K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 7

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

Title: Graph Traversal Techniques

Objective: Implementation of Graph traversal techniques: DFS and BFS.

Expected Outcome of Experiment:

CO	Outcome
CO3	Demonstrate sorting and searching methods

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:

DATA STRUCTURE:

1. int adj[10][10] //matrix of size 10x10 to store adjacency matrix
2. int visited[10] //Boolean array to check if node visited or not
3. int q[10] //queue of size 10 to traverse using BFS
4. front=rear=0

OPERATIONS:

1. **void DFS()**

   Used for traversing a graph using Depth First Search.

2. **void BFS()**

   Used for traversing a graph using Breadth First Search.

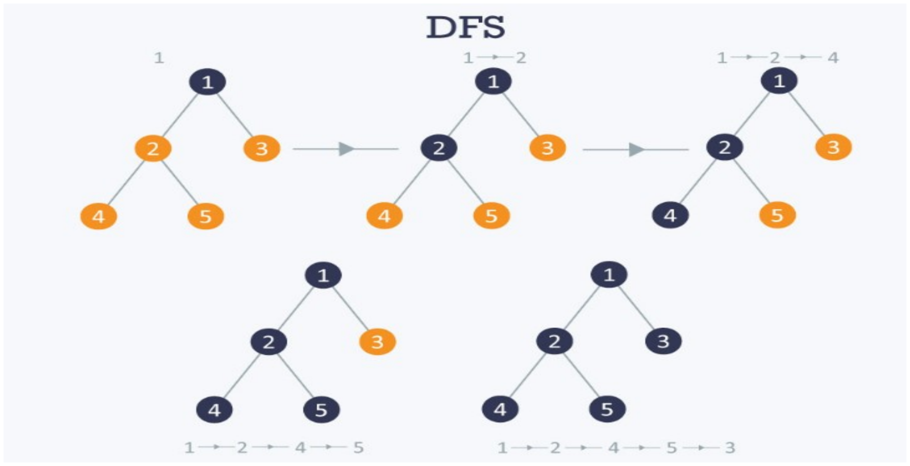
Related Theory: -

Depth First Search:

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:

1. Pick a starting node and push all its adjacent nodes into a stack.
2. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
3. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.



Time complexity O(V+E), when implemented using an adjacency list.

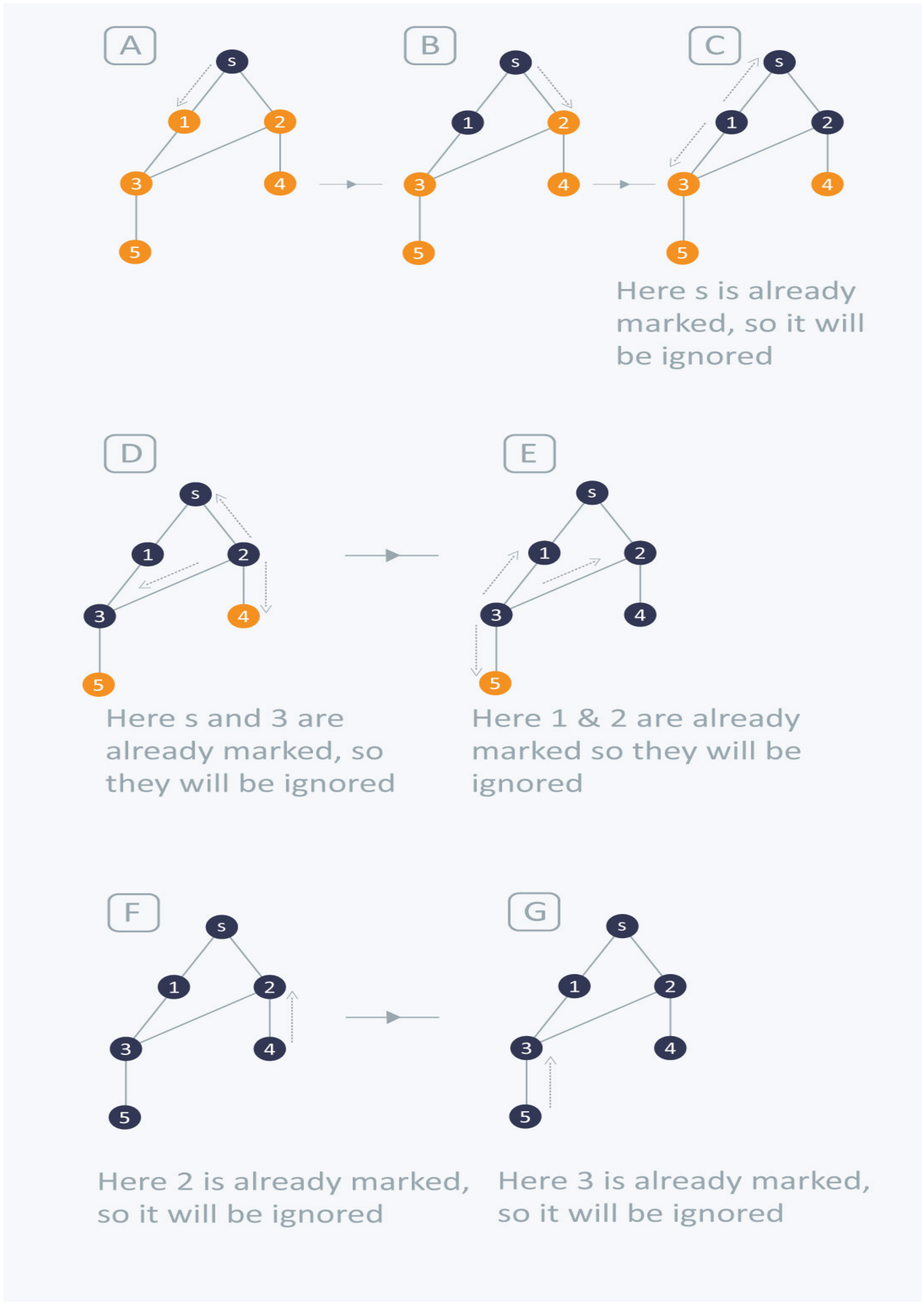
Breadth First Search:

