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

Batch:	**B2**	**Roll. No.: 100**
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:

https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

https://www.upgrad.com/blog/graphs-in-data-structure/

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 vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph.

Types of Graphs

1) **Weighted Graph.**
2) **Unweighted Graph.**
3) **Undirected Graph.**
4) **Directed Graph.**

Difference between graph and tree

A graph is a set of vertices/nodes and edges. A tree is a set of nodes and edges. In the graph, there is no unique node which is known as root. In a tree, there is a unique node which is known as root.

Similarity between graph and tree:

Both are non linear

Both consists of interconnection between data (nodes)

.

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: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

Code and output screenshots:

```
#include <iostream>

using namespace std;

//for graph of fixed size 6
int len=6;
int ctr=0;
int visited[6]={-1,-1,-1,-1,-1,-1};
int adjustancy[6][6];//{{0,1,1,1,1},{1,0,0,0,0},{1,0,0,0,0},{1,0,0,0,0},{1,0,0,0,0}} ;
bool searchinvisited(int j){
   for(int i=0;i<6;i++){
    if(visited[i]==j){
      // cout<<j;
       return true;
    }
   }
   return false;
}
void visitedadd(int j){
   visited[ctr]=j;
   ctr++;
}
int bfs( int value){
   visitedadd(value);
```

```
   for(int i=0;i<6;i++){

      if(adjustancy[value][i]==1){

       //  cout<<(searchinvisited(i));

         if(!searchinvisited(i)){ //element not visited already


            bfs(i);

         }

      }
}
cout<<value;
return 0;


}
int main()
{   cout<<"Hello World \n";


int ans[6];




   for(int i=0;i<6;i++){
      for(int j=0;j<6;j++){
      cout<<"Join "<<i<<" and "<<j<<"? \n";
      cin>>adjustancy[i][j];
   }
   }
```

```
    bfs(2);

    return 0;
}
```

Hello World

Join 0 and 0?

0

Join 0 and 1?

1

Join 0 and 2?

1

Join 0 and 3?

1

Join 0 and 4?

1

Join 0 and 5?

1

Join 1 and 0?

1

Join 1 and 1?

0

Join 1 and 2?

0

Join 1 and 3?

0

Join 1 and 4?

0

Join 1 and 5?

0

Join 2 and 0?

1

Join 2 and 1?

0

Join 2 and 2?

0

Join 2 and 3?

0

Join 2 and 4?

0

Join 2 and 5?

0

Join 3 and 0?

1

Join 3 and 1?

0

Join 3 and 2?

0

Join 3 and 3?

0

Join 3 and 4?

0

Join 3 and 5?

0

Join 4 and 0?

1

Join 4 and 1?

0

Join 4 and 2?

0

Join 4 and 3?

0

Join 4 and 4?

0

Join 4 and 5?

0

Join 5 and 0?

1

Join 5 and 1?

0

Join 5 and 2?

0

Join 5 and 3?

0

Join 5 and 4?

0

Join 5 and 5?

0

134502

Implementation Details:

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

Adjustancy matrix used for implementation of graphs. Recursion is not used here.

2. Explain the Importance of the approach followed by you

Array implemplementation used for fixed sides of number of nodes instead of linked list dynamic implementation. This approach is for limited number of nodes in graph.

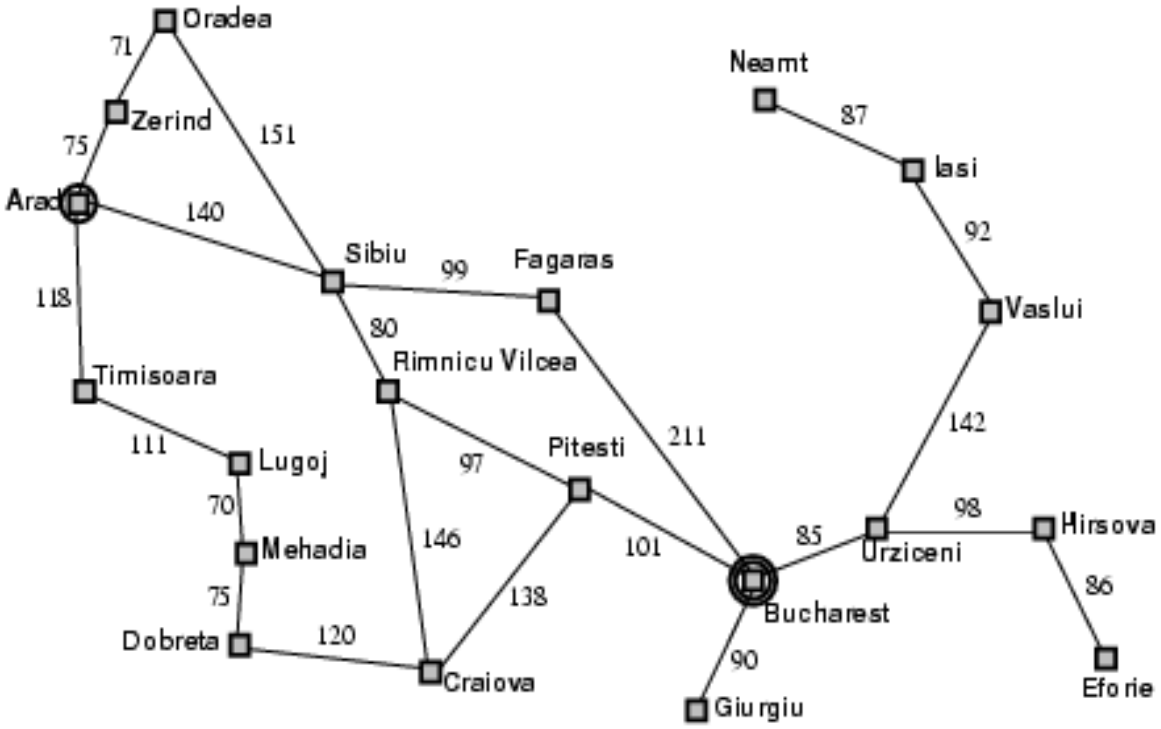
Post lab questions-

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

BFS	DFS
BFS finds the shortest path to the destination.	DFS goes to the bottom of a subtree, then backtracks.
The full form of BFS is Breadth-First Search.	The full form of DFS is Depth First Search.
It uses a queue to keep track of the next location to visit.	It uses a stack to keep track of the next location to visit.
BFS traverses according to tree level.	DFS traverses according to tree depth.
It is implemented using FIFO list.	It is implemented using LIFO list.
It requires more memory as compare to DFS.	It requires less memory as compare to BFS.
This algorithm gives the shallowest path solution.	This algorithm doesn't guarantee the shallowest path solution.
There is no need of backtracking in BFS.	There is a need of backtracking in DFS.
You can never be trapped into finite loops.	You can be trapped into infinite loops.
If you do not find any goal, you may need to expand many nodes before the solution is found.	If you do not find any goal, the leaf node backtracking may occur.

Ref - https://www.guru99.com/difference-between-bfs-and-dfs.html

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 - 118,140 75 111 80 99 71,
70,146,97,211,151,75,120,138,101,90,85,98,142,86,92,87

DFS- 118, 111,70,75,120,146,80,140,151,71,75,99,211,101,97,90,85,98,80,142,92,87

Conclusion: -

Thus we have understood graphs and BFS and DFS traversal of graph. We implemented the graph traversal and learnt about adjustancy implementation of DFS.