Name: Devansh Upadhyay

Roll No: 53

Program:

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40
struct queue {
  int items[SIZE];
  int front;
  int rear;
};
struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);
struct node {
  int vertex;
  struct node* next;
};
struct node* createNode(int);
struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};
void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();
  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);
  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);
    struct node* temp = graph->adjLists[currentVertex];
    while (temp) {
      int adjVertex = temp->vertex;
      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
```

```
40      }}}
41  struct node* createNode(int v) {
42    struct node* newNode = malloc(sizeof(struct node));
43    newNode->vertex = v;
44    newNode->next = NULL;
45    return newNode;
46  }
47  struct Graph* createGraph(int vertices) {
48    struct Graph* graph = malloc(sizeof(struct Graph));
49    graph->numVertices = vertices;
50    graph->adjLists = malloc(vertices * sizeof(struct node*));
51    graph->visited = malloc(vertices * sizeof(int));
52    int i;
53    for (i = 0; i < vertices; i++) {
54      graph->adjLists[i] = NULL;
55      graph->visited[i] = 0;
56    }
57    return graph;
58  }
59  void addEdge(struct Graph* graph, int src, int dest) {
60    struct node* newNode = createNode(dest);
61    newNode->next = graph->adjLists[src];
62    graph->adjLists[src] = newNode;
63    newNode = createNode(src);
64    newNode->next = graph->adjLists[dest];
65    graph->adjLists[dest] = newNode;
66  }
67  struct queue* createQueue() {
68    struct queue* q = malloc(sizeof(struct queue));
69    q->front = -1;
70    q->rear = -1;
71    return q;
72  }
73  int isEmpty(struct queue* q) {
74    if (q->rear == -1)
```

```
    return 1;
  else
    return 0;
}
void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }}
int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }  }
  return item;
}
void printQueue(struct queue* q) {
  int i = q->front;
  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

PS D:\5th-Sem-Practicals\DAA\Pract 6> ./a.exe

Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3
PS D:\5th-Sem-Practicals\DAA\Pract 6>
```