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer
3. A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.
4. In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.
5. To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neigbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

   Complexity: The time complexity of BFS is **O(V + E)**, where V is the number of nodes and E is the number of edges.



Here s is already marked, so it will be ignored

Here s and 3 are already marked, so they will be ignored

Here 1 & 2 are already marked so they will be ignored

Here 2 is already marked, so it will be ignored

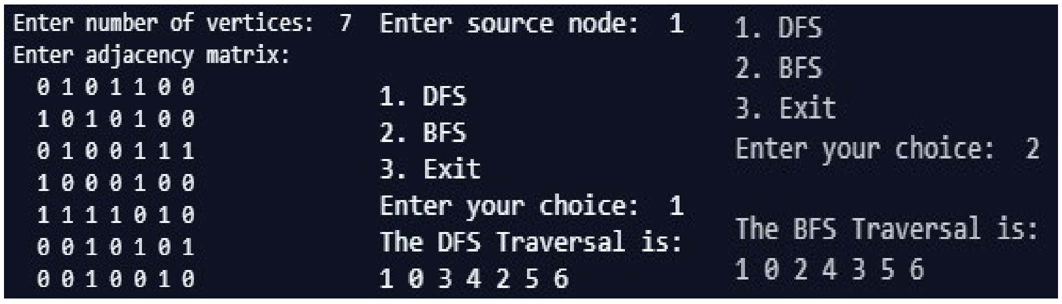
Here 3 is already marked, so it will be ignored

Implementation Details:

```
#include <stdio.h>
#include <stdlib.h>
int adj[10][10],visited[10], q[10], n;
int front = 0;
int rear = 0;
void DFS(int);
void BFS(int);

void main() {
    int i,j, source, choice;
    printf("Enter number of vertices: ");
    scanf("%d",&n);
    printf("Enter adjacency matrix: \n");
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
            scanf("%d",&adj[i][j]);
   for(i=0;i<n;i++)
        visited[i]=0;
    printf("Enter source node: ");
    scanf("%d", &source);
    do{
      printf("\n1. DFS\n2. BFS\n3. Exit\nEnter your choice: ");
      scanf("%d", &choice);
      switch(choice){
        case 1:
        printf("The DFS Traversal is:\n");
        DFS(source);
        break;
        case 2:
        printf("\nThe BFS Traversal is:\n");
        for(i=0;i<n;i++)
          visited[i]=0;
          BFS(source);
          break;
          case 3: exit(1);
      }
    }while(choice!=3);
}

void DFS(int i){
    int j;
    printf("%d ",i);
    visited[i]=1;
    for(j=0;j<n;j++)
       if(!visited[j] && adj[i][j]==1)
            DFS(j);
}

void BFS (int v){
  visited[v] = 1;
  q[rear] = v;
  rear++;
  while (rear != front){
    int i;
    int u = q[front];
    printf ("%d ", u);
    front++;
    for (i = 0; i < n; i++){
      if (!visited[i] && adj[u][i]){
        q[rear] = i;
        rear++;
        visited[i] = 1;
      }
    }
  }
  printf ("\n");
}
```

For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/DFS-BFS

OUTPUT SCREEN:



```
Enter number of vertices:  7
Enter adjacency matrix:
  0 1 0 1 1 0 0
  1 0 1 0 1 0 0
  0 1 0 0 1 1 1
  1 0 0 0 1 0 0
  1 1 1 1 0 1 0
  0 0 1 0 1 0 1
  0 0 1 0 0 1 0
```

```
Enter source node:  1

1. DFS
2. BFS
3. Exit
Enter your choice:  1
The DFS Traversal is:
1 0 3 4 2 5 6
```

```
1. DFS
2. BFS
3. Exit
Enter your choice:  2

The BFS Traversal is:
1 0 2 4 3 5 6
```

CONCLUSION:

The program ran successfully as we were able to implement menu driven program for DFS and BFS traversal of a graph.