AIM: Implementation of Graph traversal. (DFS and BFS)

THEORY:

A) Depth First Search (DFS):

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

Depth First Search Algorithm

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

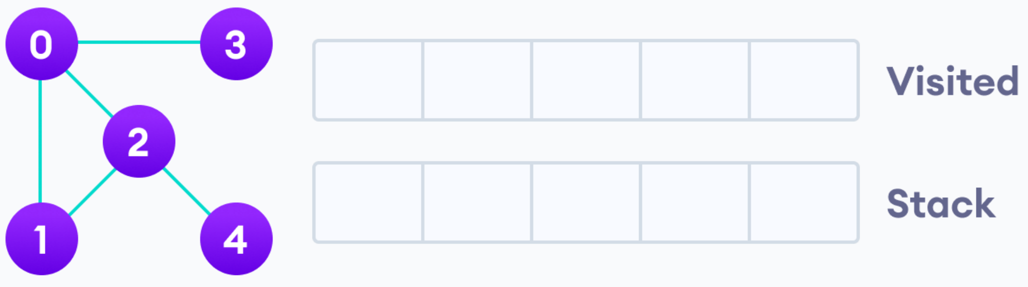
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

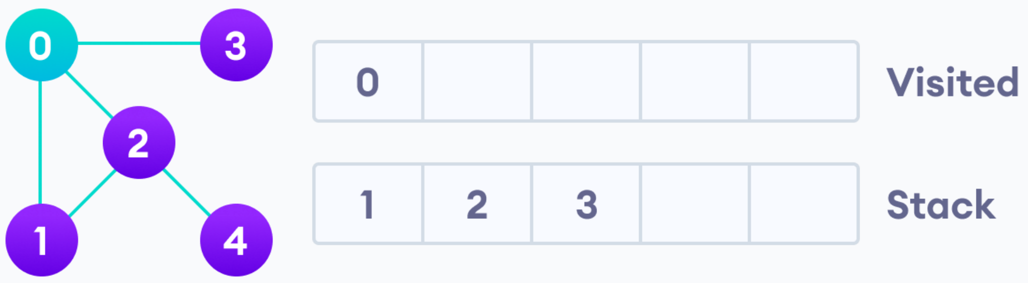
Depth First Search Example

Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



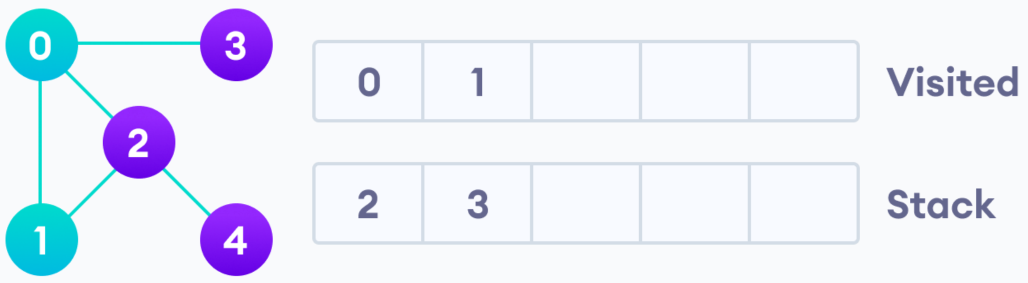
Undirected graph with 5 vertices

We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



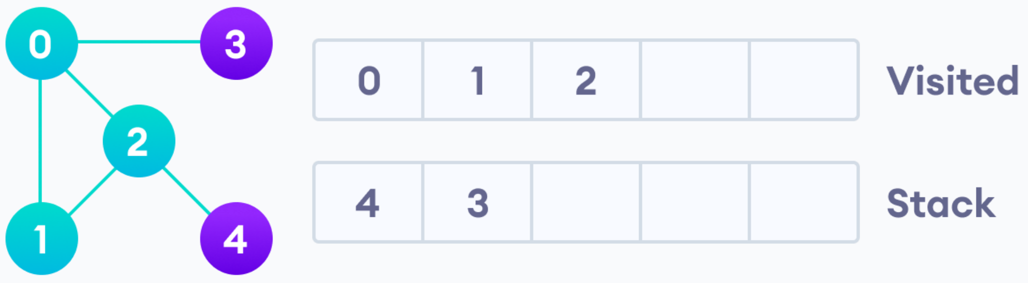
Visit the element and put it in the visited list

