Graphs Traversals

What is Graph Traversal ?

Graph traversal is the process of visiting every vertex and edge of a graph exactly once in a well-defined order.

There are two main methods of graph traversal:

1. Depth-First Search (DFS) and,
2. Breadth-First Search (BFS).

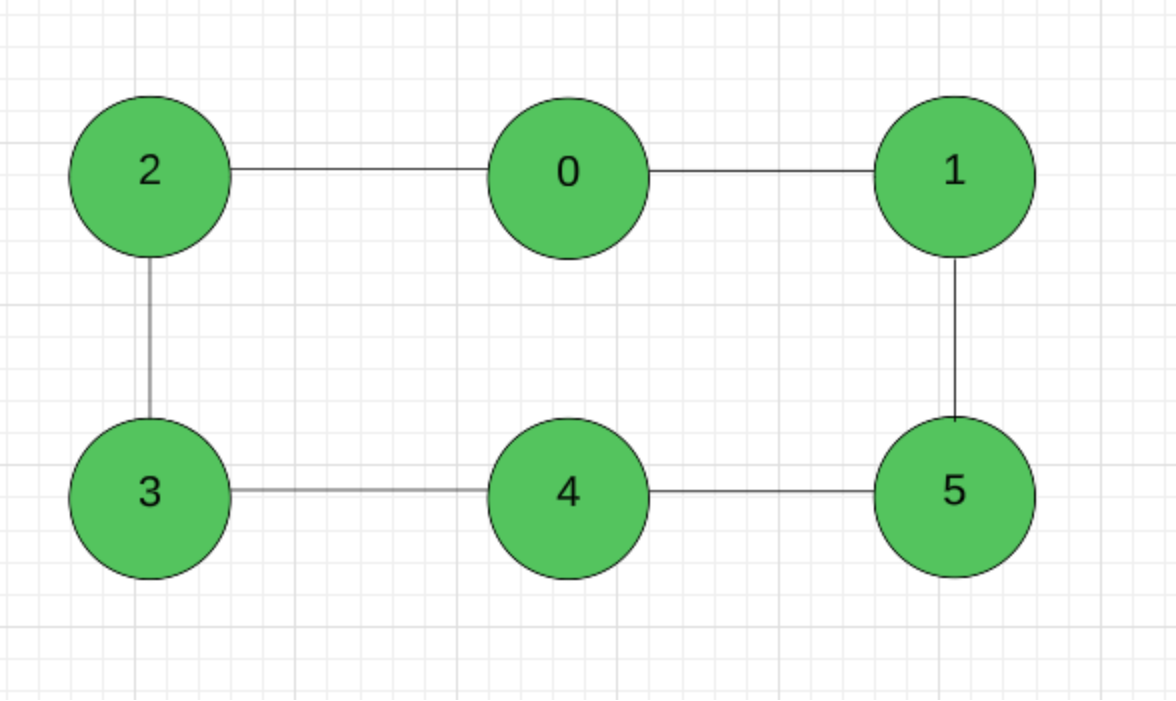
Breadth-First Search (BFS):

BFS stands for Breadth-First Search, which is a graph traversal algorithm used to explore all the vertices of a graph in a breadth-first manner, i.e., visiting all the neighbors of a vertex before moving to the next level.

In simpler terms, BFS algorithm visits all the nodes of a graph level by level, starting from the root node, and moving towards the leaves.

Here's an example to illustrate the BFS algorithm:

Consider the following graph:



If we start BFS from node 2, the algorithm will visit the nodes in the following order: 2, 0, 3, 1, 4, 5.

Explanation:

Because we are starting from node 2, BFS will first visit or print node 2 and then visit or print node 0 and node 3, since they are the neighbors of node 2. So at this point, we visited nodes 2, 0, and 3.

Now, this graph is an undirected graph and we are standing at node 2. Now from this node 2, we can move either toward node 0 or node 3. We can move in any one direction it's our choice. So I am moving toward node 0.

