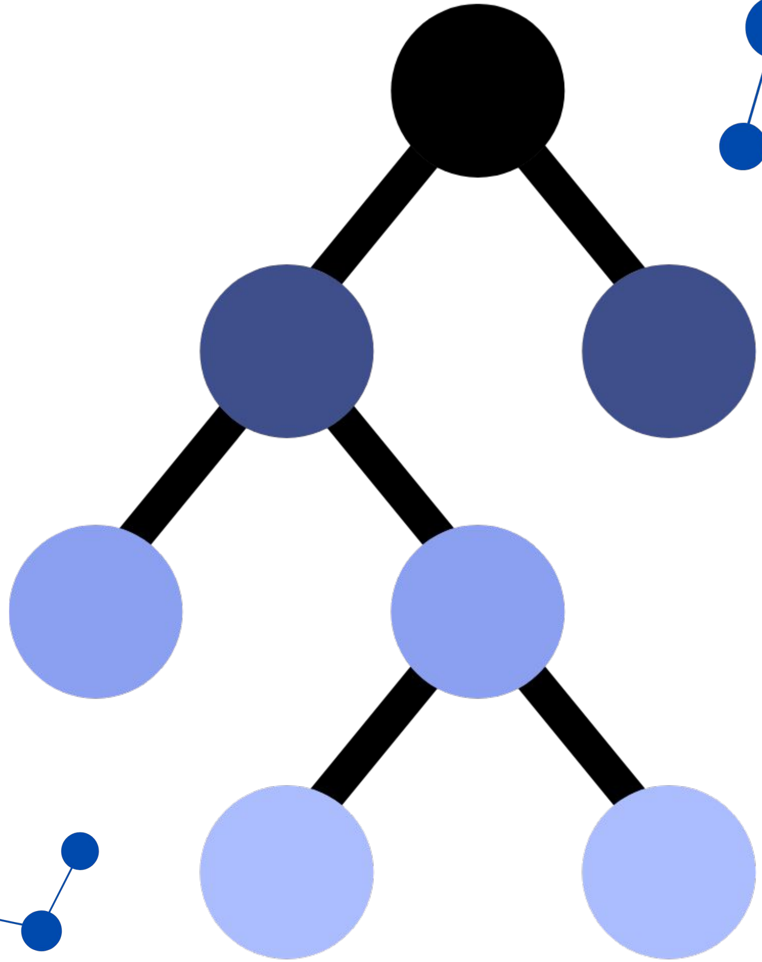
Trees

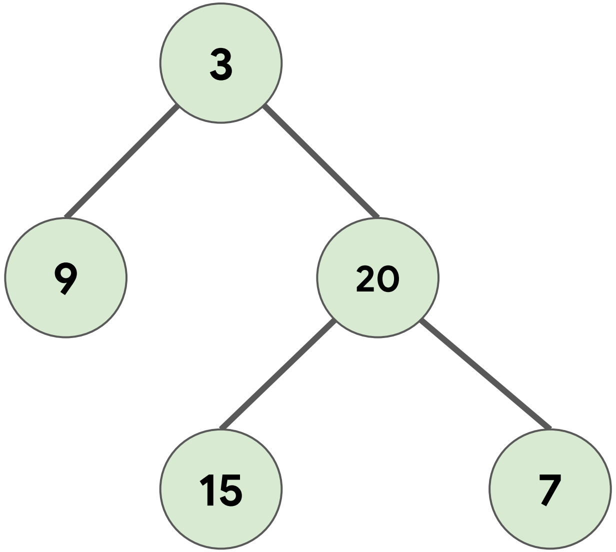
Binary Trees and Binary Search Trees

Part II



Simulation

Given the root of a binary tree, return its maximum depth. A binary tree maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.



Example 1:

Input: `root = [3,9,20,null,null,15,7]`

Output: 3

Example 2:

Input: `root = [1,null,2]`

Output: 2

Simulation - Solution

- What if the tree is empty?
  Answer: `max_depth = 0`

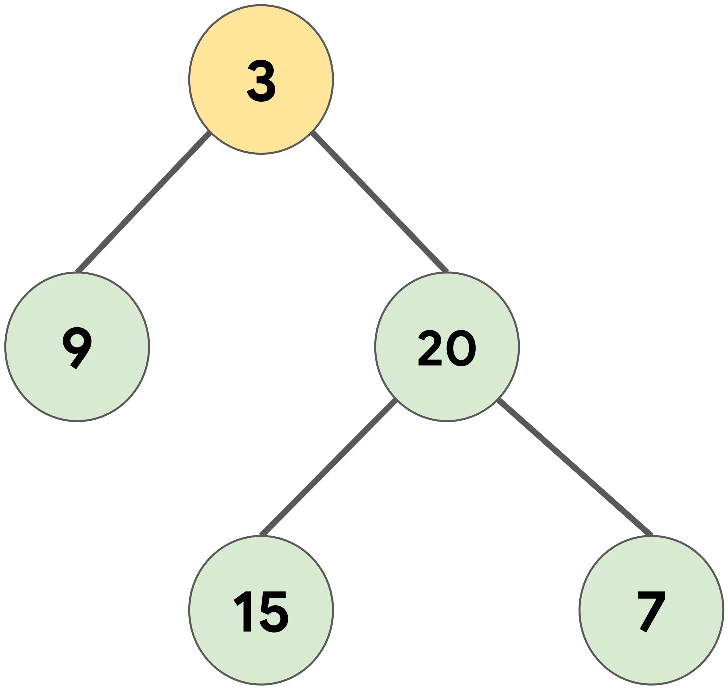
- What if we have just a node with no children?
  Answer: `max_depth = 1`

- What if the current node has left and/or right children?
  Answer: `max_depth = 1 + max(left_child_max_depth, right_child_max_depth)`

By recursively calculating the `max_depth` of each subtree

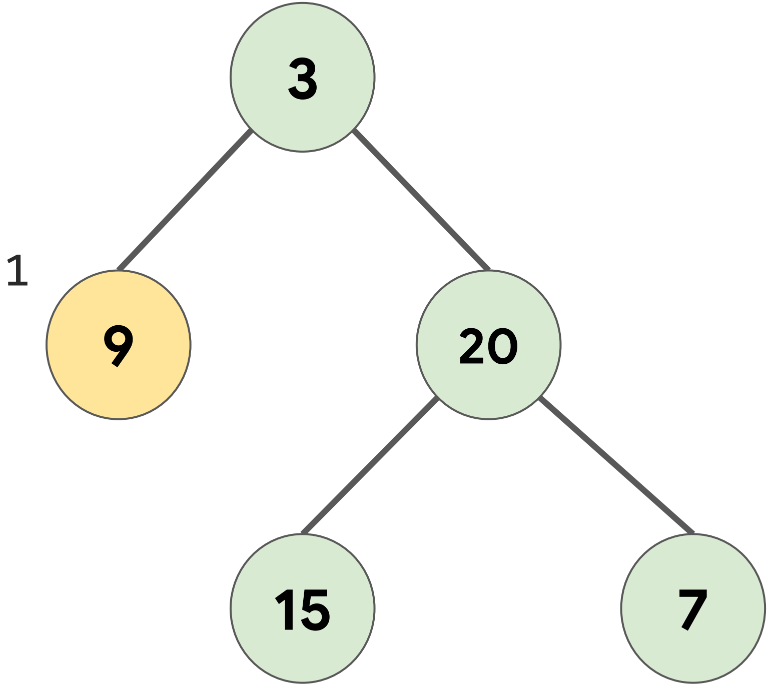
Simulation - Solution

```
1 + max(max_depth(root.left), max_depth(root.right))
```