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

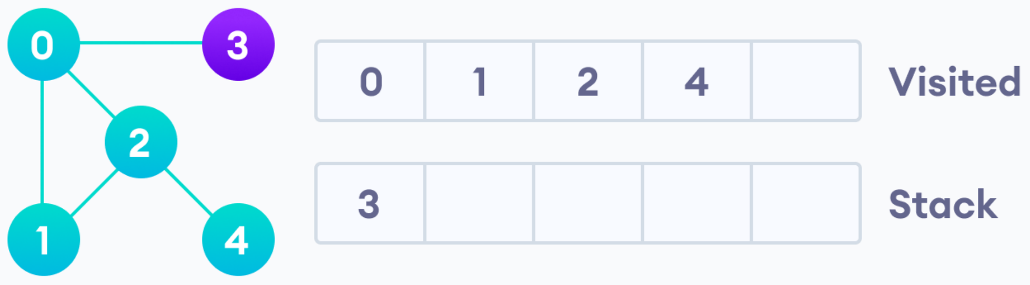


Visit the element at the top of stack

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



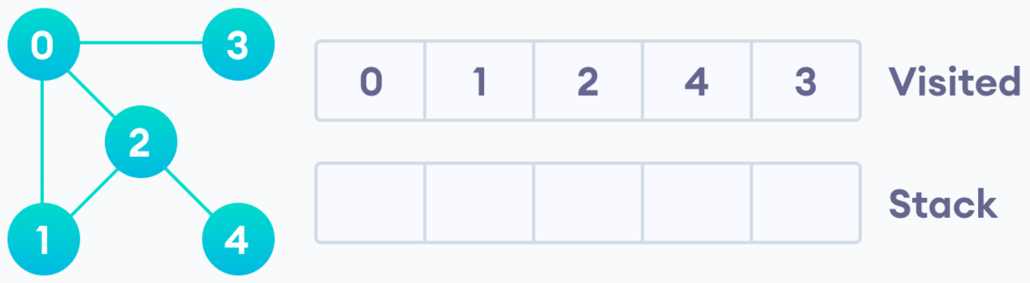
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have

completed the Depth First Traversal of the graph.



After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.

Complexity of Depth First Search

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is O(V).

Application of DFS Algorithm

1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph

B) Breadth First Search (BFS):

Traversal means visiting all the nodes of a graph. Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

BFS algorithm

A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

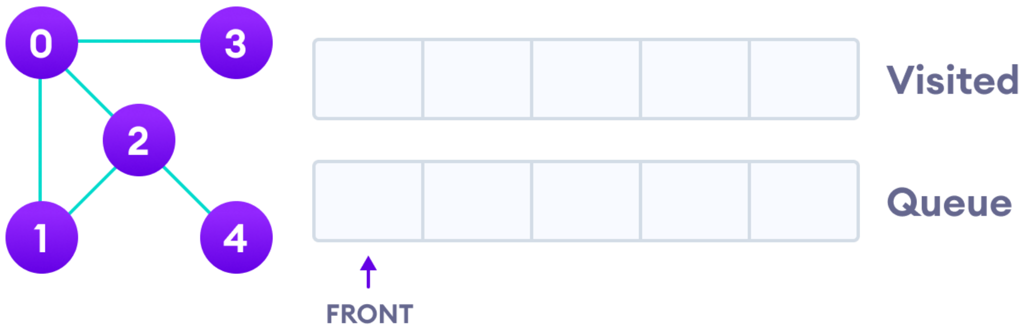
The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