Now, we are at node 0 and BFS will try to print node 0 but node 0 is already visited or printed in the past so we don’t print this node 0 again. Now BFS will try to print the neighbors of node 0 and neighbors of node 0 are node 2 and node 1, but node 2 is already printed in past so we don’t print this node 2 again and BFS will print the node 1. Now from node 0, we have again 2 choices either we move towards node 1 or we can move towards node 2. Its our choice, so I am moving toward node 1.

Now, we are at node 1 and BFS will try to print node 1 but node 1 is already visited or printed in the past so we don’t print this node 1 again. Now BFS will try to print the neighbors of node 1 and neighbors of node 1 are node 0 and node 5, but node 0 is

already printed in past so we don’t print this node 0 again and BFS will print the node 5. Now from node 1, we have again 2 choices either we move towards node 0 or we can move towards node 5. Its our choice, so I am moving toward node 5.

Now, we are at node 5 and BFS will try to print node 5 but node 5 is already visited or printed in the past so we don’t print this node 5 again. Now BFS will try to print the neighbors of node 5 and neighbors of node 5 are node 1 and node 4, but node 1 is already printed in past so we don’t print this node 1 again and BFS will print the node 4. Now from node 5, we have again 2 choices either we move towards node 4 or we can move towards node 1. Its our choice, so I am moving toward node 4.

Now, we are at node 4 and BFS will try to print node 4 but node 4 is already visited or printed in the past so we don’t print this node 4 again. Now BFS will try to print the neighbors of node 4 and neighbors of node 4 are node 3 and node 5, Now node 5 and node 3 are already printed in past so we don’t print this node 5 and node 3 again. At this point we traverse the entire Graph using BFS traversal.

Additional Data Structures we need to perform BFS:

In BFS, we typically use two data structures:

1. **A Boolean array or hash table:**
    a. to keep track of which vertices have been visited. We initialize this array to all `false` or all `unvisited`, and then set the value of a vertex to `true` or `visited` when we visit it. This allows us to avoid visiting the same vertex twice.
2. **A Queue:**
    a. The queue is used to keep track of the vertices that have been visited but not processed yet. We start by adding the source vertex to the queue and mark it as visited. Then we repeatedly remove the next vertex from the front of the queue, visit all its unvisited neighbors, mark them as visited, and add them to the back of the queue. We continue this process until there are no more vertices left in the queue.

Time and Space Complexity Of Breadth First Search:

The time complexity of BFS (Breadth-First Search) is O(V + E), where V is the number of vertices and E is the number of edges in the graph. This is because BFS visits each

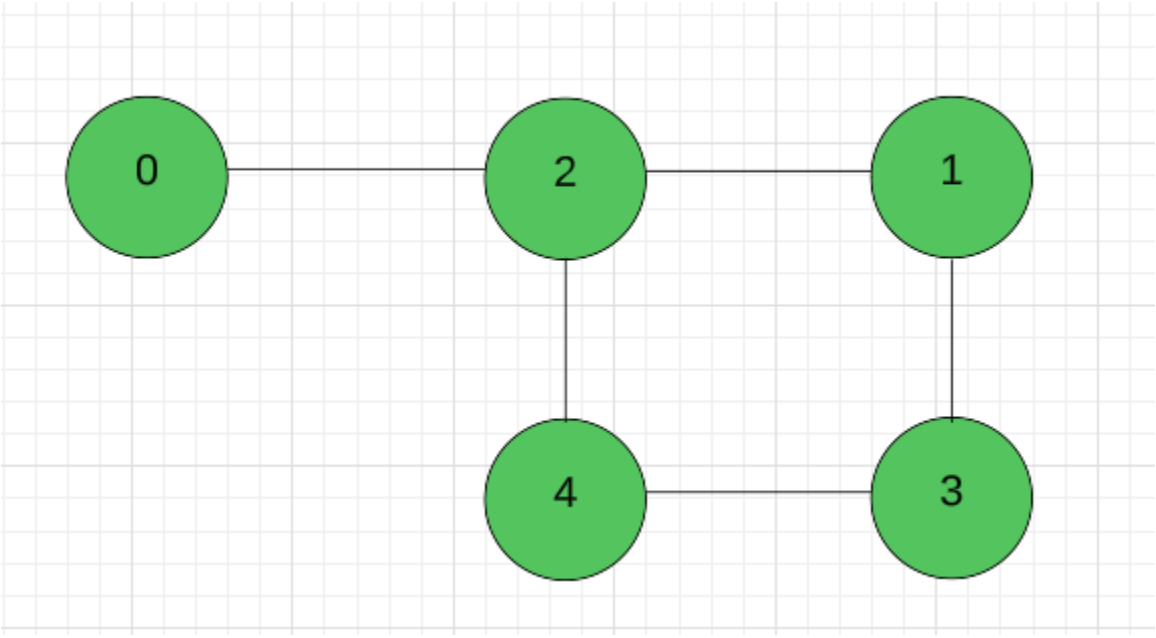
vertex and each edge exactly once. The worst-case scenario for BFS is when we visit all the vertices and edges in the graph.

