DSA Tutorial - Graph Traversals

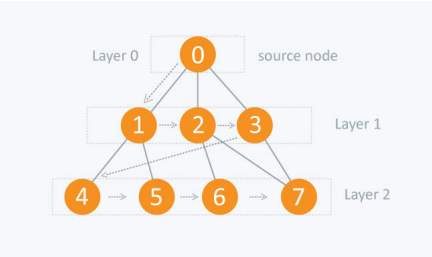
April 20, 2018

1 Breadth First Search

There are many ways to traverse graphs. BFS is the most commonly used approach.
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 neighbour nodes.
As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

- First move horizontally and visit all the nodes of the current layer
- Move to the next layer

Consider the following diagram:

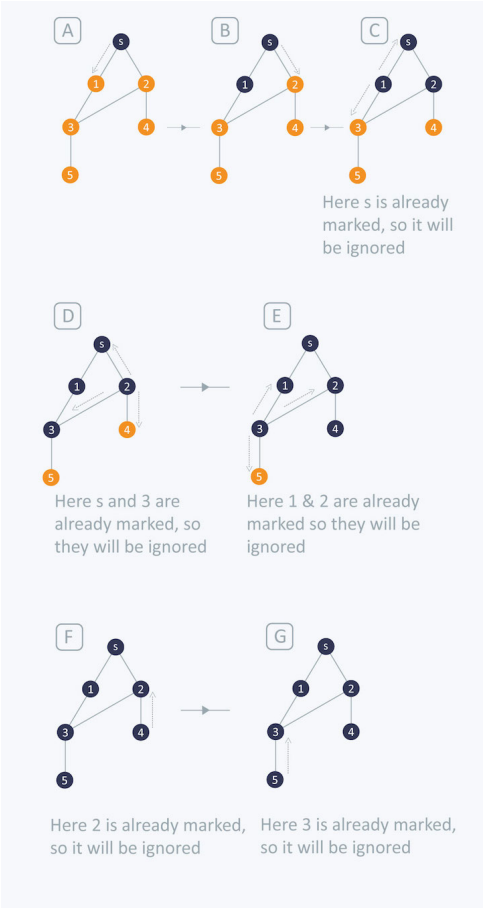


The distance between the nodes in layer 1 is comparitively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

1.1 Traversing child nodes

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.
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.
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 neighbours 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.

1.2 Traversing process



1.3 Complexity

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

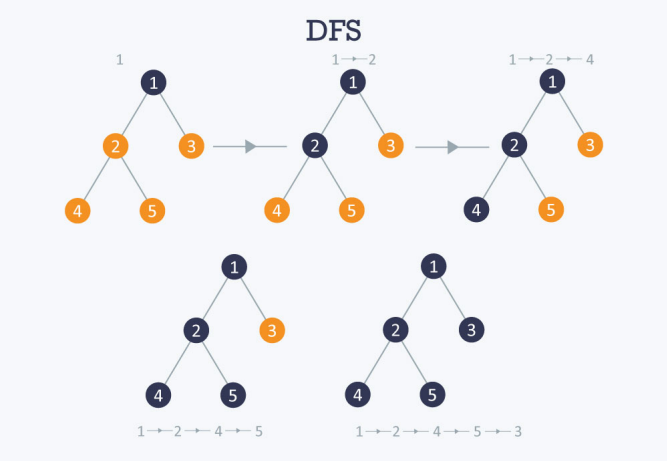
2 0-1 BFS

This type of BFS is used to find the shortest distance between two nodes in a graph provided that the edges in the graph have the weights 0 or 1. If you apply the BFS explained earlier in this article, you will get an incorrect result for the optimal distance between 2 nodes.
In this approach, a boolean array is not used to mark the node because the condition of the optimal distance will be checked when you visit each node. A double-ended queue is used to store the node. In 0-1 BFS, if the weight of the edge = 0, then the node is pushed to the front of the dequeue. If the weight of the edge = 1, then the node is pushed to the back of the dequeue.

3 Depth First Search (DFS)

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: Pick a starting node and push all its adjacent nodes into a stack. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. 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. Using DFS, We can also find out if a graph is bipartite or not.
The following image shows how DFS works.



3.1 Complexity

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

4 Problems

4.1 DFS

1. https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/practice-problems/algorithm/dfs-3/
2. http://codeforces.com/contest/893/problem/C
3. http://www.spoj.com/problems/BUGLIFE/

4.2 BFS

1. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/practice-problems/algorithm/bfs/
2. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/practice-problems/algorithm/monk-and-the-islands/

5 References

1. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/
2. https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/