```

1906587

LE 8.2:- WAP to implement Depth First Search.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int vertex;
    struct node* next;
};
struct node* createNode(int v);

struct Graph {
    int numVertices;
    int* visited;
    struct node** adjLists;
};
void DFS(struct Graph* graph, int vertex) {
    struct node* adjList = graph->adjLists[vertex];
    struct node* temp= adjList;
    graph->visited[vertex]=1;
    printf("Visited %d \n", vertex);
    while(temp!= NULL) {
        int connectedVertex = temp-> vertex;
        if (graph ->visited [connectedVertex] == 0){
            DFS(graph, connectedVertex);
        }
        temp= temp->next;
    }
}
struct node* createNode(int v) {
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}
```

1906587

```
struct Graph* createGraph(int vertices) {
  struct Graph * graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;
   graph->adjLists = malloc(vertices * sizeof(struct node*));
   graph->visited = malloc(vertices * sizeof(int));
   int i;
   for(i= 0; i<vertices; i++) {
       graph->adjLists[i]= NULL;
       graph->visited[i]= 0;
    }
   return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
    struct 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;
}

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

```
int main() {
    struct Graph* graph = createGraph(4);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
    printGraph(graph);
    DFS(graph, 2);
    return 0;
}
```

INPUT/OUTPUT :-

```
Adjacency list of vertex 0
2-> 1->
Adjacency list of vertex 1
2-> 0->
Adjacency list of vertex 2
3-> 1-> 0->
Adjacency list of vertex 3
2->
Visited 2
Visited 3
Visited 1
Visited 0
```