Algorithms and Data Structures for Data Science

Binary Search Trees

CS 277

March 22, 2023

Brad Solomon

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Review trees and binary trees

Discuss search on binary trees

Extend binary trees into binary *search* trees

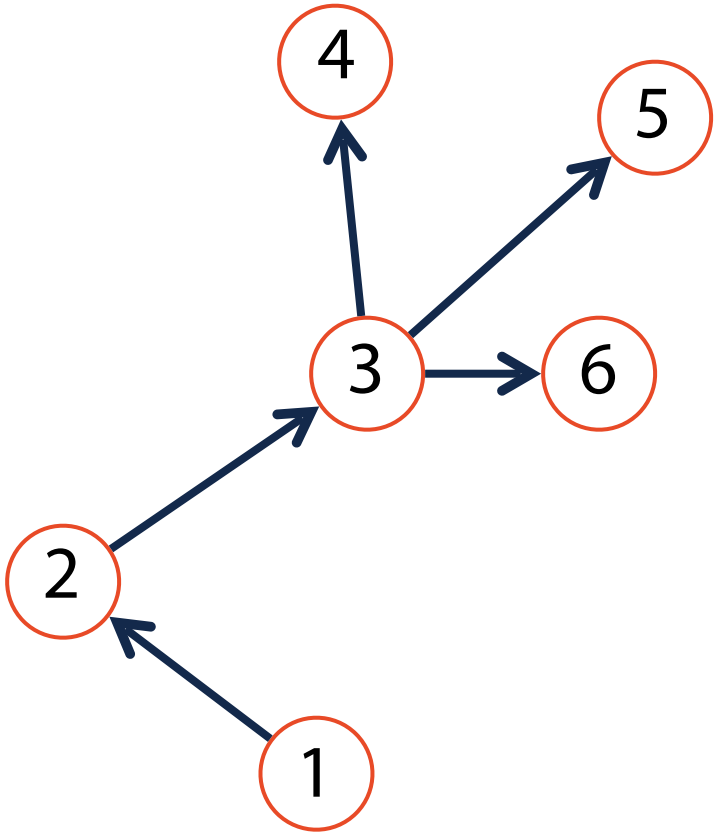
Trees

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

(In CS 277) a tree is also:

1) Acyclic — contains no cycles

2) Rooted — root node connected to all nodes



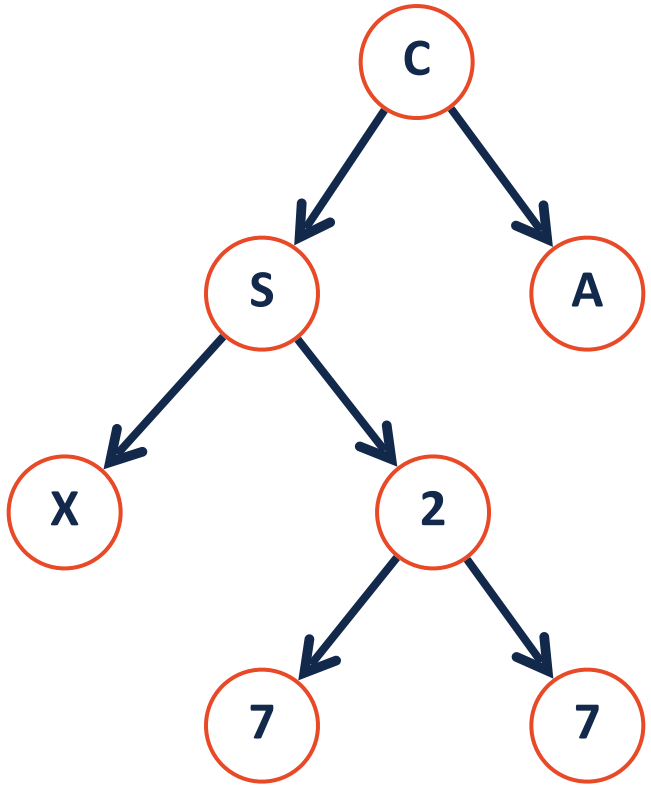
Binary Tree

A **binary tree** is a tree T such that:

$$T = None$$

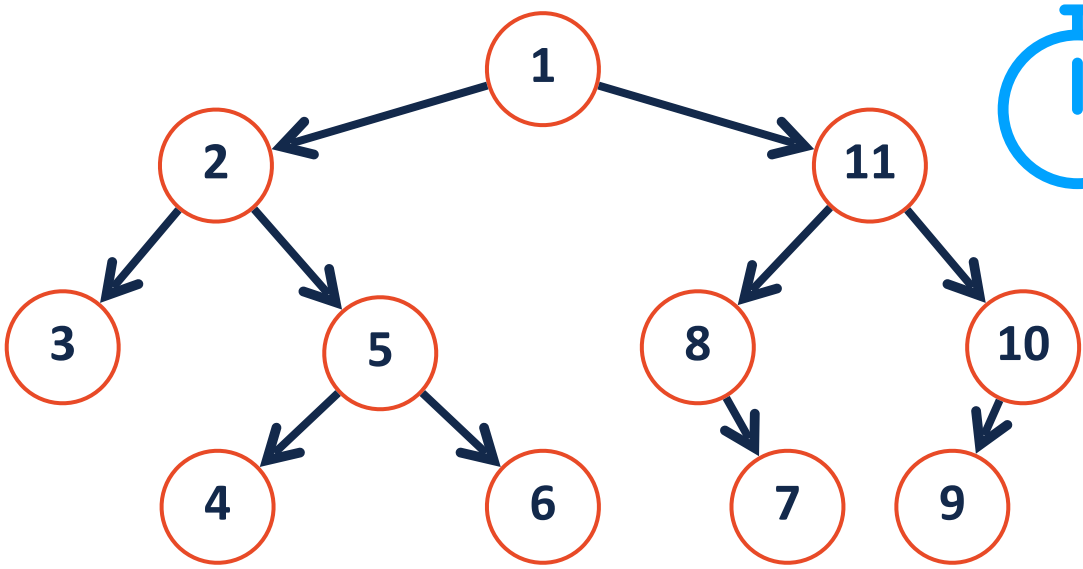
or

$$T = treeNode(val, T_L, T_R)$$




```
1  class treeNode:
2      def __init__(self, val, left=None, right=None):
3          self.val = val
4          self.left = left
5          self.right = right
```

Tree Traversals



Pre-order:

In-order:

Post-order:

Tree Abstract Data Type

What is a tree? What properties does it have? What functions?

