 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

Aim: Implementing the Graph Traversal Approach

IDE: Visual Studio Code

1. **Implement the Breadth-First Search**

Theory: -

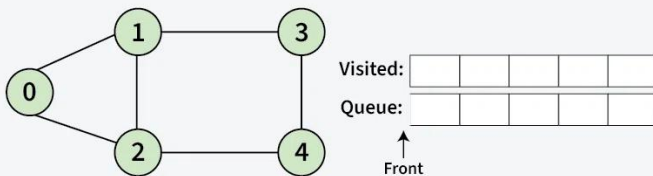
- Breadth First Search (BFS) is a fundamental graph traversal algorithm. It begins with a node, then first traverses all its adjacent. Once all adjacent are visited, then their adjacent are traversed. This is different from DFS in a way that closest vertices are visited before others. We mainly traverse vertices level by level. A lot of popular graph algorithms like Dijkstra's shortest path, Kahn's Algorithm, and Prim's algorithm are based on BFS. BFS itself can be used to detect cycle in a directed and undirected graph, find shortest path in an unweighted graph and many more problems.

BFS from a Given Source:

- The algorithm starts from a given source and explores all reachable vertices from the given source. It is similar to the Breadth-First Traversal of a tree. Like tree, we begin with the given source (in tree, we begin with root) and traverse vertices level by level using a queue data structure. The only catch here is that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.
- **Initialization:** Enqueue the given source vertex into a queue and mark it as visited.
- 1. **Exploration:** While the queue is not empty:
 - Dequeue a node from the queue and visit it (e.g., print its value).
 - For each unvisited neighbor of the dequeued node:
 - Enqueue the neighbor into the queue.
 - Mark the neighbor as visited.
- 2. **Termination:** Repeat step 2 until the queue is empty.
- This algorithm ensures that all nodes in the graph are visited in a breadth-first manner, starting from the starting node.