The space complexity of BFS is also O(V + E), as we need to store the adjacency list or matrix of the graph, the queue for BFS, and the Boolean array or hash table to keep track of the visited vertices. In the worst-case scenario, when all the vertices and edges are connected, the queue can hold all the vertices, which makes the space complexity O(V).

Depth First Search (DFS):

DFS stands for Depth-First Search, which is another algorithm used to traverse a graph. Depth-First Search or DFS algorithm is a recursive algorithm that uses the backtracking principle. In DFS, we start at a source node and explore as far as possible along each branch before backtracking.

For example, let's say we have the following undirected graph:



DFS Traversal of given graph is: 0, 2, 4, 3, 1

Note: First visit the the current node and then move towards the adjacent node (if more than one adjacent nodes are there in this case, move toward any one of the adjacent node).

Explanation:

Step 1: Starting from node 0, mark it as visited and add it to the result list.

```
Visited: [0]
Result: [0]
```

Step 2: Explore the unvisited neighbors of node 0 in the order they appear in the adjacency list. The only neighbor of node 0 is node 2, which has not been visited yet. Therefore, move to node 2.

```
Visited: [0, 2]
Result: [0]
```

Step 3: Mark node 2 as visited and add it to the result list. Then, explore the unvisited neighbors of node 2. The neighbors of node 2 are nodes 1 and 4. According to the order they appear in the adjacency list, move to node 4.

```
Visited: [0, 2, 4]
Result: [0, 2]
```

Step 4: Mark node 4 as visited and add it to the result list. Then, explore the unvisited neighbors of node 4. The only neighbor of node 4 is node 3, which has not been visited yet. Therefore, move to node 3.

```
Visited: [0, 2, 4, 3]
Result: [0, 2, 4]
```

Step 5: Mark node 3 as visited and add it to the result list. There are no unvisited neighbors of node 3. Therefore, backtrack to node 4.

```
Visited: [0, 2, 4, 3]
Result: [0, 2, 4, 3]
```

Step 6: There are no unvisited neighbors of node 4. Therefore, backtrack to node 2.

```
Visited: [0, 2, 4, 3]
Result: [0, 2, 4, 3]
```

Step 7: There is only one unvisited neighbor of node 2, which is node 1. Therefore, move to node 1.

```
Visited: [0, 2, 4, 3, 1]
Result: [0, 2, 4, 3]
```

Step 8: Mark node 1 as visited and add it to the result list. There are no unvisited neighbors of node 1. Therefore, backtrack to node 2.

```
Visited: [0, 2, 4, 3, 1]
Result: [0, 2, 4, 3, 1]
```

Step 9: There are no unvisited neighbors of node 2. Therefore, backtrack to node 0.

```
Visited: [0, 2, 4, 3, 1]
Result: [0, 2, 4, 3, 1]
```

Step 10: There are no unvisited neighbors of node 0. Therefore, the DFS traversal is complete. The final result is:

```
Result: [0, 2, 4, 3, 1]
```

Time and Space Complexity of DFS:

The time complexity of DFS is O(V + E), where V is the number of vertices in the graph, and E is the number of edges. This is because DFS traverses every vertex and edge exactly once.

The space complexity of DFS depends on the implementation. If DFS is implemented recursively, the space complexity is O(V), where V is the number of vertices in the graph. This is because the recursive implementation uses a function call stack to store the visited vertices.