K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: D2 **Roll. No.: 16010122323**

Experiment:6

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of Graph traversal menu driven program (DFS and BFS).

Objective: To understand graph as data structure and methods of traversing Graph

Expected Outcome of Experiment:

CO	Outcome
CO3	Demonstrate sorting and searching methods.

Websites/books referred:

Abstract: -

Definition of Graph, types of graphs, and difference and similarity between graph & tree)

Graph: A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as:

A Graph consists of a finite set of vertices (or nodes) and set of Edgeswhich connect a pair of nodes.

Types of Graph:

Digraph:
A graph G = (V, E) with a mapping f such that every edge maps onto some ordered pair of vertices (Vi, Vj) is called Digraph. It is also called Directed Graph. Ordered pair (Vi, Vj) means an edge between Vi and Vj with an arrow directed from Vito Vj.

Complete Graph:

A simple graph with n vertices is called a complete graph if the degree of eachvertex is n-1, that is, one vertex is attach with n-1 edges. A complete graph is also called Full Graph.

Finite Graphs:

A graph is said to be finite if it has finite number of vertices and finite numberof edges.

Infinite Graph:

A graph is said to be infinite if it has infinite number of vertices as well asinfinite number of edges.

Trivial Graph:

A graph is said to be trivial if a finite graph contains only one vertex and no edge.

Difference between graph and tree

No.	Graph	Tree
1	Graph is a non-linear data structure.	Tree is a non-linear data structure.
2	It is a collection of vertices/nodes and edges.	It is a collection of nodes and edges.
3	Each node can have any number of edges.	General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes.
4	There is no unique node called root in graph.	There is a unique node called root in trees.
5	A cycle can be formed.	There will not be any cycle.

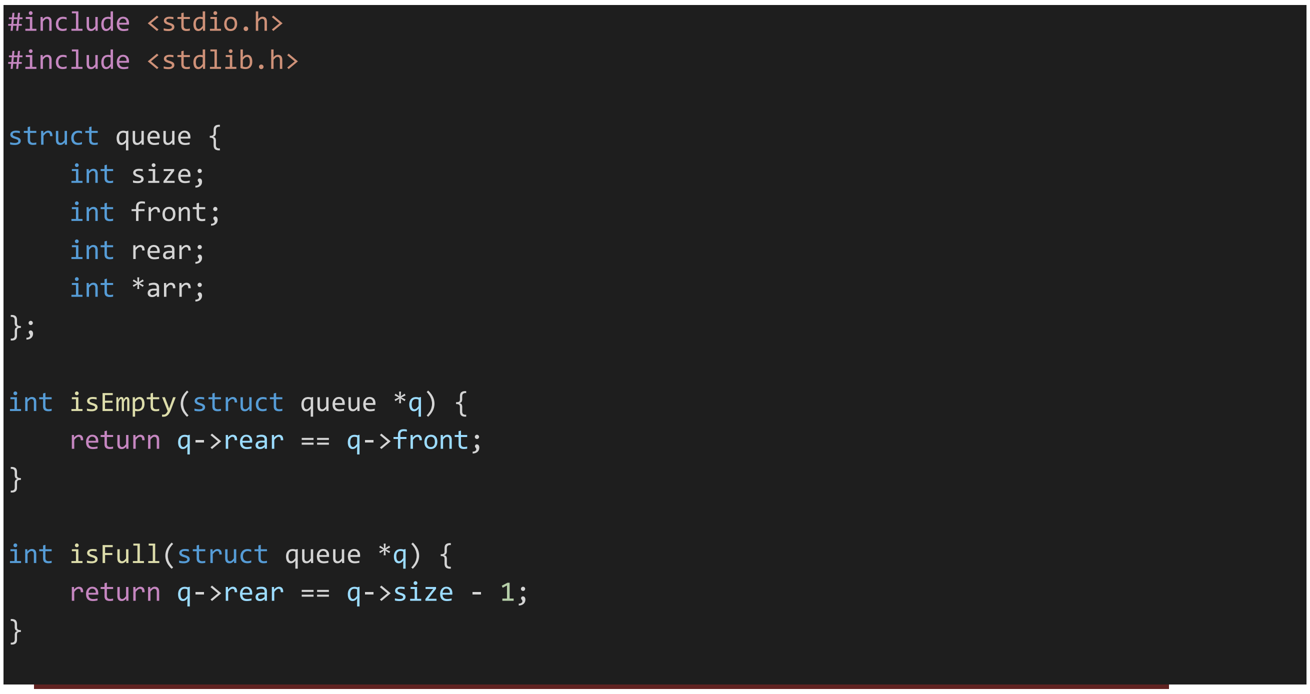
Similarity between graph and tree:

Graph and tree are the non-linear data structure which is used to solve various complex problems. A graph is a group of vertices and edges where an edge connects a pair of vertices whereas a tree is considered as a minimally connected graph which must be connected and free from loops.