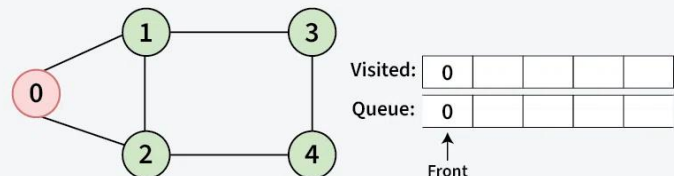
How Does the BFS Algorithm Work :-

01
Step Initially queue and visited array are empty.



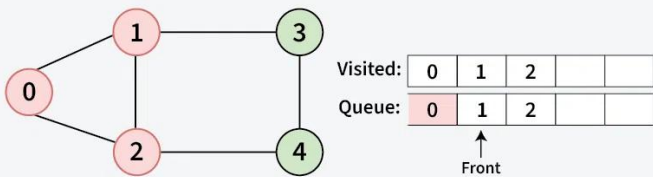
BFS on Graph

02
Step Push 0 into queue and mark it visited.



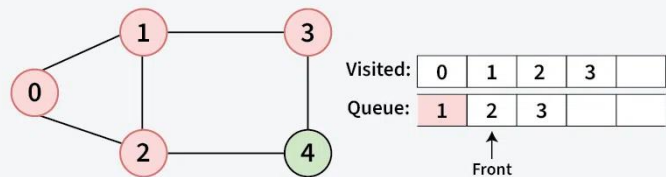
BFS on Graph

03
Step Remove 0 from the front of queue and visit the unvisited neighbours and push them into queue.



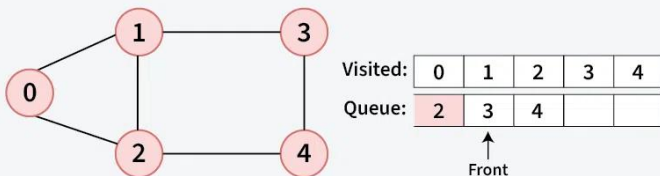
BFS on Graph

04
Step Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



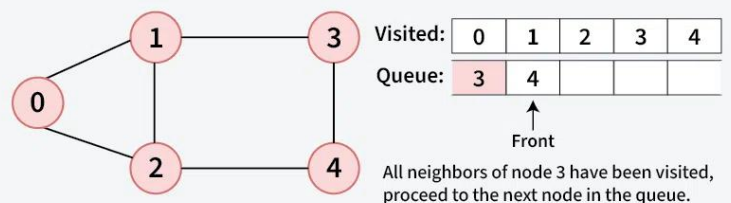
BFS on Graph

05
Step Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



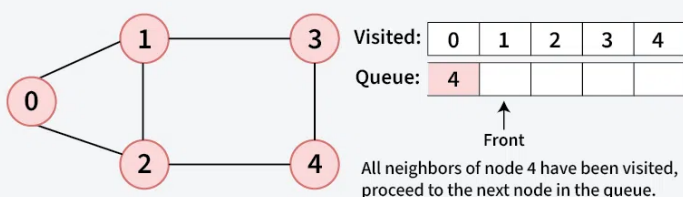
BFS on Graph

06
Step Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.




BFS on Graph

07
Step Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.



BFS on Graph

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

Programming Language: - C++

Code :-

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

void bfs(vector<vector<int>>& adj, int s, vector<bool>& visited);

void bfsDisconnected(vector<vector<int>>& adj) {
    vector<bool> visited(adj.size(), false);

    for (int i = 0; i < adj.size(); ++i) {
        if (!visited[i]) {
            bfs(adj, i, visited);
        }
    }
}


void bfs(vector<vector<int>>& adj, int s, vector<bool>& visited) {
    queue<int> q;

    visited[s] = true;
    q.push(s);

    while (!q.empty()) {
        int curr = q.front();
        q.pop();
        cout << curr << " ";

        for (int x : adj[curr]) {
            if (!visited[x]) {
                visited[x] = true;
                q.push(x);
            }
        }
    }
}

void addEdge(vector<vector<int>>& adj, int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

```
int main(){
    int V = 6;
    vector<vector<int>> adj(V);
    vector<vector<int>> edges = { {1, 2}, {2, 0}, {0, 3}, {4, 5} };
    for (auto& e : edges)
        addEdge(adj, e[0], e[1]);
    cout << "Complete BFS of the graph:" << endl;
    bfsDisconnected(adj);
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12\" ; if ($?) { g++ Breadth_First_Search.cpp -o Breadth_First_Search } ; if ($?) { .\Breadth_First_Search }
Complete BFS of the graph:
0 2 3 1 4 5
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> █
```

Space Complexity:- _____

Justification: -


Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

2. Implement the Depth-First Search

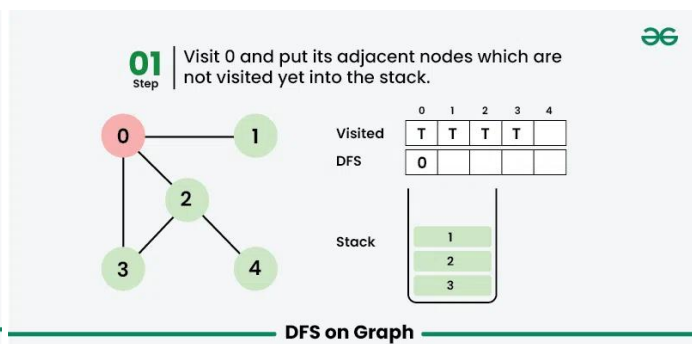
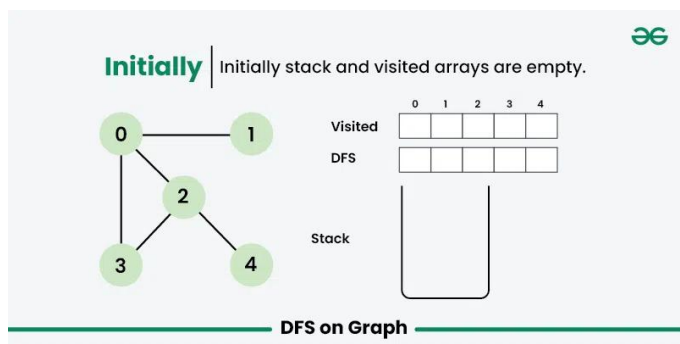
Theory: -