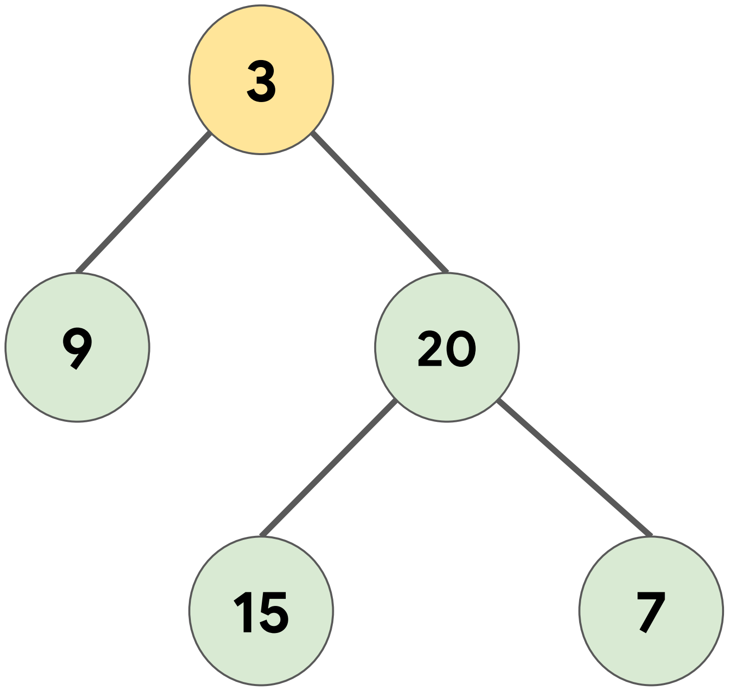
Simulation - Solution

```
1 + max(max_depth(root.left), max_depth(root.right))
```



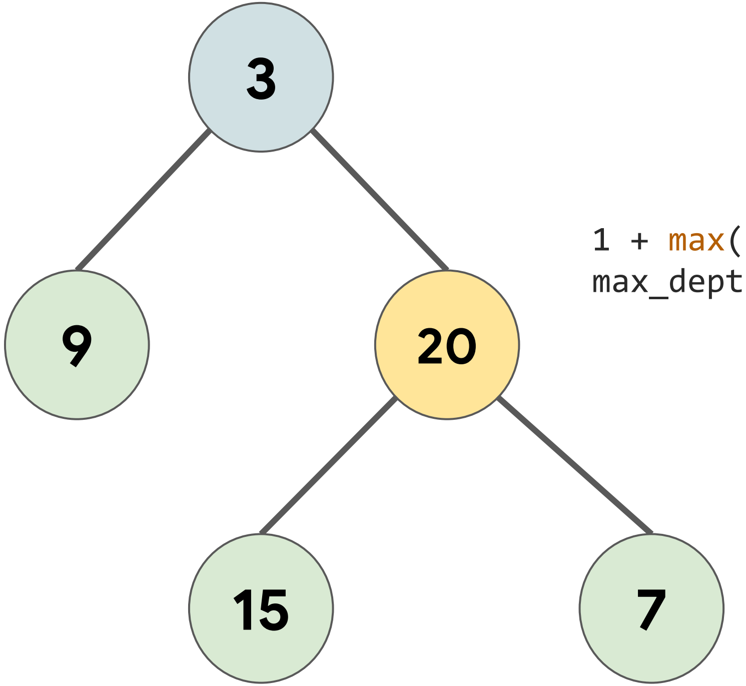
Simulation - Solution

```
1 + max(1, max_depth(root.right))
```



Simulation - Solution

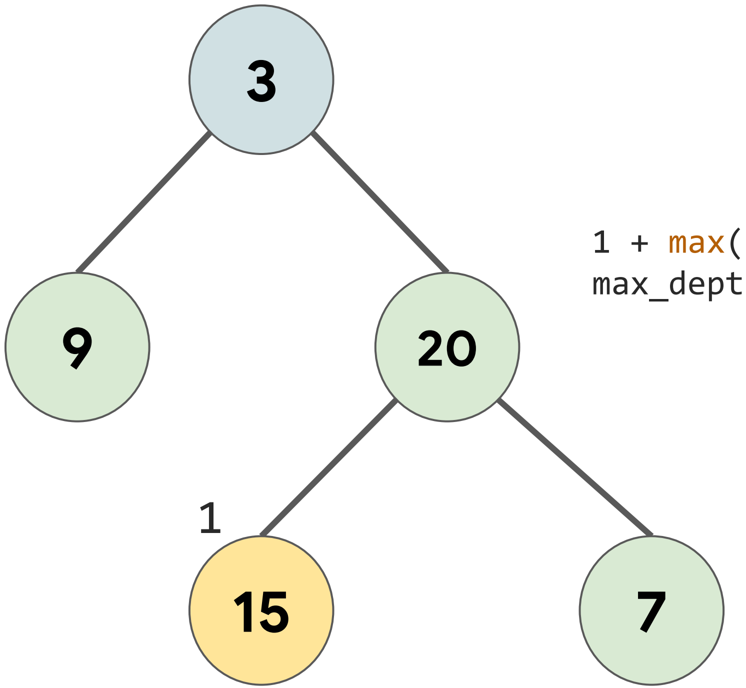
`1 + max(1, max_depth(root.right))`



`1 + max(max_depth(root.left), max_depth(root.right))`

Simulation - Solution

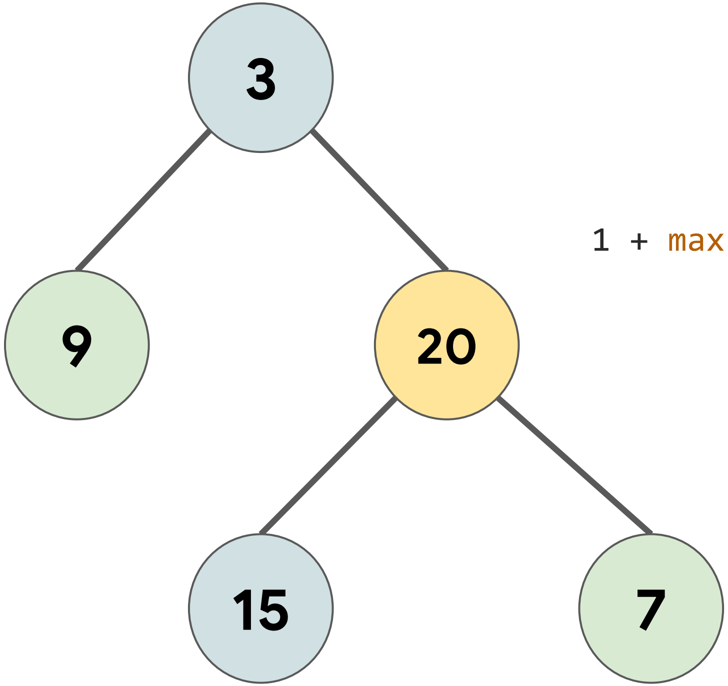
`1 + max(1, max_depth(root.right))`




```
1 + max(max_depth(root.left),
max_depth(root.right))
```