Tree ADT

Insert: Add an object into tree

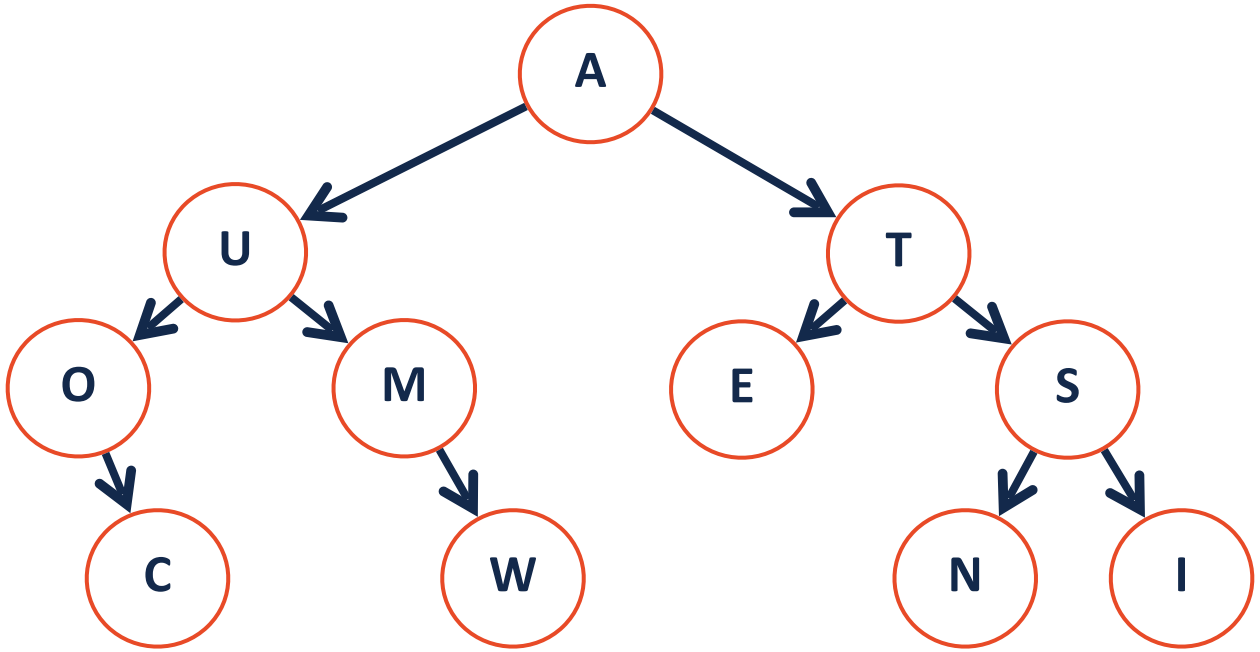
Remove: Remove a specific object from tree

Traverse: Visit every node in tree (all objects)

Search: Find a specific object in the tree

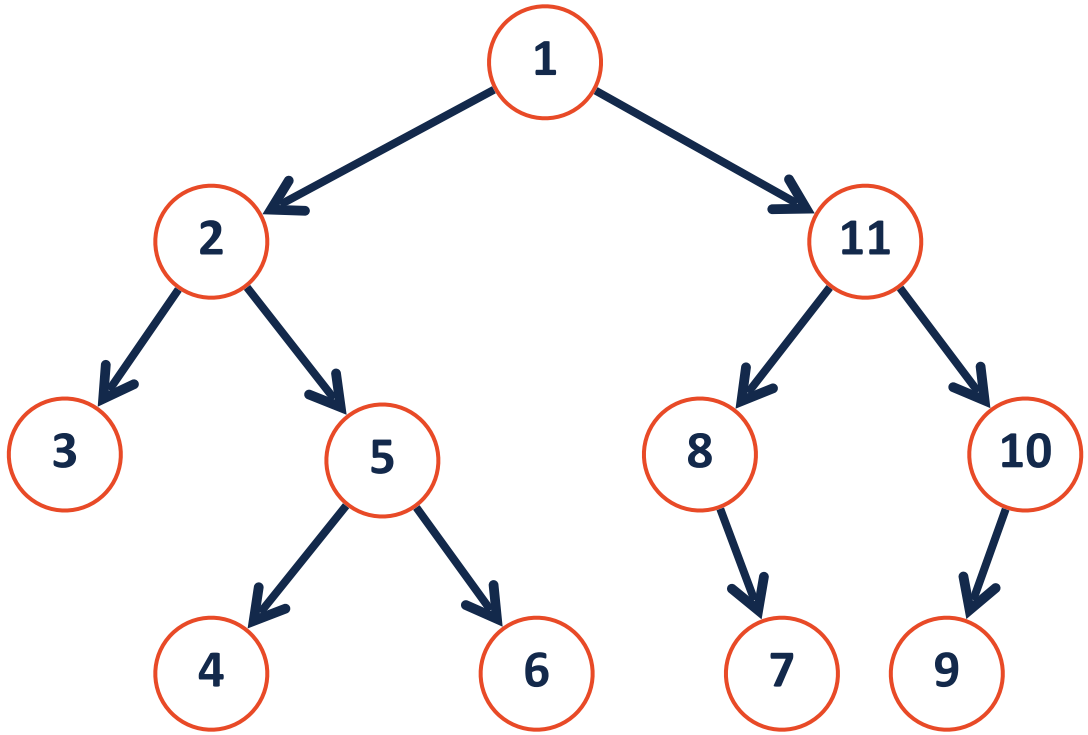
Searching a Binary Tree

There are two main approaches to searching a binary tree:



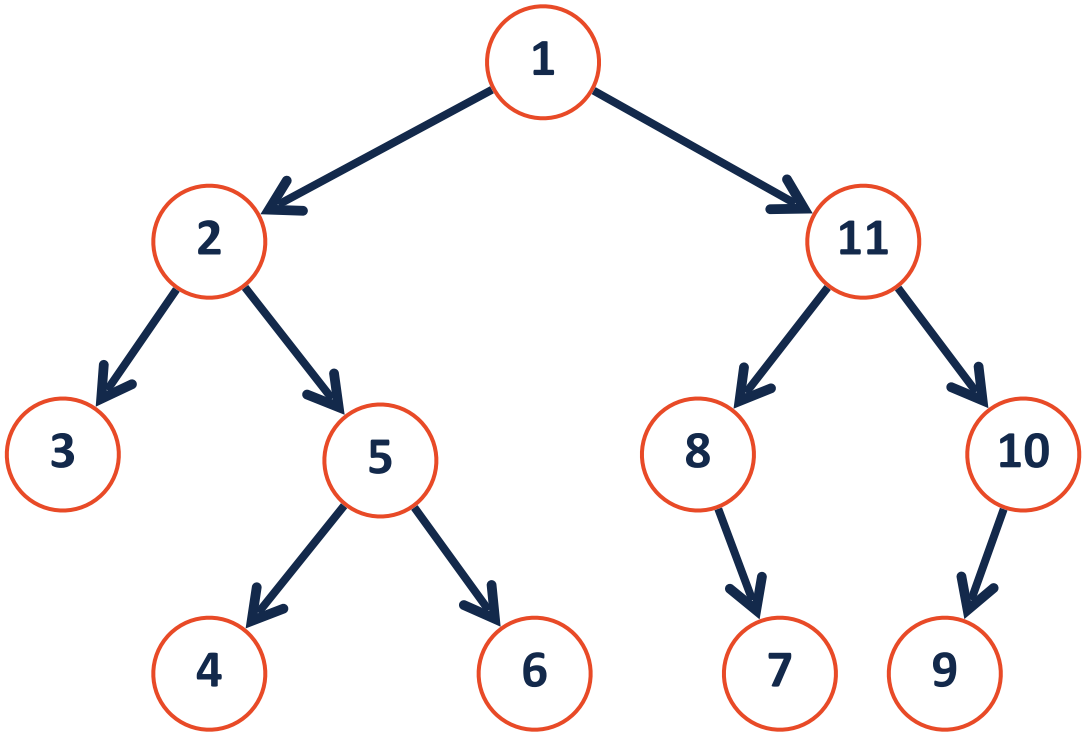
Depth First Search

Explore as far along one path as possible before backtracking



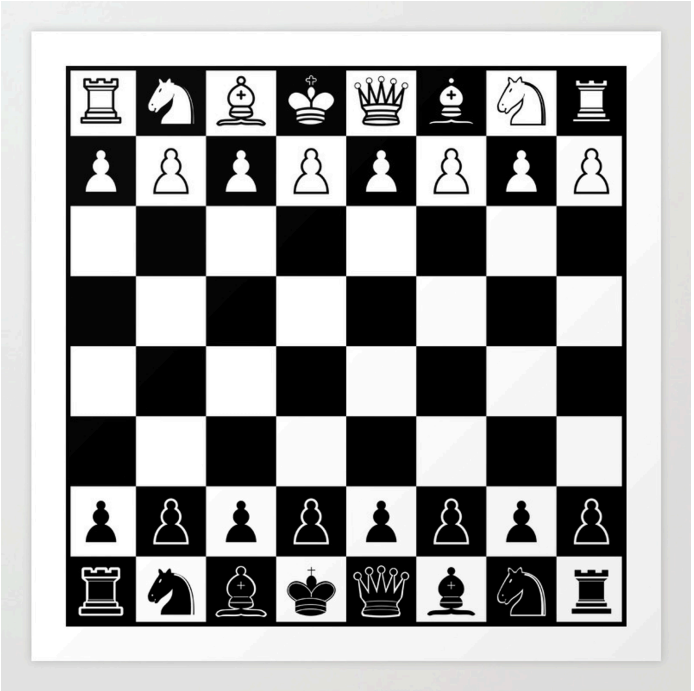
Breadth First Search

Fully explore depth i before exploring depth i+1



What search algorithm is best?

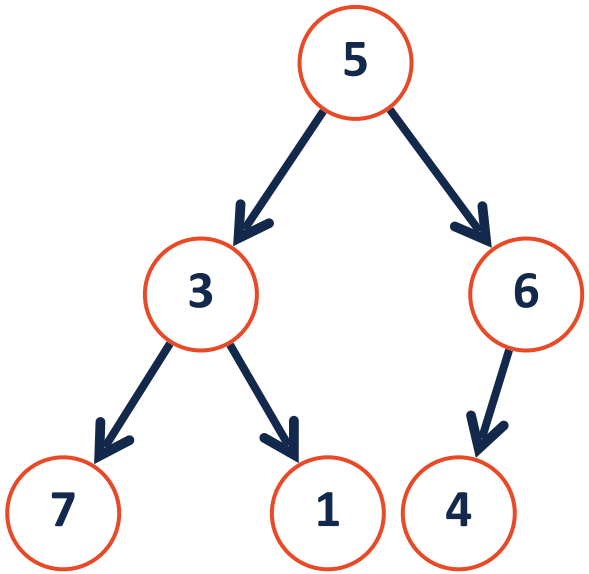
The average ‘branch factor’ for a game of chess is ~31. If you were searching a decision tree for chess, which search algorithm would you use?