Algorithm for DFS/BFS:

```
Step 1: Set Status = 1(ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3(processed state).

Step 5: Traverse all the neighbours of N that are in the ready state
(STATUS = 1) andset the STATUS = 2 (waiting state)

(END OF LOOP)

Step 6: STOP
```

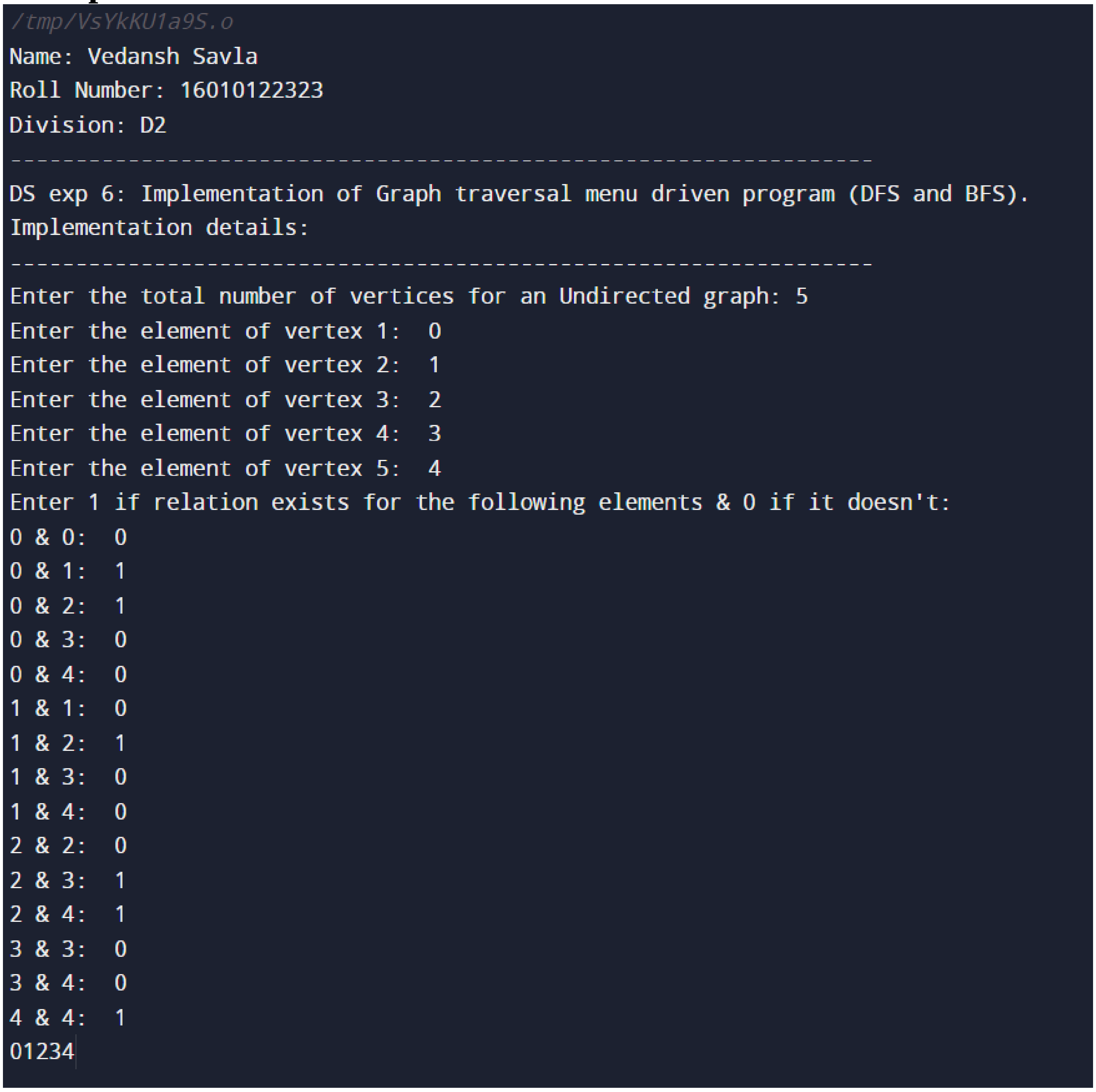
Code and output screenshots:



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

struct queue {
    int size;
    int front;
    int rear;
    int *arr;
};

int isEmpty(struct queue *q) {
    return q->rear == q->front;
}

int isFull(struct queue *q) {
    return q->rear == q->size - 1;
}
```

```c
void enqueue(struct queue *q, int val) {
    if (isFull(q)) {
        printf("This Queue is Full\n");
    } else {
        q->rear++;
        q->arr[q->rear] = val;
    }
}

int dequeue(struct queue *q) {
    int a = -1;
    if (isEmpty(q)) {
        printf("The Queue is Empty\n");
    } else {
        q->front++;
        a = q->arr[q->front];
    }
    return a;
}

int main() {
    int ver, graph[100], rel[100][100], visited[100] = {0};

    printf("Name: Vedansh Savla\n");
    printf("Roll Number: 16010122323\nDivision: D2\n");
    printf("------------------------------------------------------------------\n");
    printf("DS exp 6: Implementation of Graph traversal menu driven program (DFS and
BFS).\n");
    printf("Implementation details:\n");
    printf("------------------------------------------------------------------\n");
    printf("Enter the total number of vertices for an Undirected graph:\t");
    scanf("%d", &ver);

    // For taking input of elements present at vertices
    for (int i = 0; i < ver; i++) {
        printf("Enter the element of vertex %d:\t", i + 1);
        scanf("%d", &graph[i]);
    }

    // For taking input of Relation matrix for Undirected graph
    printf("Enter 1 if relation exists for the following elements & 0 if it
doesn't:\n");
    for (int i = 0; i < ver; i++) {
        for (int j = i; j < ver; j++) {
            printf("%d & %d:\t", graph[i], graph[j]);
            scanf("%d", &rel[i][j]);
            rel[j][i] = rel[i][j];
        }
        printf("\n");
    }

    // Initialization of the queue
    struct queue q;
    q.size = 200;
    q.front = q.rear = -1;
    q.arr = (int *)malloc(q.size * sizeof(int));

    int i = 0;
    printf("%d", i);
    visited[i] = 1;
    enqueue(&q, i);

    while (!isEmpty(&q)) {
        int node = dequeue(&q);
        for (int j = 0; j < ver; j++) {
            if (rel[node][j] == 1 && visited[j] == 0) {
                printf("%d", j);
                visited[j] = 1;
                enqueue(&q, j);
            }
        }
    }

    return 0;
}
```

Output:



```
/tmp/VsYkKU1a9S.o
Name: Vedansh Savla
Roll Number: 16010122323
Division: D2
------------------------------------------------------------------
DS exp 6: Implementation of Graph traversal menu driven program (DFS and BFS).
Implementation details:
------------------------------------------------------------------
Enter the total number of vertices for an Undirected graph: 5
Enter the element of vertex 1:  0
Enter the element of vertex 2:  1
Enter the element of vertex 3:  2
Enter the element of vertex 4:  3
Enter the element of vertex 5:  4
Enter 1 if relation exists for the following elements & 0 if it doesn't:
0 & 0:  0
0 & 1:  1
0 & 2:  1
0 & 3:  0
0 & 4:  0
1 & 1:  0
1 & 2:  1
1 & 3:  0
1 & 4:  0
2 & 2:  0
2 & 3:  1
2 & 4:  1
3 & 3:  0
3 & 4:  0
4 & 4:  1
01234
```

Implementation Details:

1. Enlist all the Steps followed and various options explored, explain your program logic, classes and methods used.

The program then declares an empty queue and enqueues the 1st element. It then explores all the adjacent vertex and enqueues all of them into queue. Then it dequeues an element and visits all its adjacent nodes. It prints the dequeued element and as a result, we get out breadth first search output.

2. Explain the Importance of the approach followed by you

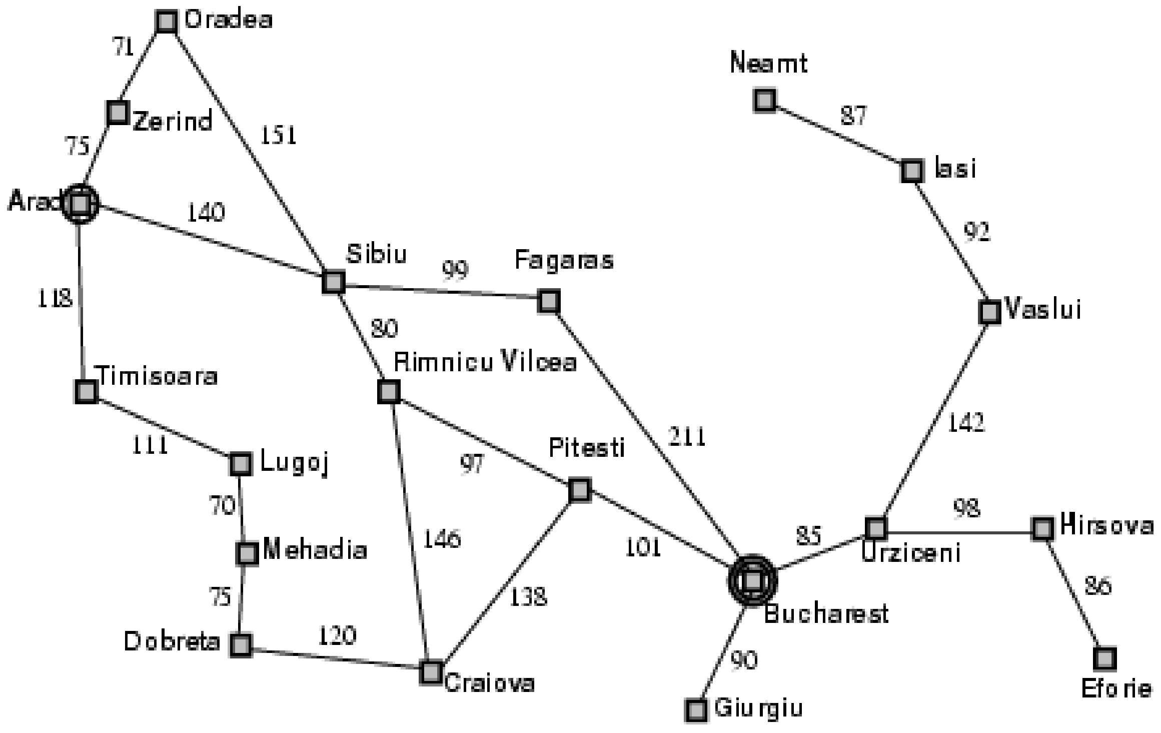
The program I created is versatile in nature and can have any number of Vertices in the graph. The user enters the element present in each of the vertex and also, we input an adjacency matrix form the user to indicate the relation between the vertices.

Post lab questions-

a. **Differentiate between BFS and DFS.**

BFS	DFS
BFS stands for Breadth First Search.	DFS stands for Depth First Search.
It a vertex-based technique to find the shortest path in a graph.	It is an edge-based technique because the vertices along the edge are explored first from the starting to the end node.
BFS is a traversal technique in which all the nodes of the same level are explored first, and then we move to the next level.	DFS is also a traversal technique in which traversal is started from the root node and explore the nodes as far as possible until we reach the node that hasno unvisited adjacent nodes.
Queue data structure is used for the BFS traversal.	Stack data structure is used for the BFS traversal.
BFS does not use the backtracking concept.	DFS uses backtracking to traverse all the unvisited nodes.
BFS finds the shortest path having a minimum number of edges to traverse from the source to the destination vertex.	In DFS, a greater number of edges are required to traverse from the source vertex to the destination vertex.

b. **Give sequence of the nodes visited as per BFS and DFS strategy for following example. Source- Arad, Destination- Bucharest (Traversal would stop after destination is reached)**



BFS is as below:

1. Arad
2. Zerind
3. Sibiu
4. Timisoara
5. Oradea
6. Fagaras
7. Rimnicu Vilcea
8. Lugoj
9. Bucharest.

DFS is as below:

1. Arad
2. Timisoara
3. Lugoj
4. Mehadia
5. Dobreta
6. Craiova
7. Pitesti
8. Bucharest

Conclusion: -

Hence in the conclusion we can say that the title of our experiment that is travelling algorithms in graphs were studied,completely understood and implemented well in our experiment