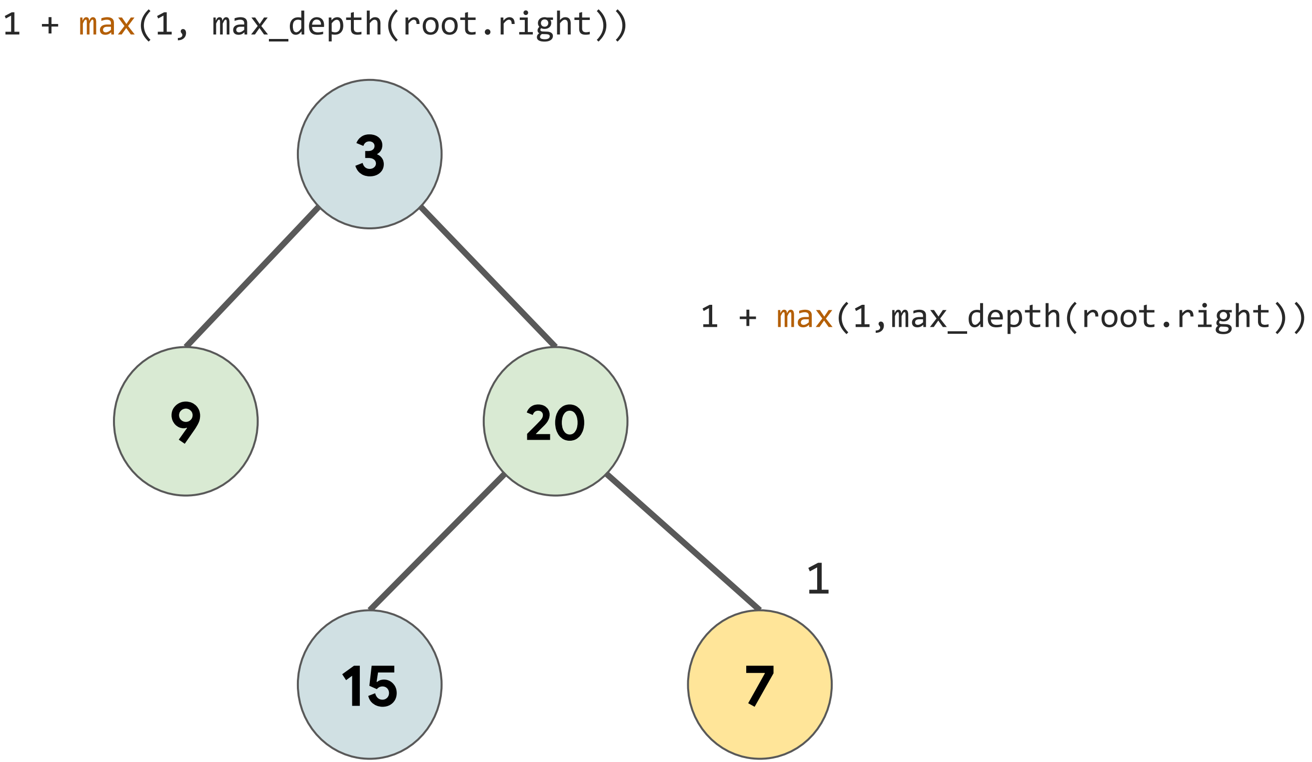
Simulation - Solution

`1 + max(1, max_depth(root.right))`



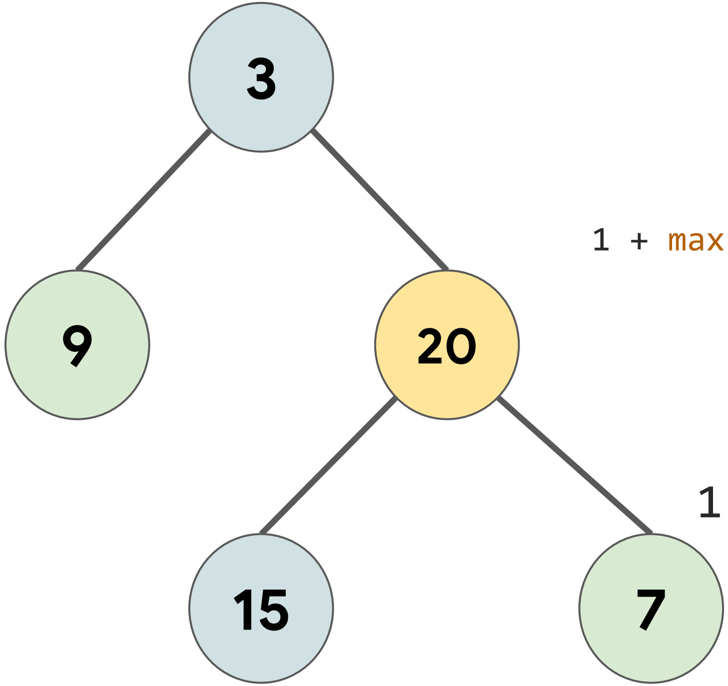
`1 + max(1,max_depth(root.right))`

Simulation - Solution



Simulation - Solution

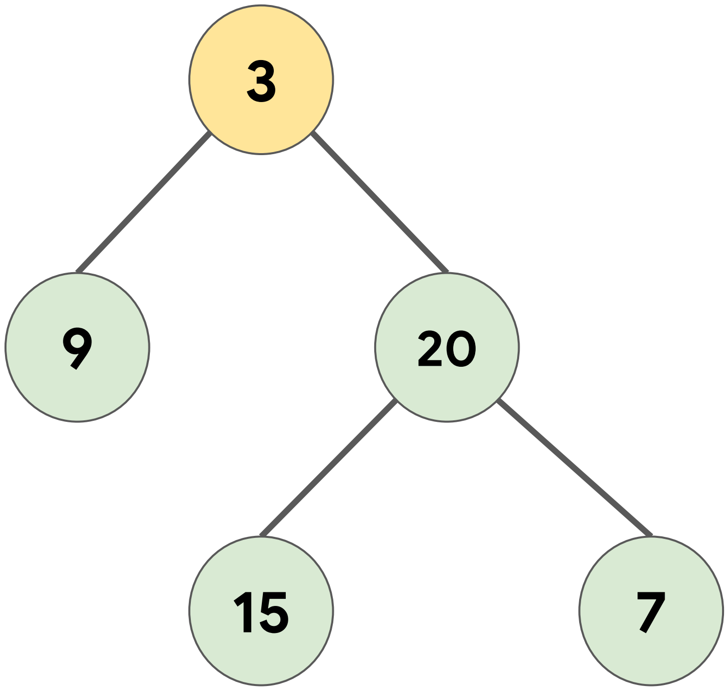
`1 + max(1, max_depth(root.right))`



`1 + max(1, 1) = 2`

Simulation - Solution

```
1 + max(1,2) = 3
```



Question

[Insert into a Binary Search Tree]

Basic Operation on Trees

Operation - Searching

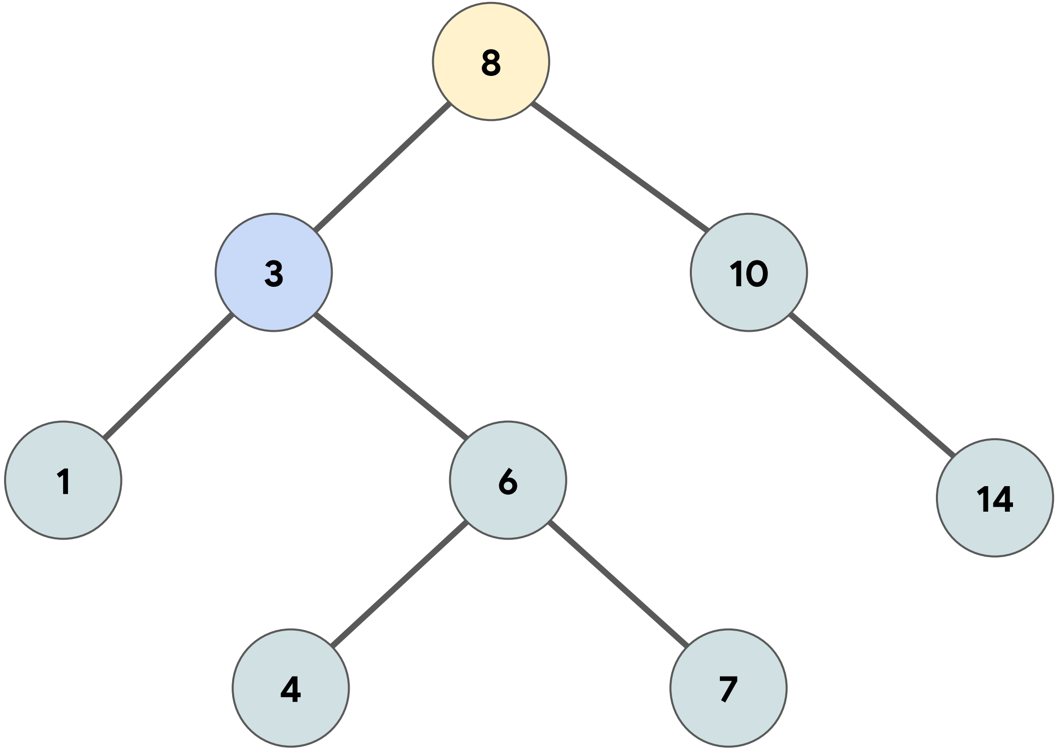
- The algorithm depends on the property of BST that if each left subtree has values below parent and each right subtree has values above the parent.