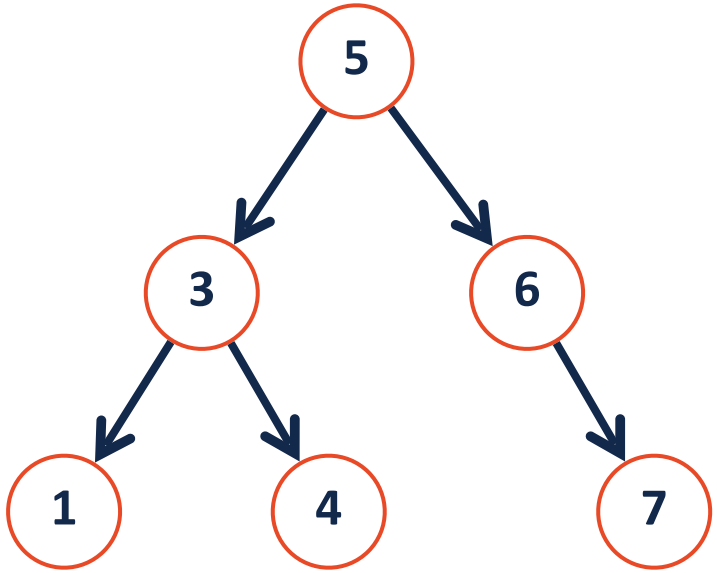


Improved search on a binary tree

5	3	6	7	1	4

1	3	4	5	6	7



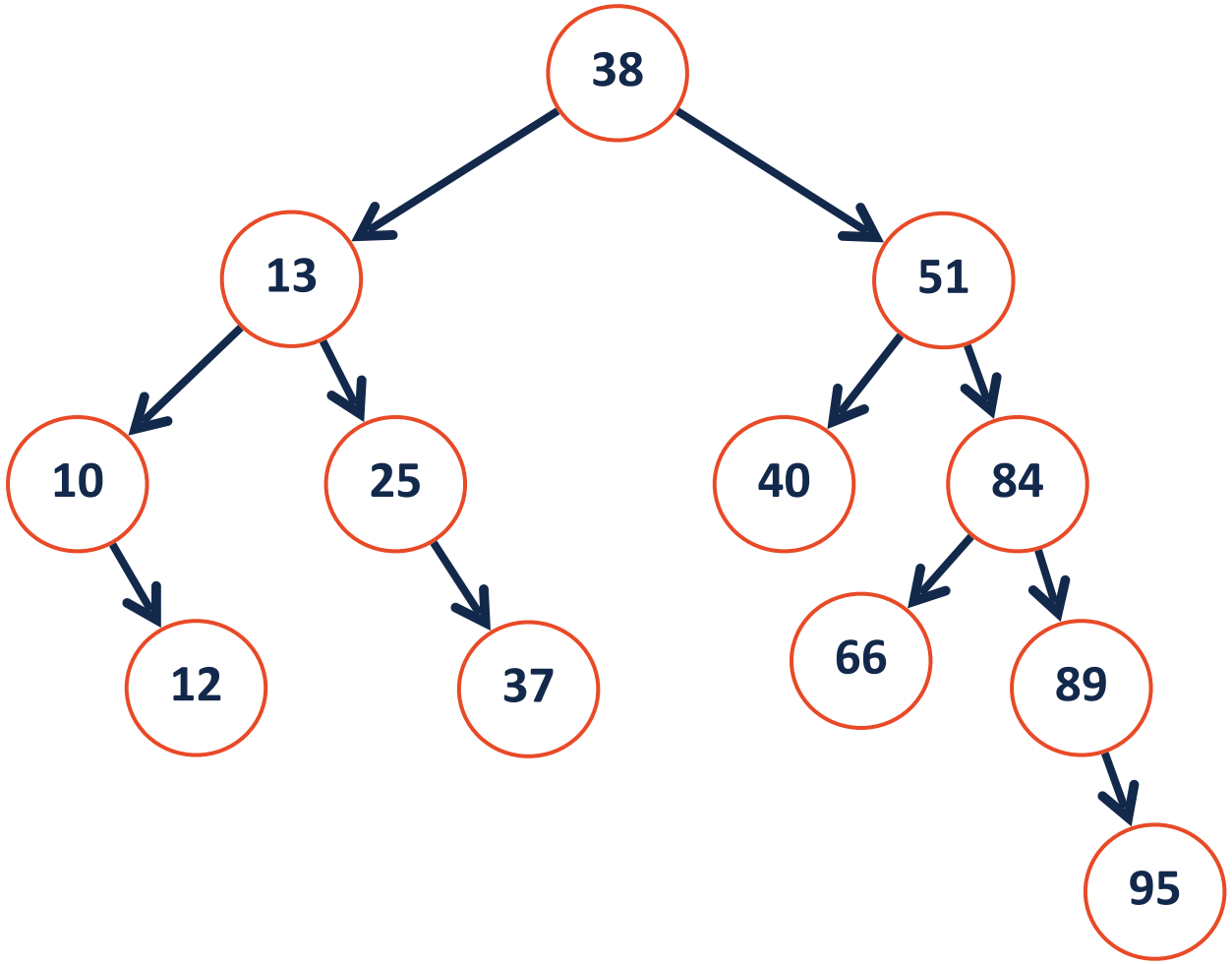


Binary Search Tree (BST)