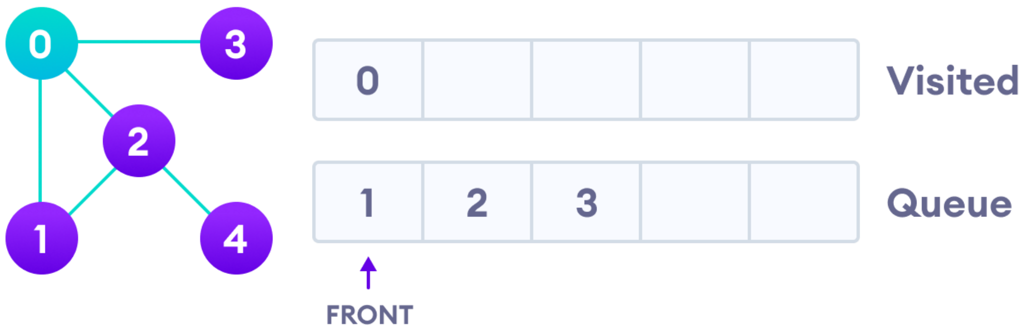
BFS example

Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



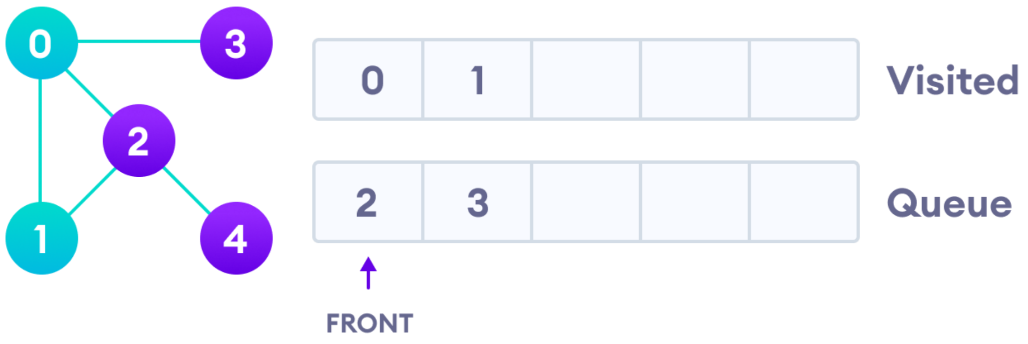
Undirected graph with 5 vertices

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



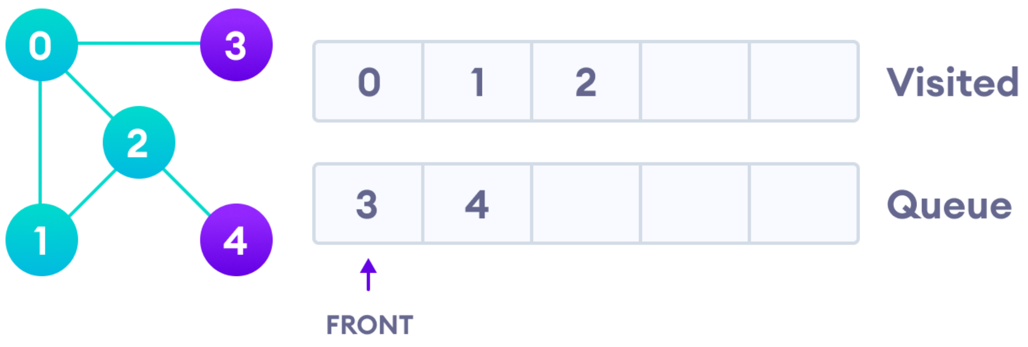
Visit start vertex and add its adjacent vertices to queue

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

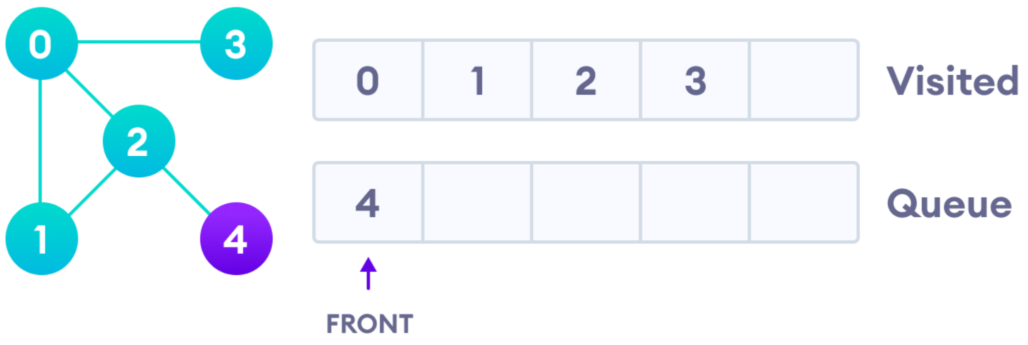


Visit the first neighbour of start node 0, which is 1

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.