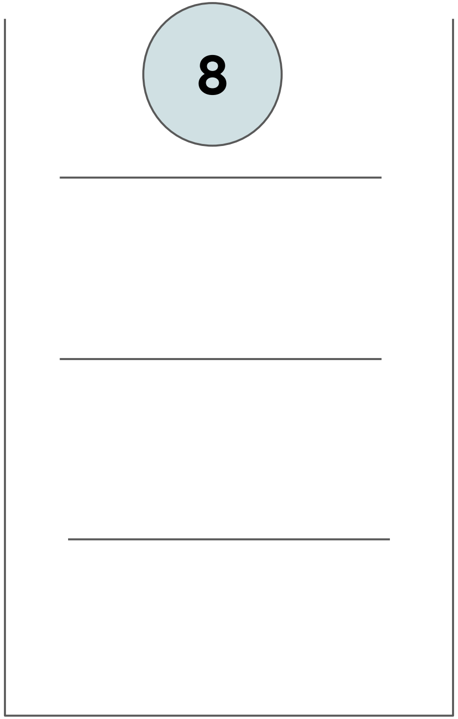
- If the value is below the parent, we can say for sure that the value is not in the right subtree; we need to only search in the left subtree

- If the value is above the parent, we can say for sure that the value is not in the left subtree; we need to only search in the right subtree.

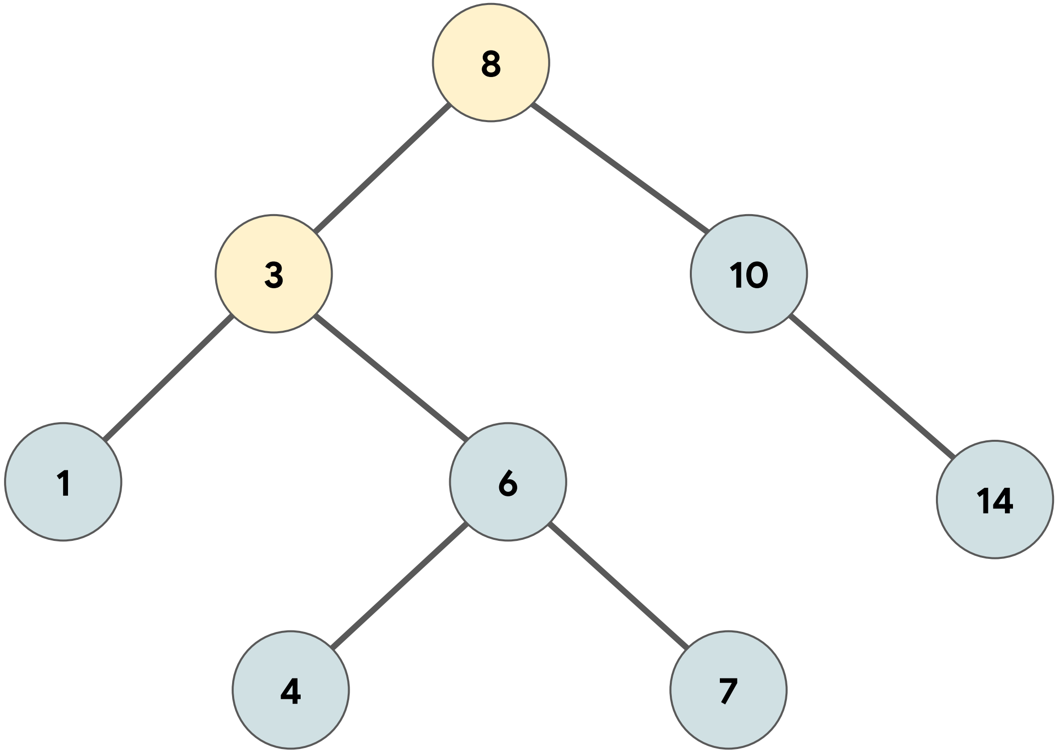
- Let us try to visualize this with a diagram searching for 4 in the tree:

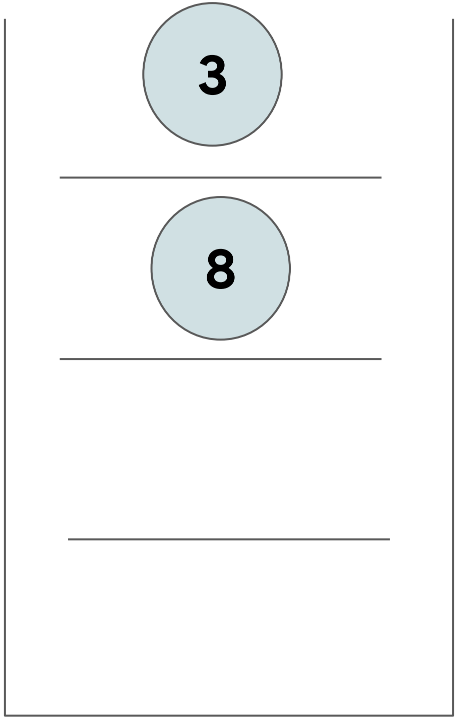
Operation - Searching



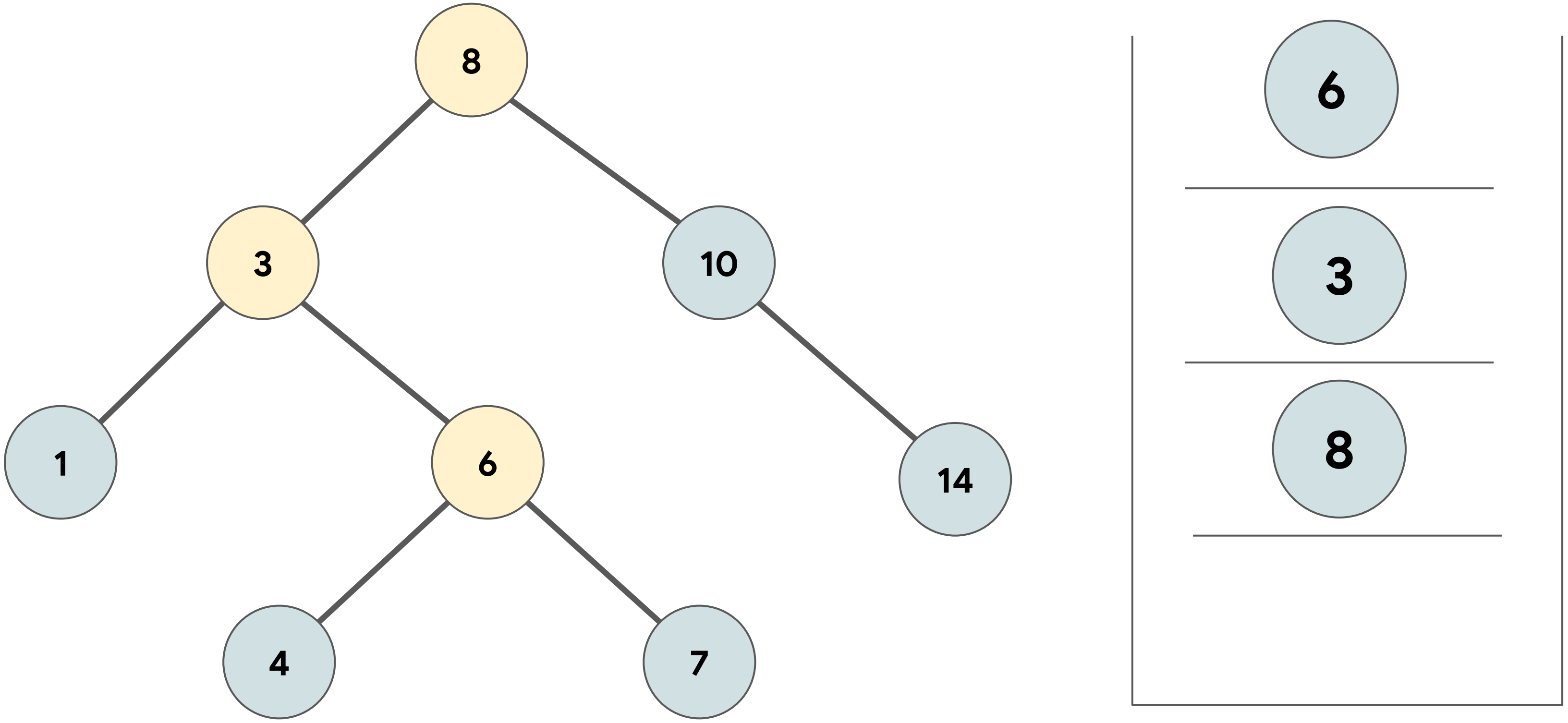


Operation - Searching

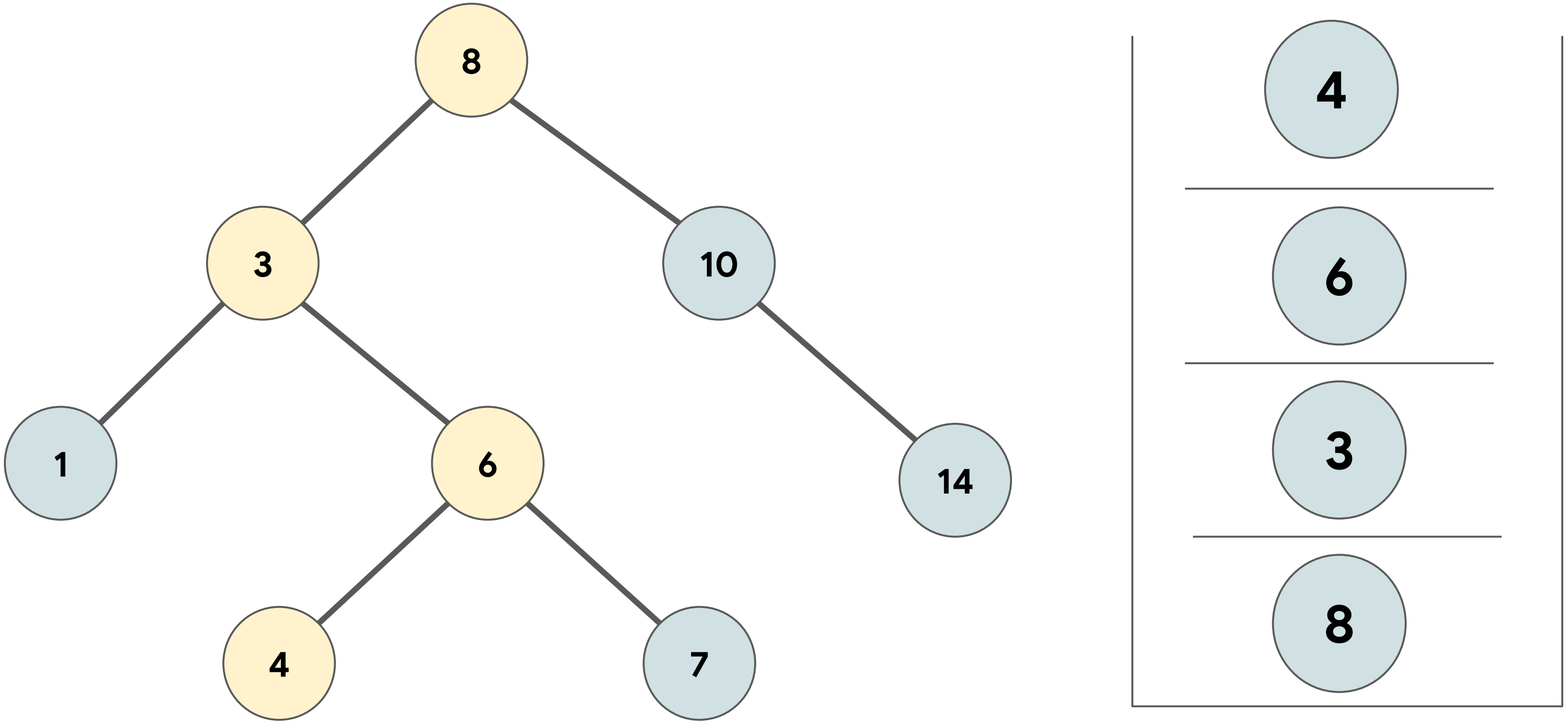




Operation - Searching



Operation - Searching



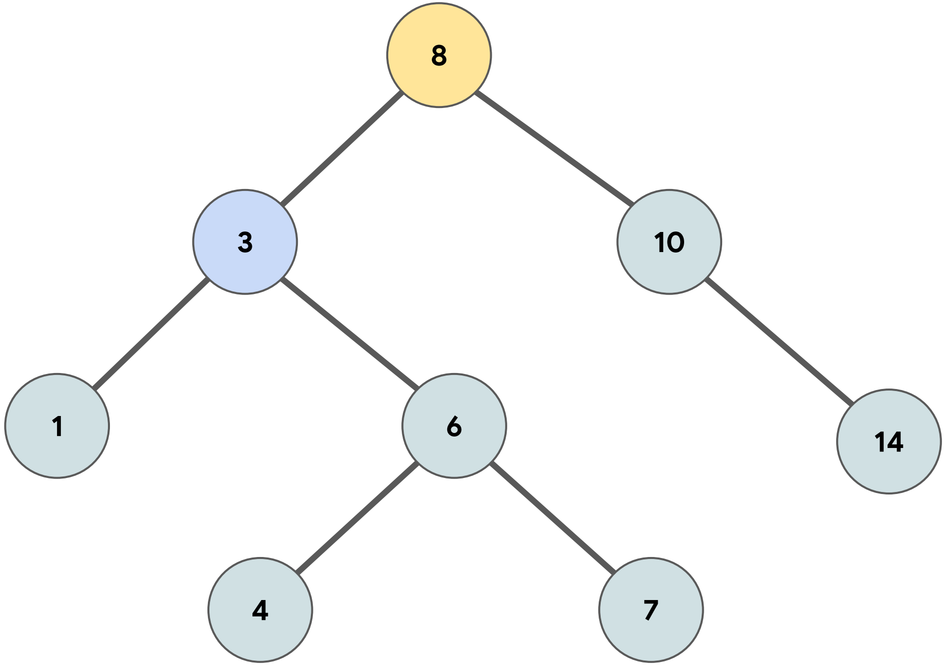
Operation - Insertion

- Inserting a value in the correct position is similar to searching because we try to maintain the BST rule that the left subtree is lesser than root and the right subtree is larger than root.
- We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

Let's try to visualize how we add a number 5 to an existing BST.

