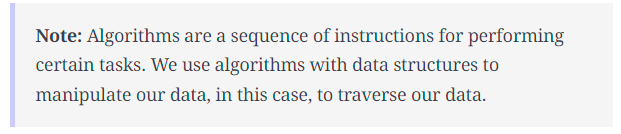
**Intro to Tree Traversal and Searching**

To use trees, we can **traverse** them by visiting/checking every node of a tree. If a tree is “traversed”, this means every node has been visited. There are four ways to traverse a tree. These four processes fall into one of two categories: breadth-first traversal or depth-first traversal.

* **Inorder:** Think of this as moving up the tree, then back down. You traverse the left child and its sub-tree until you reach the root. Then, traverse down the right child and its subtree. This is a depth-first traversal.
* **Preorder:** This of this as starting at the top, moving down, and then over. You start at the root, traverse the left sub-tree, and then move over to the right sub-tree. This is a depth-first traversal.
* **Postorder:** Begin with the left-sub tree and move over to the right sub-tree. Then, move up to visit the root node. This is a depth-first traversal.
* **Levelorder:** Think of this as a sort of zig-zag pattern. This will traverse the nodes by their levels instead of subtrees. First, we visit the root and visit all children of that root, left to right. We then move down to the next level until we reach a node that has no children. This is the left node. This is a breadth-first traversal.

So, what’s the difference between a *breadth-first* and *depth-first traversal*? Let’s take a look at the algorithms Depth-First Search (DFS) and Breath-First Search (BFS) to understand this better.



Depth -First Search

**Overview:** We follow a path from the starting node to the ending node and then start another path until all nodes are visited. This is commonly implemented using stacks, and it requires less memory than BFS. It is best for topographical sorting, such as graph backtracking or cycle detection.

The steps for the DFS algorithm are as follows:

1. Pick a node. Push all adjacent nodes into a stack.
2. Pop a node from that stack and push adjacent nodes into another stack.
3. Repeat until the stack is empty or you have reached your goal. As you visit nodes, you must mark them as visited before proceeding, or you will be stuck in an infinite loop.

Breadth-First Search

**Overview:** We proceed level-by-level to visit all nodes at one level before going to the next. The BFS algorithm is commonly implemented using queues, and it requires more memory than the DFS algorithm. It is best for finding the shortest path between two nodes.

The steps for the BFS algorithm are as follows:

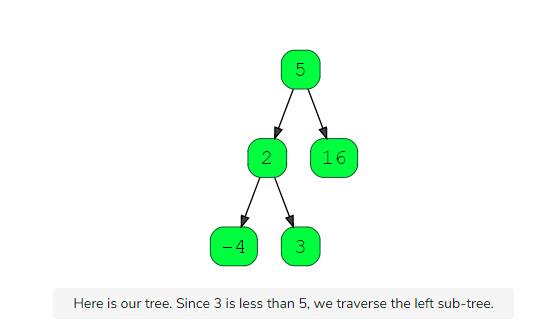
1. Pick a node. Enqueue all adjacent nodes into a queue. Dequeue a node, and mark it as visited. Enqueue all adjacent nodes into another queue.
2. Repeat until the queue is empty of you have met your goal.
3. As you visit nodes, you must mark them as visited before proceeding, or you will be stuck in an infinite loop.

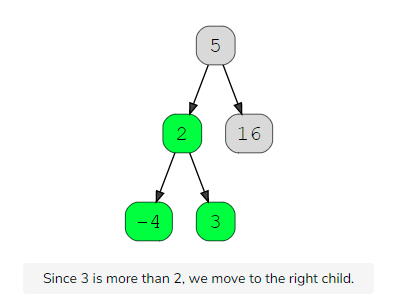
### Search in Binary Search Trees

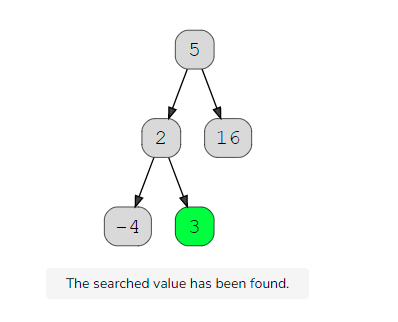
It’s important to know how to perfom a search in a tree. Searching means we are locating a specific element or node in our data structure. Since data in a Binary Search Tree is ordered, searching is quite easy. Let’s see how it’s done.

1. Start at the root.
2. If the value is less than the value of the current node, we traverse the left sub-tree. If it is more, we traverse the right sub-tree.
3. Continue this process until you reach a node with that value or reach a leaf node, meaning that the value doesn’t exist.

In the below example, we are searching for the value 3 in our tree. Take a look.







Let’s see that in Java code now!

|  |
| --- |
| public class BinarySearchTree {  …    public boolean search(int value) {  if (root == null)  return false;  else  return root.search(value);  }  }  public class BSTNode {  …  public boolean search(int value) {  if (value == this.value)  return true;  else if (value < this.value) {  if (left == null)  return false;  else  return left.search(value);  } else if (value > this.value) {  if (right == null)  return false;  else  return right.search(value);  }  return false;  }  } |