Ankit Raj | LAB-9 | 1906534

Q8) 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”, i);
    else
        printf(“Bfs is not possible.\n”);
        break;
    }
}
```

Input/output

```
Enter no. 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 node which are reachable are:
1 2 3 4.
```

Q) 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* adjLists = graph->adLists[Vertex];
     Struct node* temp = adjList;
};

Void DFS(Struct Graph* graph, int vertex){
     Struct node* adjList =graph->adjList[vertex];
     struct node* temp=adjList;
     graph->visited [vertex] =1;

   printf("visited %d \n", vertex);
   While(temp != NULL){
       int ConnectedVertex =temp->Vertex;
```

```
    if(graph -> visited [connected Vertex]==0){
        DFS(graph, connected Vertex);
    }
    temp = temp -> next;
}

Struct node* createNode(int v){
    Structnode* newNode = malloc (size of
                                  (Struct node));
    newNode -> vertex = v;
    newNode -> next = NULL;
    return newNode;
}

Struct Graph* CreateGraph(int Vertices){
   Struct Graph* graph = malloc(sizeOf(Struct Graph));
   graph -> numVertices = Vertices;
   graph -> adjLists = mallac(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;
   } returngraph;
}
```

```
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);
     return0;
 }
```

Input/output

```
Adjacency list Of Vertex 0
 2 -> 1 ->
                                        Visited 2
Adjacency list Of Vertex 1              Visited 3
 2 -> 0 ->                              Visited 1
Adjancency list Of Vertex 2             Visited 0.
  3 -> 1 -> 0 ->

Adjancency list of Vetex 3
   2 ->
```