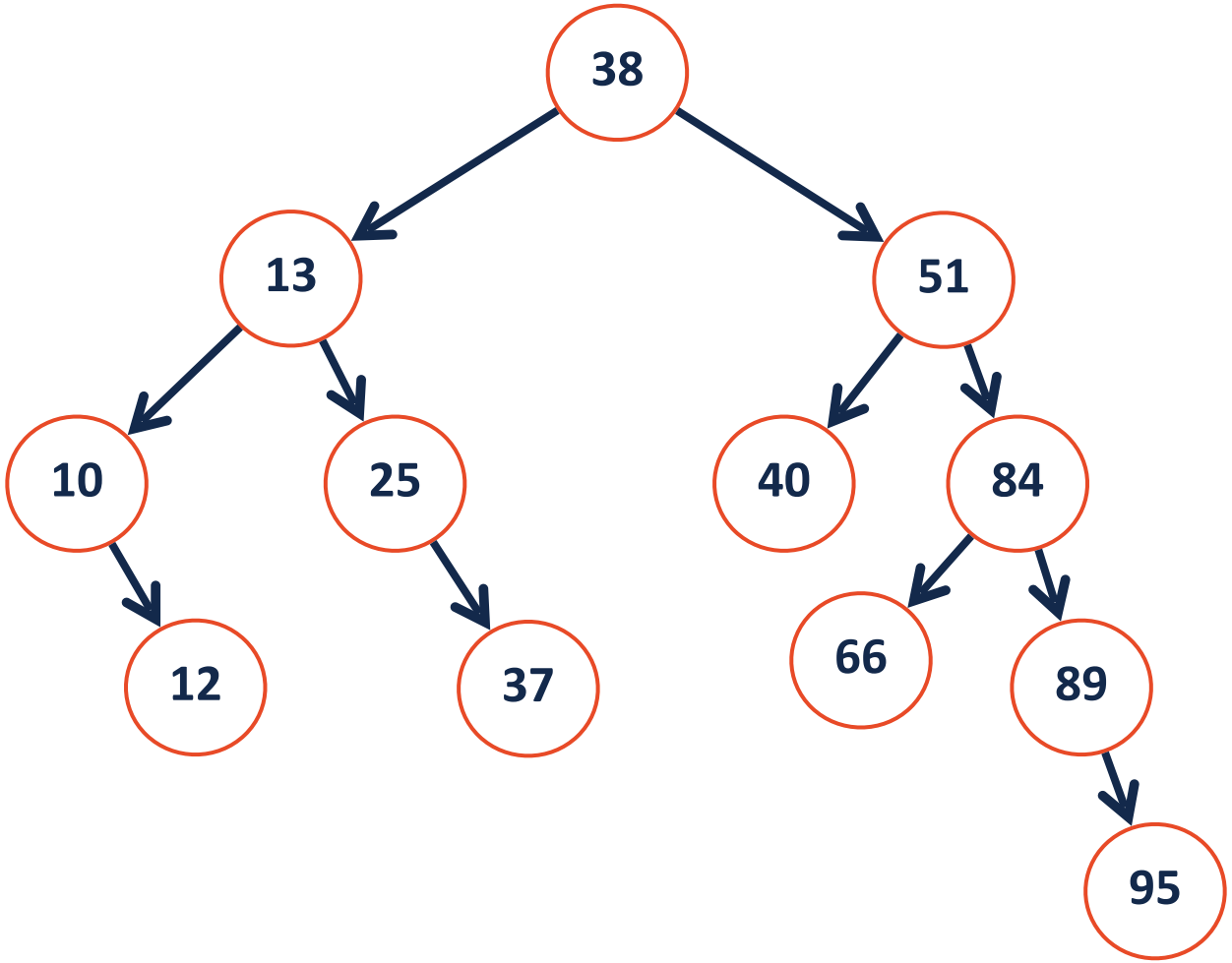
A **BST** is a binary tree $T = treeNode(val, T_L, T_r)$ such that:

$$\forall n \in T_L,\ n.val < T.val$$

$$\forall n \in T_R,\ n.val > T.val$$



BST In-Order Traversal



Dictionary ADT

Data is often organized into key/value pairs:

Word → Definition

Course Number → Lecture/Lab Schedule

Node → Edges

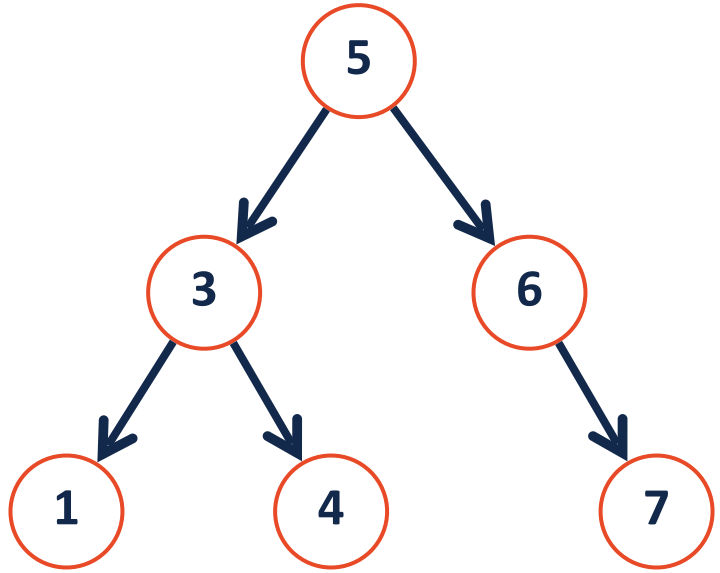
Flight Number → Arrival Information

URL → HTML Page

Average Image Color → File Location of Image

Binary Search Tree

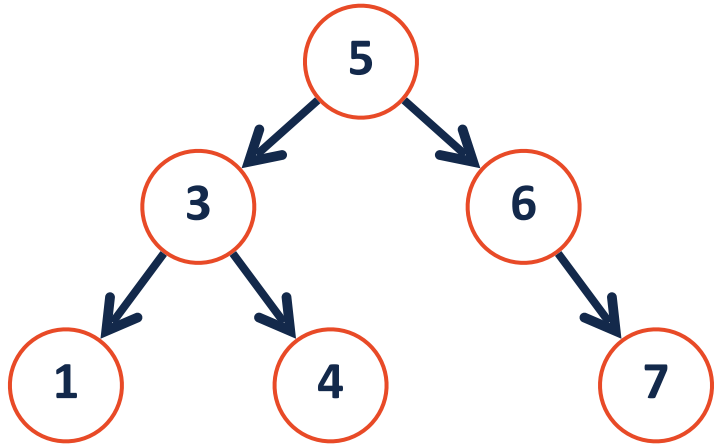
```
1  class bstNode:
2      def __init__(self, key, val, left=None, right=None):
           self.key = key
3          self.val = val
4          self.left = left
5          self.right = right
```



Key	5	3	6	7	1	4
Value	A	B	C	D	E	F

BST Insert

Base Case:

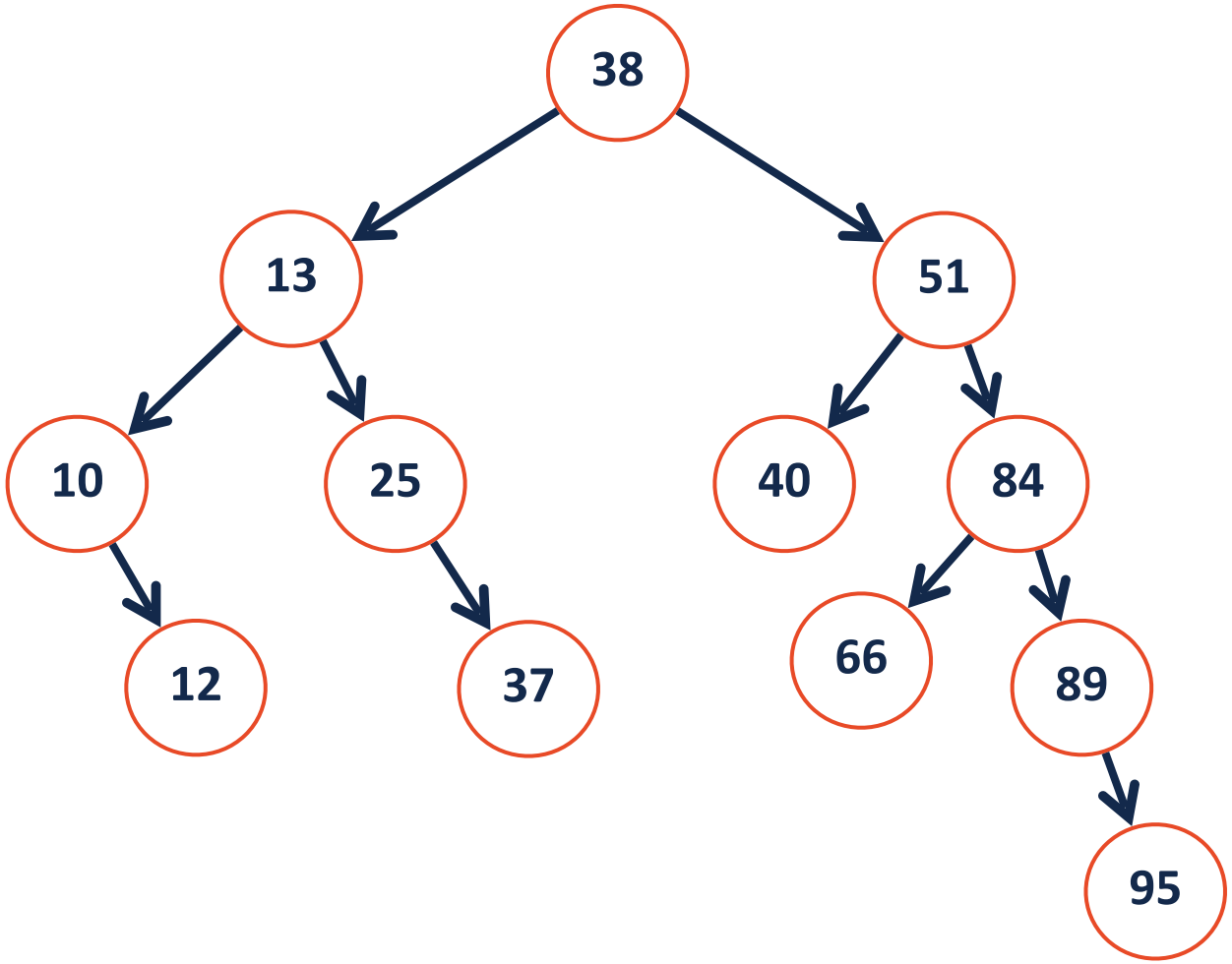


Recursive Step:

Combining:

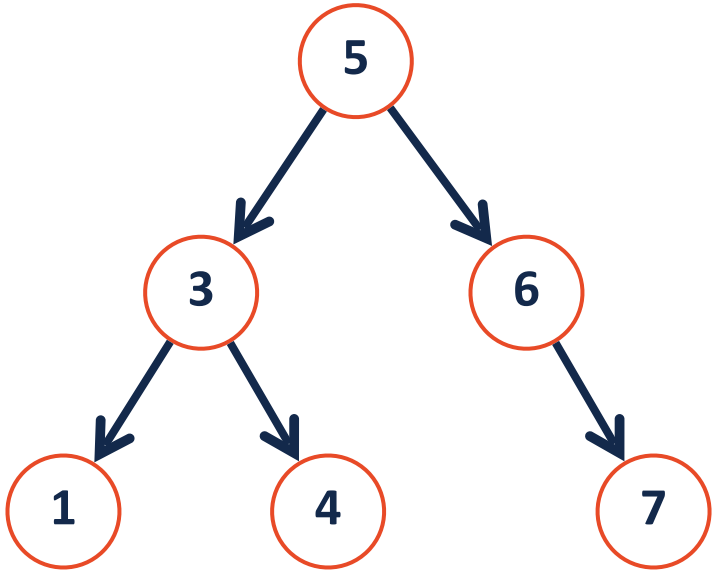
BST Insert

`insert(33)`



BST Insert

```
1  def insert(root, key, value):
2      if root == None:
3          root = bstNode(key,value)
4      else:
5          insert_helper(root, key, value)
6      return root
7
8  def insert_helper(node, key, value):
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

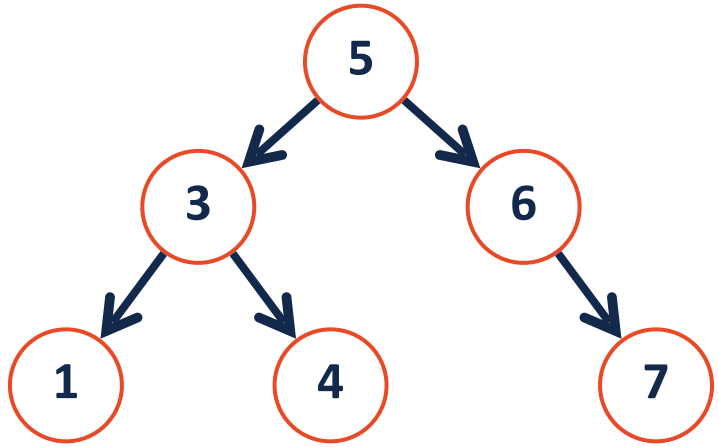


BST Insert

What binary would be formed by inserting the following sequence of integers: `[3, 7, 2, 1, 4, 8, 0]`

