Algorithms and Data Structures for Beginners

21 / 35

20 - Breadth-First Search

Mark Lesson Complete View Code Prev Next

15 Search Range

Suggested Problems

Status	Star	Problem	Difficulty	Video Solution	Code
☐	☆	Binary Tree Level Order Traversal	Medium	▶	C++
☐	☆	Binary Tree Right Side View	Medium	▶	C++

Breadth First Search

Concept

In depth-first search, we prioritized depth. For breath-first serach, we prioritize breadth. We focus visiting all the nodes on one level before moving on to the next level.

Generally, breadth-first search is implemented iteratively and that is the implementation we will be covering in this course. You can write it recursively but it is a lot more challenging.

BFS makes use of a queue data structure, more specifically, a deque, allowing us to remove elements both from the head and the tail in $O(1)$ time. This will make sense soon.

The pseudocode for BFS is shown below.

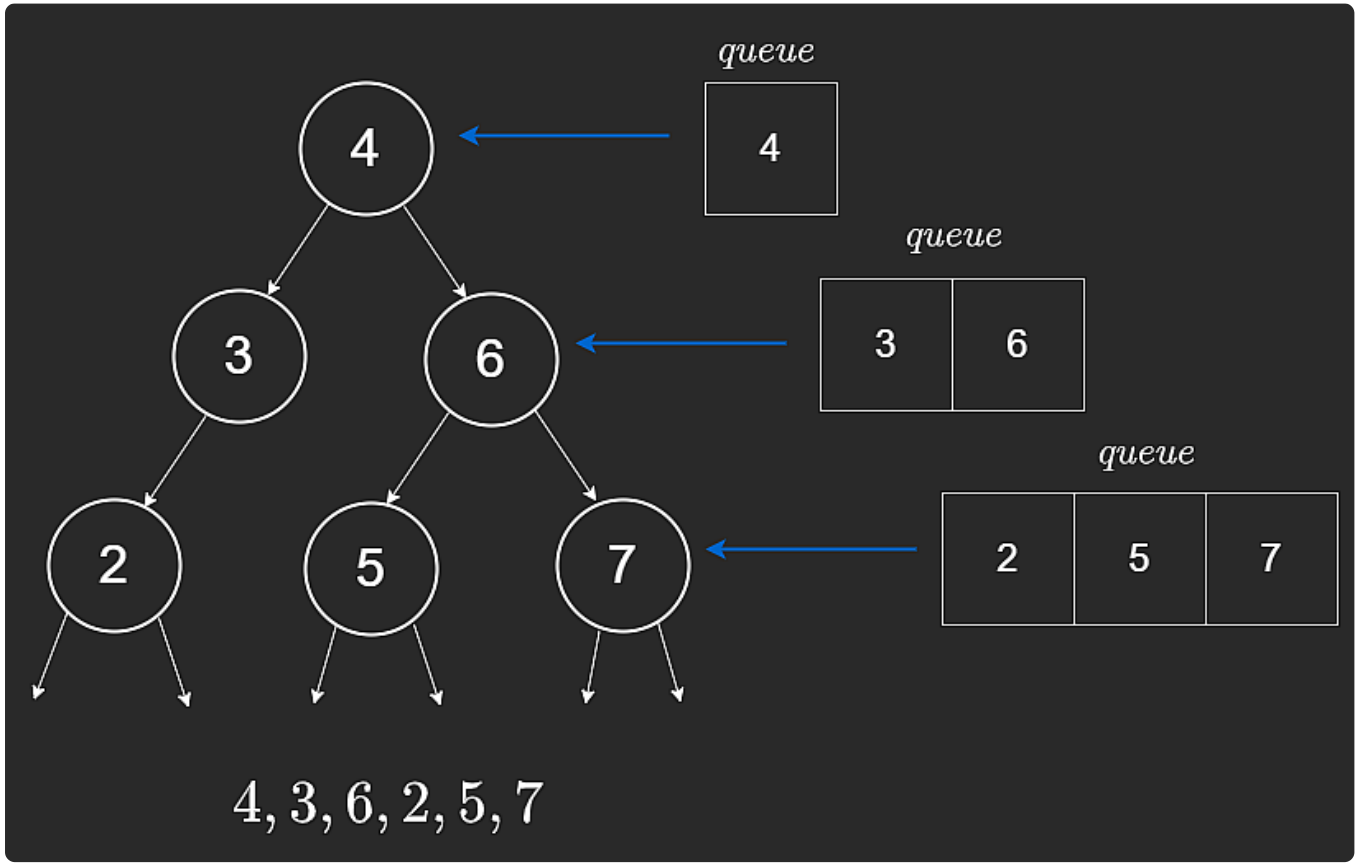
```
fn bfs(root):
    queue = deque()
    if root is not null:
        queue.append(root)

    level = 0
    while queue.length > 0:
        print("level: ", level)
        for i = 0 until queue.length:
            curr = queue.popleft()
            print(curr.val)
            if curr.left:
                queue.append(curr.left)
            if curr.right:
                queue.append(curr.right)
        level += 1
```

Let's take an example of a binary tree, `[4,3,6,2,null,5,7]` and apply the BFS algorithm. Remember, our goal is to visit all the nodes on one level before moving to the next.

If we append our `root` node to our queue and loop through the queue such that at any given time, our queue only holds the nodes on a certain level, we will ensure that we visit the levels in order and that we don't mix up the levels either. This is exactly what the code inside of the while loop achieves. As long as our queue is not empty, we remove the node(s) that is present in our queue and add its children to the queue (which would be the next level). Therefore, when we remove `root`, we add its children, which are `3,6` to the queue. Next, we remove `3` and add its child `2`. We then remove `6` and add `5,7` to the queue. Because of the FIFO nature of a queue, we ensure that we visit the nodes from left to right.

Our queue becomes empty once we have visited all of the nodes.

The visual below demonstrates what the state of the queue at every level of the tree would look like.



Time Complexity

Technically, the total work done is $c * n$ where n is the number of nodes in the tree and c is the amount of work we perform at each node. We performed a total of three operations per node - printing the node, appending the node, and removing it. This is what the c represents. For the case of asymptotic analysis, we can drop this constant, meaning the algorithm belongs to $O(n)$.

Closing Notes

Breadth First Search will be extremely useful when it comes to graph traversals, which we shall see soon.

Github Privacy Terms