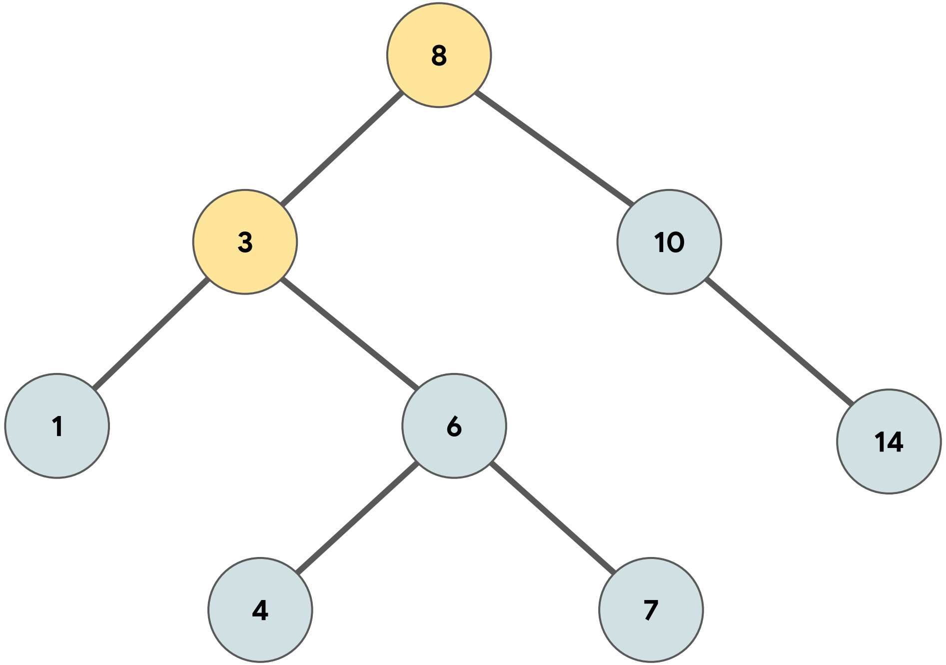
Operation - Insertion

Insert 5 in to the BST



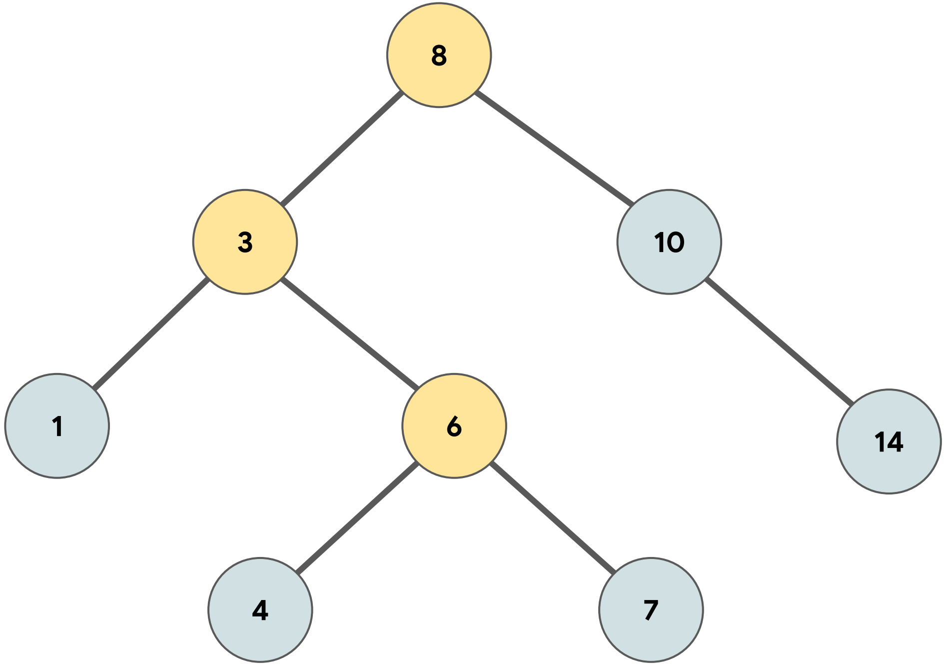
Operation - Insertion

Insert 5 in to the BST



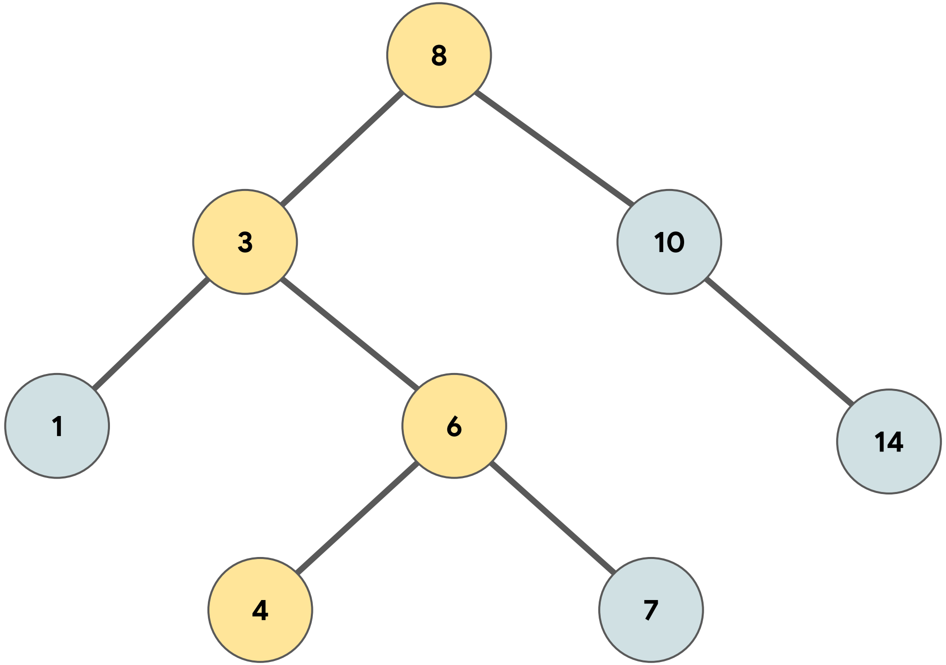
Operation - Insertion

Insert 5 in to the BST



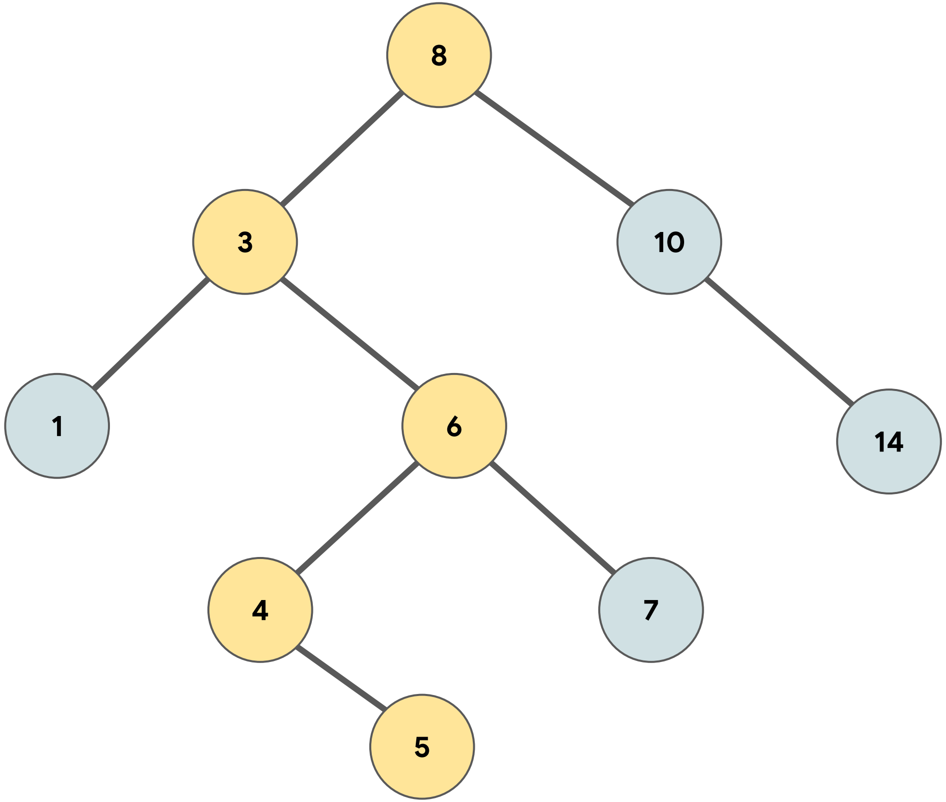
Operation - Insertion

Insert 5 in to the BST



Operation - Insertion

Insert 5 in to the BST



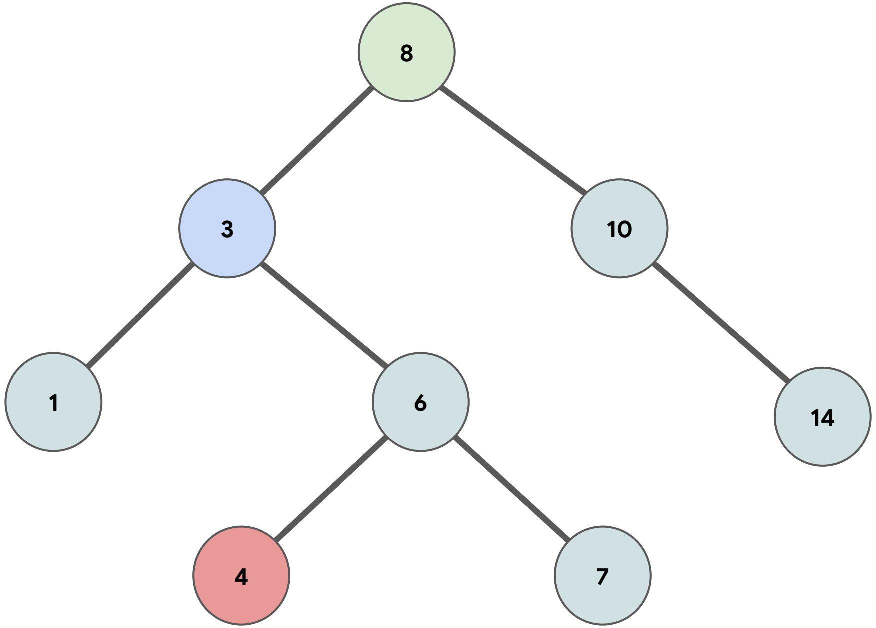
Practice Problem

Delete Node in a BST