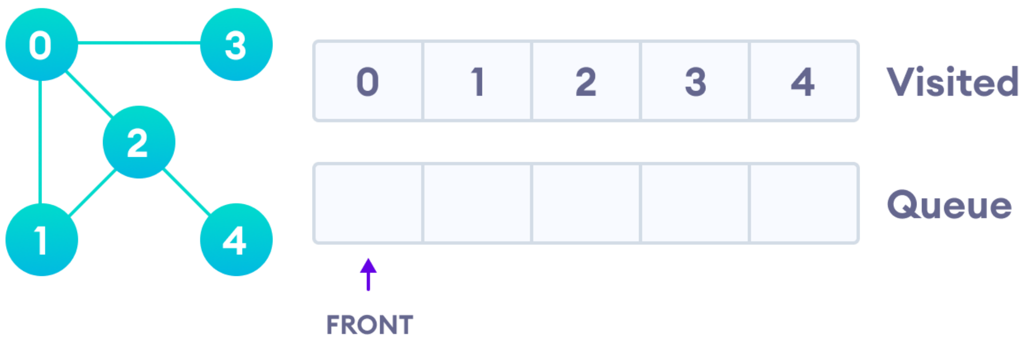


Visit 2 which was added to queue earlier to add its neighbours



4 remains in the queue

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.



Visit last remaining item in the queue to check if it has unvisited neighbours

Since the queue is empty, we have completed the Breadth First Traversal of the graph.

BFS Algorithm Complexity

The time complexity of the BFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.
The space complexity of the algorithm is O(V).

BFS Algorithm Applications

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph
6. In minimum spanning tree

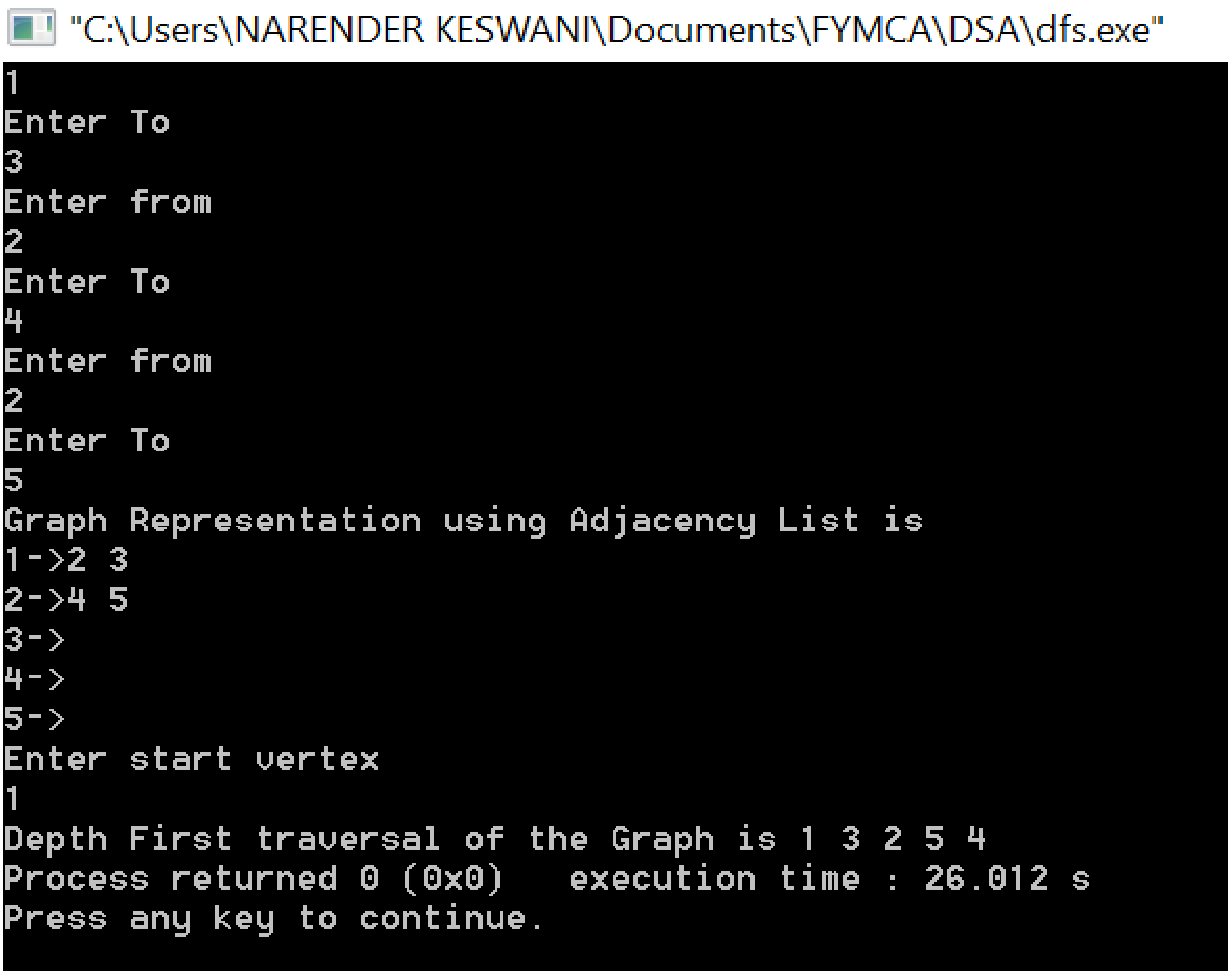
A) Depth First Search (DFS):