- Depth, First Search (or DFS) for a graph is similar to the Depth First Traversal of a tree. Like trees, we traverse all adjacent vertices one by one. When we traverse an adjacent vertex, we finish the traversal of all vertices reachable through that adjacent vertex. After we finish traversing one adjacent vertex and its reachable vertices, we move to the next adjacent vertex and repeat the process. This is similar to a tree, where we first completely traverse the left subtree and then move to the right subtree. The key difference is that, unlike trees, graphs may contain cycles (a node may be visited more than once). We use a boolean visited array to avoid processing a node multiple times.

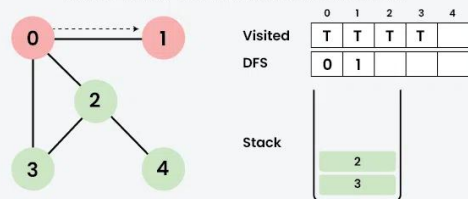
Depth First Search Algorithm:-

- 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 that aren't in the visited list to the top of the stack.
 4. Keep repeating steps 2 and 3 until the stack is empty.

DFS from a Given Source of Undirected Graph:-

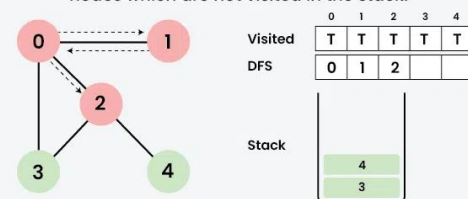


02 Step | Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



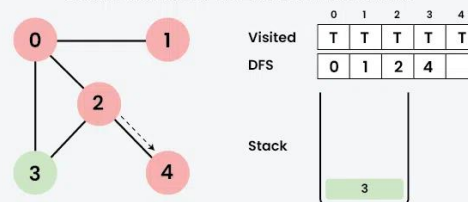
- DFS on Graph

03 Step | Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



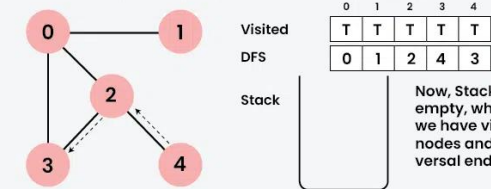
DFS on Graph

04 Step | Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



DFS on Graph

05 Step | Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.




Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

DFS on Graph

Algorithm: -

[illegible]

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

Programming Language: - C++

Code :-

```
#include <bits/stdc++.h>
using namespace std;

void addEdge(vector<vector<int>>& adj, int s, int t) {
    adj[s].push_back(t);
    adj[t].push_back(s);
}

void DFSRec(vector<vector<int>>& adj, vector<bool>& visited, int s) {
    visited[s] = true;
    cout << s << " ";

    for (int i : adj[s])
        if (visited[i] == false)
            DFSRec(adj, visited, i);
}

void DFS(vector<vector<int>>& adj) {
    vector<bool> visited(adj.size(), false);

    for (int i = 0; i < adj.size(); i++) {
        if (visited[i] == false) {
            DFSRec(adj, visited, i);
        }
    }
}


int main() {
    int V = 6;

    vector<vector<int>> adj(V);
    vector<vector<int>> edges = { {1, 2}, {2, 0}, {0, 3}, {4, 5} };

    for (auto& e : edges)
        addEdge(adj, e[0], e[1]);

    cout << "Complete DFS of the graph:" << endl;
    DFS(adj);

    return 0;
}
```


 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Graph Traversal Approach	
Experiment No: 12	Date:	Enrollment No: 92200133030

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12\" ; if ($?) { g++ Depth_First_Search.cpp -o Depth_First_Search } ; if ($?) { .\Depth_First_Search }
Complete DFS of the graph:
0 2 1 3 4 5
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 12> |
```

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

Conclusion:-