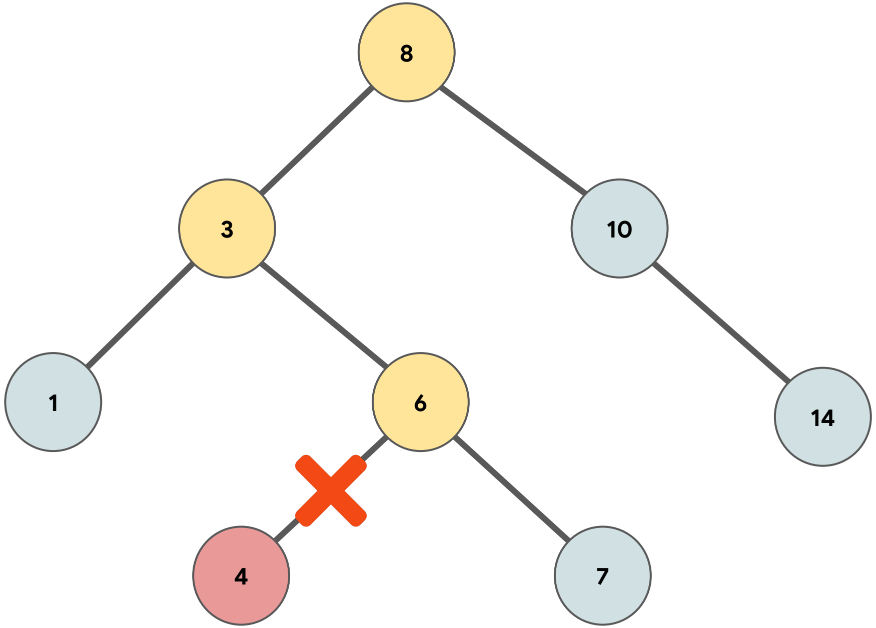
Operation - Deletion

- There are three cases for deleting a node from a binary search tree.

  Case One: In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree. 4 is to be deleted.



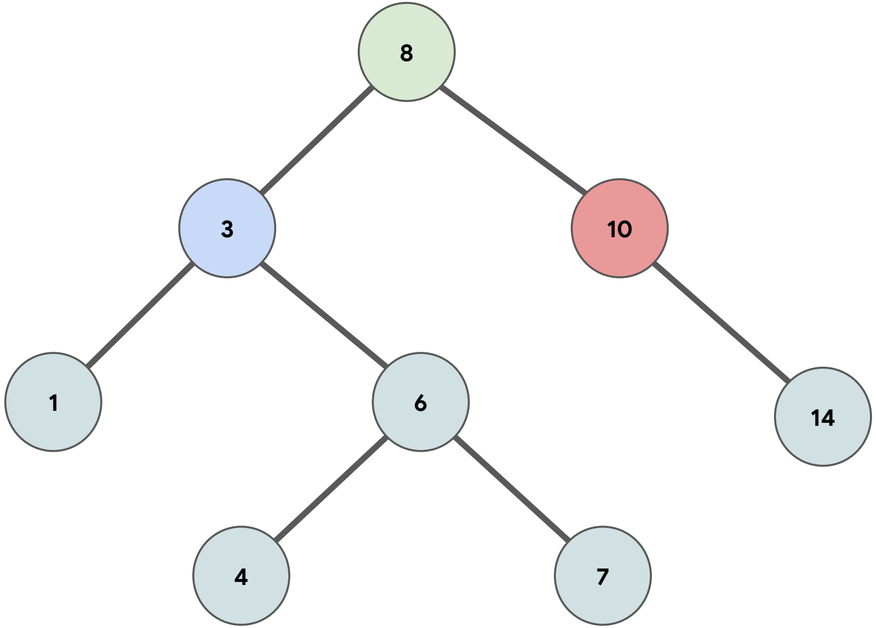
Operation - Deletion



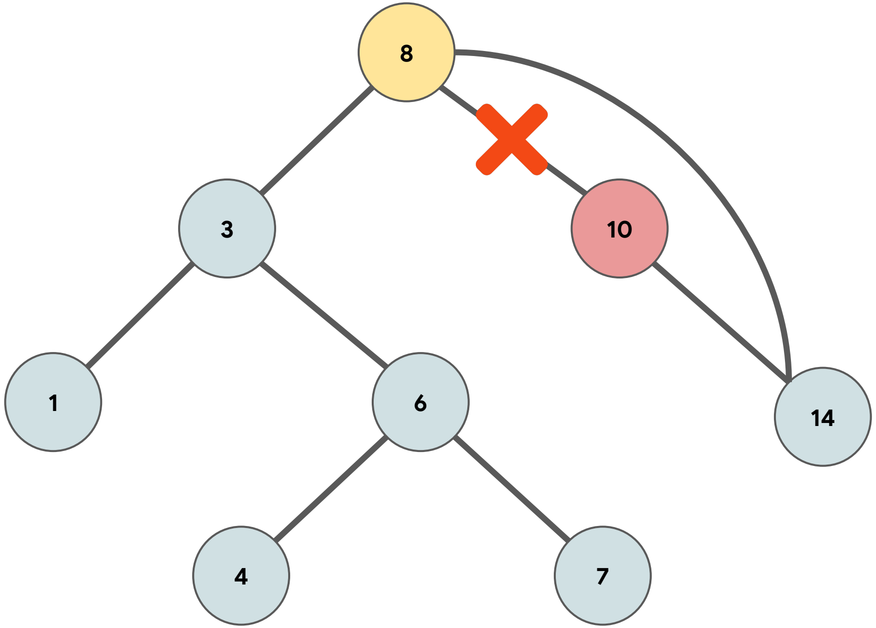
Operation - Deletion

Case Two: In the second case, the node to be deleted lies has a single child node. In such a case follow the steps below:

1. Replace that node with its child node.
2. Remove the child node from its original position.



Operation - Deletion

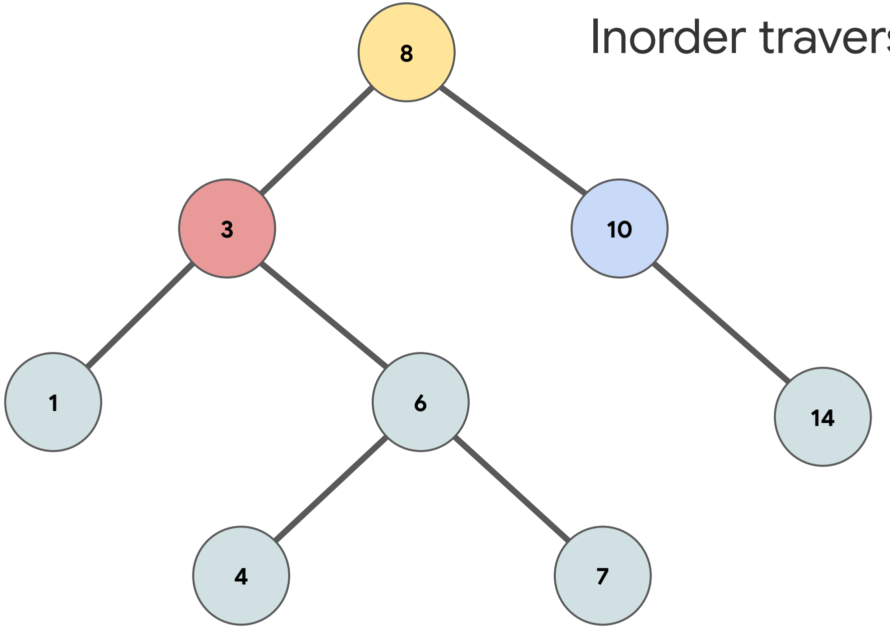


Operation - Deletion

Case Three: The node to be deleted has two children. In such a case follow the steps below:

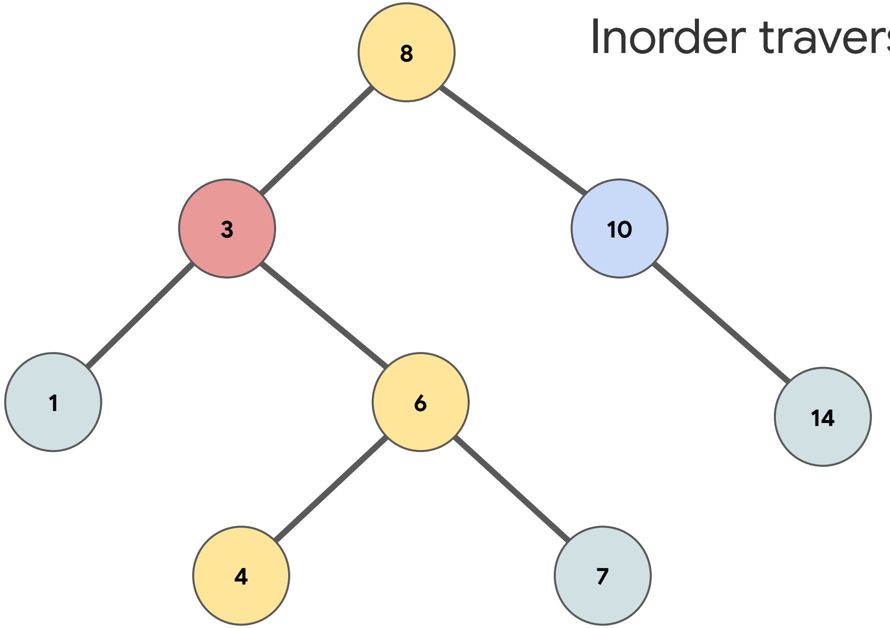
1. Get the **inorder successor** of that node. Why?
2. Replace the node with the inorder successor.
3. Remove the inorder successor from its original position.

Operation - Deletion



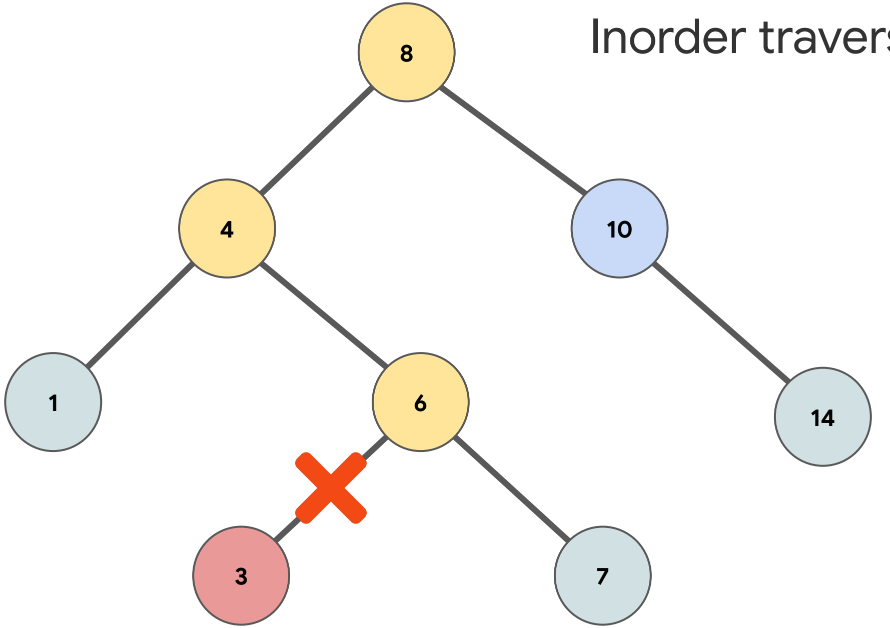
Inorder traversal: 1 3 4 6 7 8 10 14

Operation - Deletion



Inorder traversal: 1 3 **4** 6 7 8 10 14

Operation - Deletion



Inorder traversal: 1 **4** 6 7 8 10 14

Time and Space Complexity Analysis

Binary Tree

- Traversing
  - Time = ?
- Searching
  - Time = ?
- Insertion
  - Time = ?
- Deletion
  - Time = ?
- Space = ?

Binary Search Tree

- Traversing
  - Time = ?
- Searching
  - Time = ?
- Insertion
  - Time = ?
- Deletion
  - Time = ?
- Space = ?

Time and Space Complexity Analysis

Binary Tree

- Traversing
  - Time = $O(n)$
- Searching
  - Time = $O(n)$
- Insertion
  - Time = $O(n)$
- Deletion
  - Time = $O(n)$
- Space = $O(n)$ … **Why?**

Binary Search Tree

- Traversing
  - Time = $O(n)$.
- Searching
  - Time = $O(h)$ where **h** is the height of BST
- Insertion
  - Time = $O(h)$
- Deletion
  - Time = $O(h)$
- Space = $O(n)$

Common Pitfalls

- Null pointer exceptions
- Assuming the tree is balanced
- Wrong choice of traversal
- Wrong recurrence relations and base cases
- Stack overflow

Applications of Trees

- Representation structure in **File Explorer**. (Folders and Subfolders) uses N-ary Tree.
- **Auto-suggestions** when you google something using Trie.
- Used in **decision-based machine learning algorithms**.
- Tree forms the backbone of other complex data structures like heap, priority queue, spanning tree, etc.
- A binary tree is used in **database indexing** to store and retrieve data in an efficient manner.
- **Binary Search Trees (BST)** can be used in **sorting algorithms**.

Practice Questions

- Merge Two Binary Trees
- Search in Binary Search Trees
- Same Tree
- Lowest Common Ancestor of Binary Search Tree
- Validate Binary Search Trees
- Binary Tree Zigzag Level Order Traversal
- Maximum Difference Between Node and Ancestor
- Kth smallest Element in BST
- Maximum Sum BST in Binary Tree

Quote of the Day

"A tree with strong roots laughs at storms."
- Malay Proverb