SOURCE CODE:

```
#include<iostream>
#include<vector>
#include<stack>
using namespace std;
void addEdge(int u,int v,vector<int>* V)
{
	V[u].push_back(v);
}
void DFS(int s,vector<int>* adj,int n)
{
   int visited[n+1]={0};
   stack<int> stack;
   stack.push(s);
   vector<int>::iterator i;
   while (!stack.empty())
   {
      s = stack.top();
      stack.pop();

      if (!visited[s])
      {
         cout << s << " ";
         visited[s] = true;
      }
      for (i = adj[s].begin(); i != adj[s].end(); ++i)
         if (!visited[*i])
            stack.push(*i);
   }
}
int main()
{
	int n,e,u,v,start;
	cout<<"Enter no of vertices"<<endl;
	cin>>n;
	cout<<"Enter no of Edges"<<endl;
	cin>>e;
	int copy=n;
	vector<int> V[n+1];
	for(int i=0;i<e;i++)
	{
		cout<<"Enter from"<<endl;
		cin>>u;
		cout<<"Enter To"<<endl;
		cin>>v;
		addEdge(u,v,V);
	}
	cout<<"Graph Representation using Adjacency List is"<<endl;
	vector<int>::iterator it;
```

```
        for(int i=1;i<=n;i++)
        {
                cout<<i<<"->";
                for(it=V[i].begin();it!=V[i].end();it++)
                {
                        cout<<*it<<" ";
                }
                cout<<endl;
        }
        cout<<"Enter start vertex"<<endl;
        cin>>start;
        cout<<"Depth First traversal of the Graph is ";
        DFS(start,V,n);
        return 0;
}
```

OUTPUT:



```
"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\dfs.exe"
1
Enter To
3
Enter from
2
Enter To
4
Enter from
2
Enter To
5
Graph Representation using Adjacency List is
1->2 3
2->4 5
3->
4->
5->
Enter start vertex
1
Depth First traversal of the Graph is 1 3 2 5 4
Process returned 0 (0x0)   execution time : 26.012 s
Press any key to continue.
```

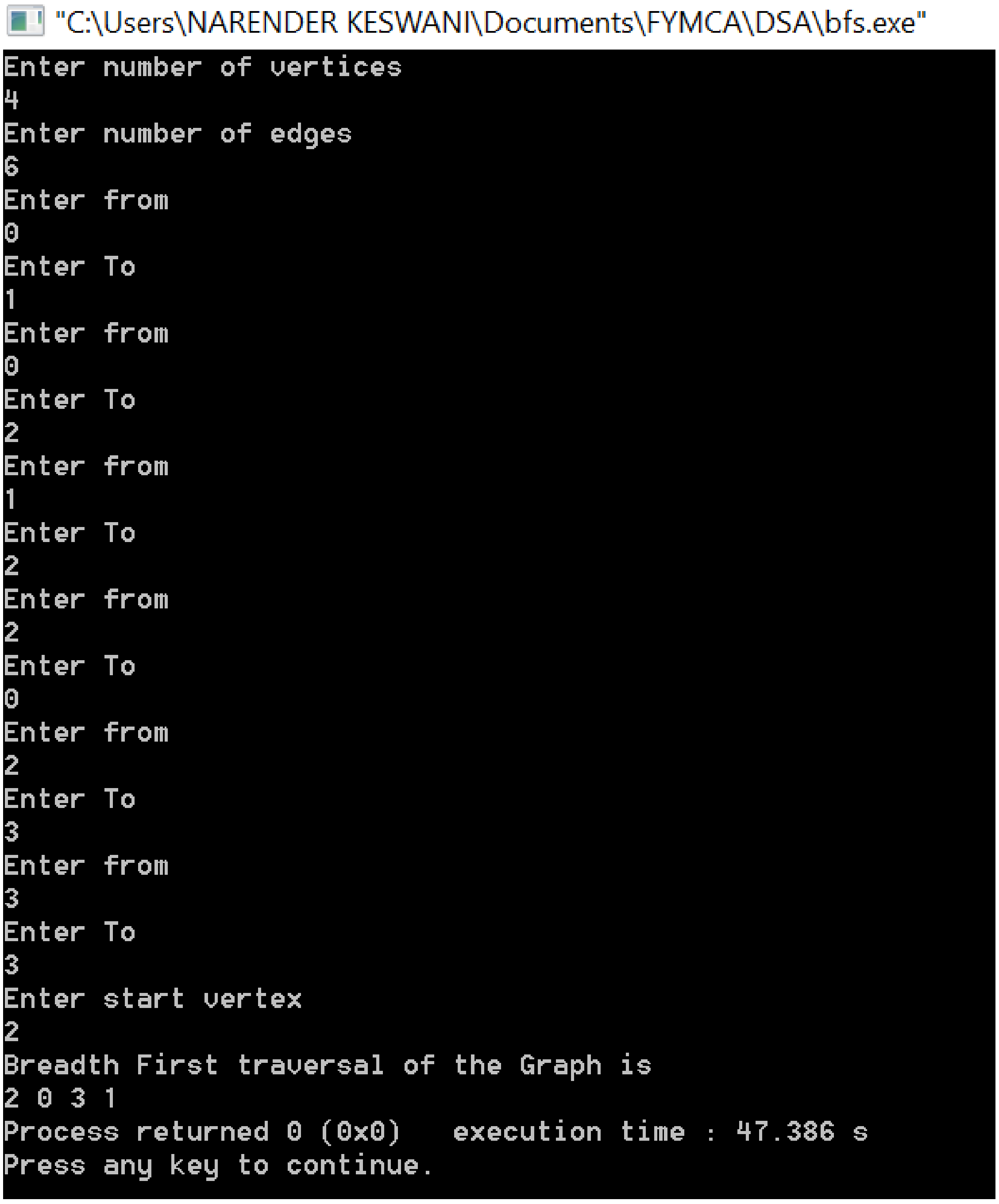
B) Breadth First Search (BFS):

SOURCE CODE:

```
#include<iostream>
#include <list>
using namespace std;

class Graph
{
    int V;
    list<int> *adj;
public:
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

void Graph::BFS(int s)
{
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
            visited[i] = false;

    list<int> queue;
    visited[s] = true;
    queue.push_back(s);
    list<int>::iterator i;

    while(!queue.empty())
    {
            s = queue.front();
            cout << s << " ";
            queue.pop_front();

            for (i = adj[s].begin(); i != adj[s].end(); ++i)
            {
                    if (!visited[*i])
                    {
                            visited[*i] = true;
```

```
                        queue.push_back(*i);
                    }
                }
            }
        }

        int main()
        {
          int vt,e,u,v,start;
          cout<<"Enter number of vertices"<<endl;
           cin>>vt;
           Graph g(vt);
          cout<<"Enter number of edges"<<endl;
           cin>>e;
           for(int i=0;i<e;i++)
           {
                    cout<<"Enter from"<<endl;
                    cin>>u;
                    cout<<"Enter To"<<endl;
                    cin>>v;
                    g.addEdge(u,v);
           }
           cout<<"Enter start vertex"<<endl;
           cin>>start;
           cout<<"Breadth First traversal of the Graph is "<<endl;
           g.BFS(start);
           return 0;
        }
```

OUTPUT:



"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\bfs.exe"

```
Enter number of vertices
4
Enter number of edges
6
Enter from
0
Enter To
1
Enter from
0
Enter To
2
Enter from
1
Enter To
2
Enter from
2
Enter To
0
Enter from
2
Enter To
3
Enter from
3
Enter To
3
Enter start vertex
2
Breadth First traversal of the Graph is
2 0 3 1
Process returned 0 (0x0)   execution time : 47.386 s
Press any key to continue.
```

CONCLUSION:

From this practical, I have learned how to implement the DFS & BFS in c++.