BST Find

Base Case:

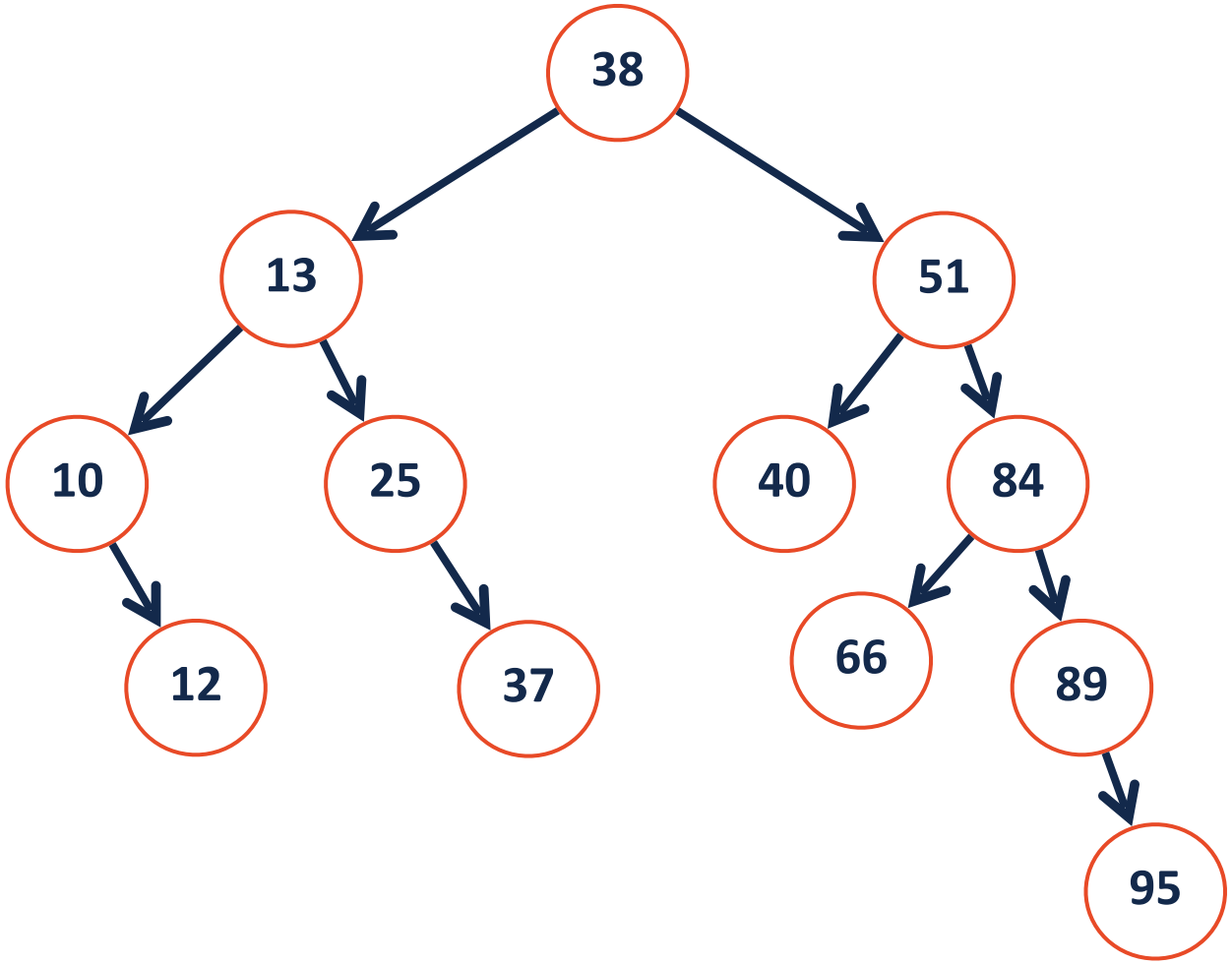


Recursive Step:

Combining:

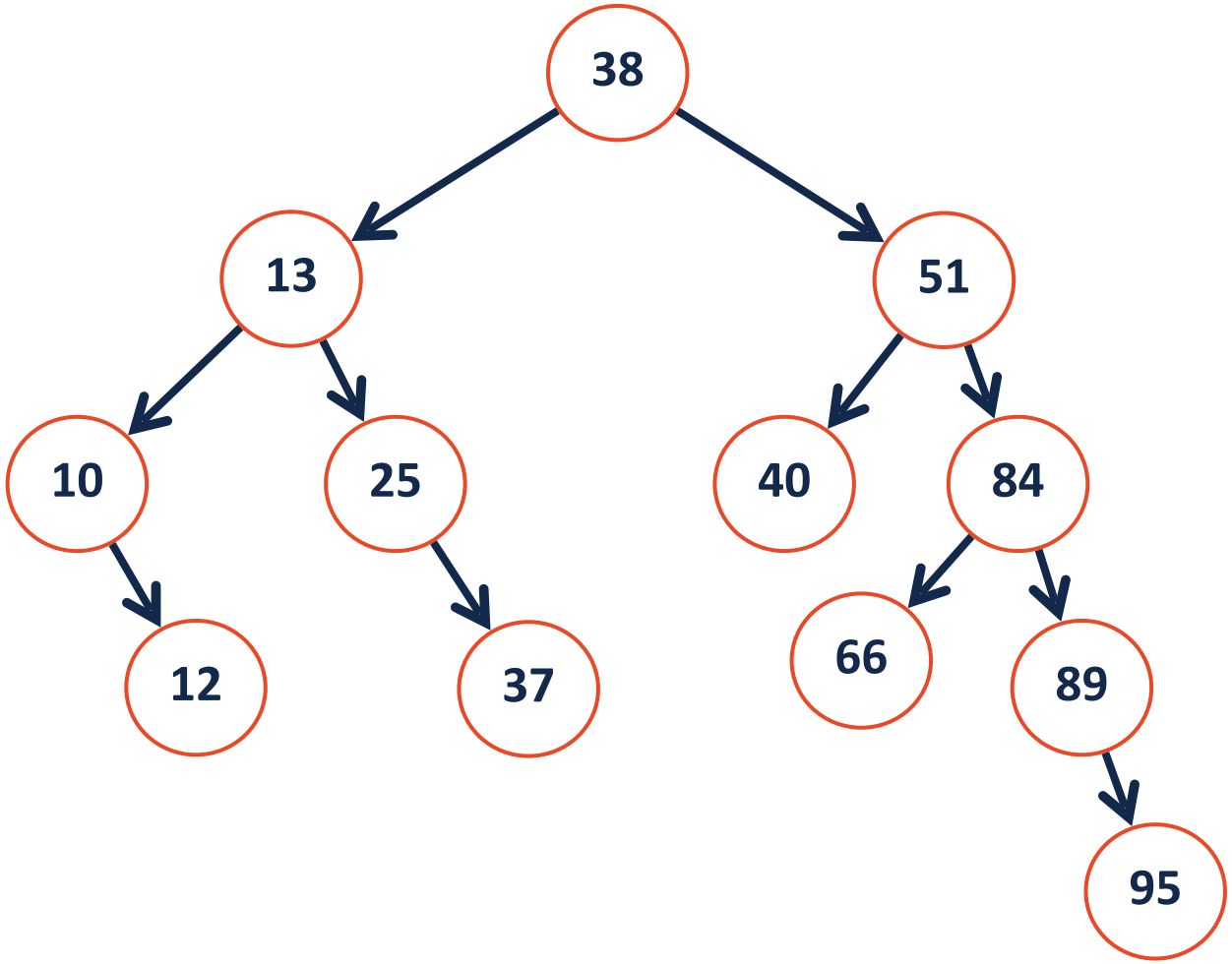
BST Find

`find(66)`



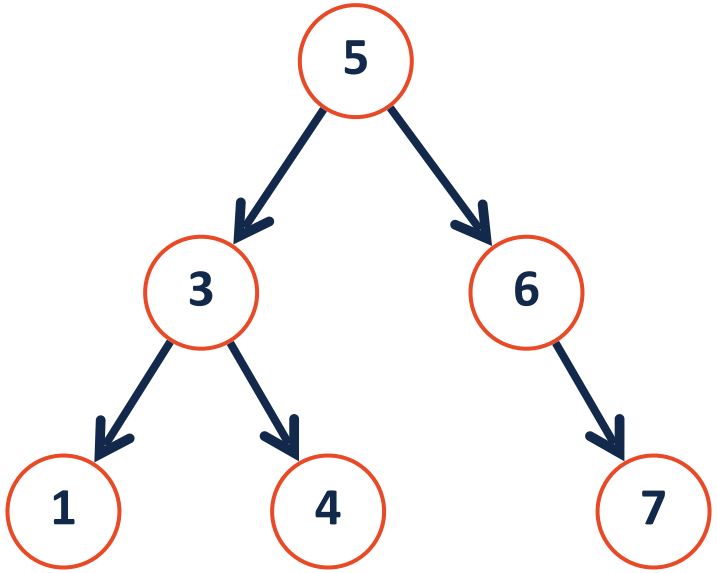
BST Find

`find(9)`



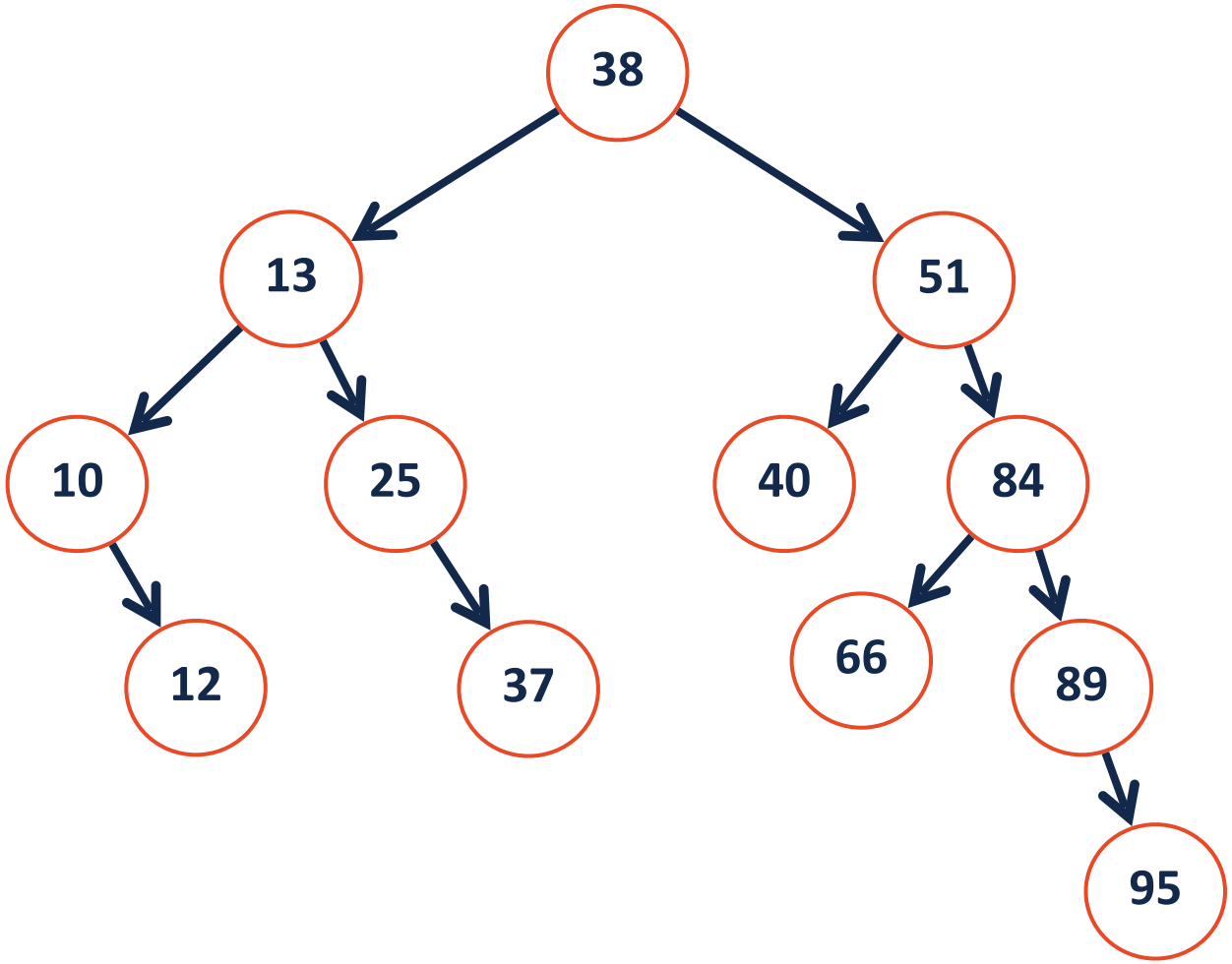
BST Find

```
1  def find(root, key):
2      return find_helper(root, key).val
3
4  def find_helper(node,key):
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```



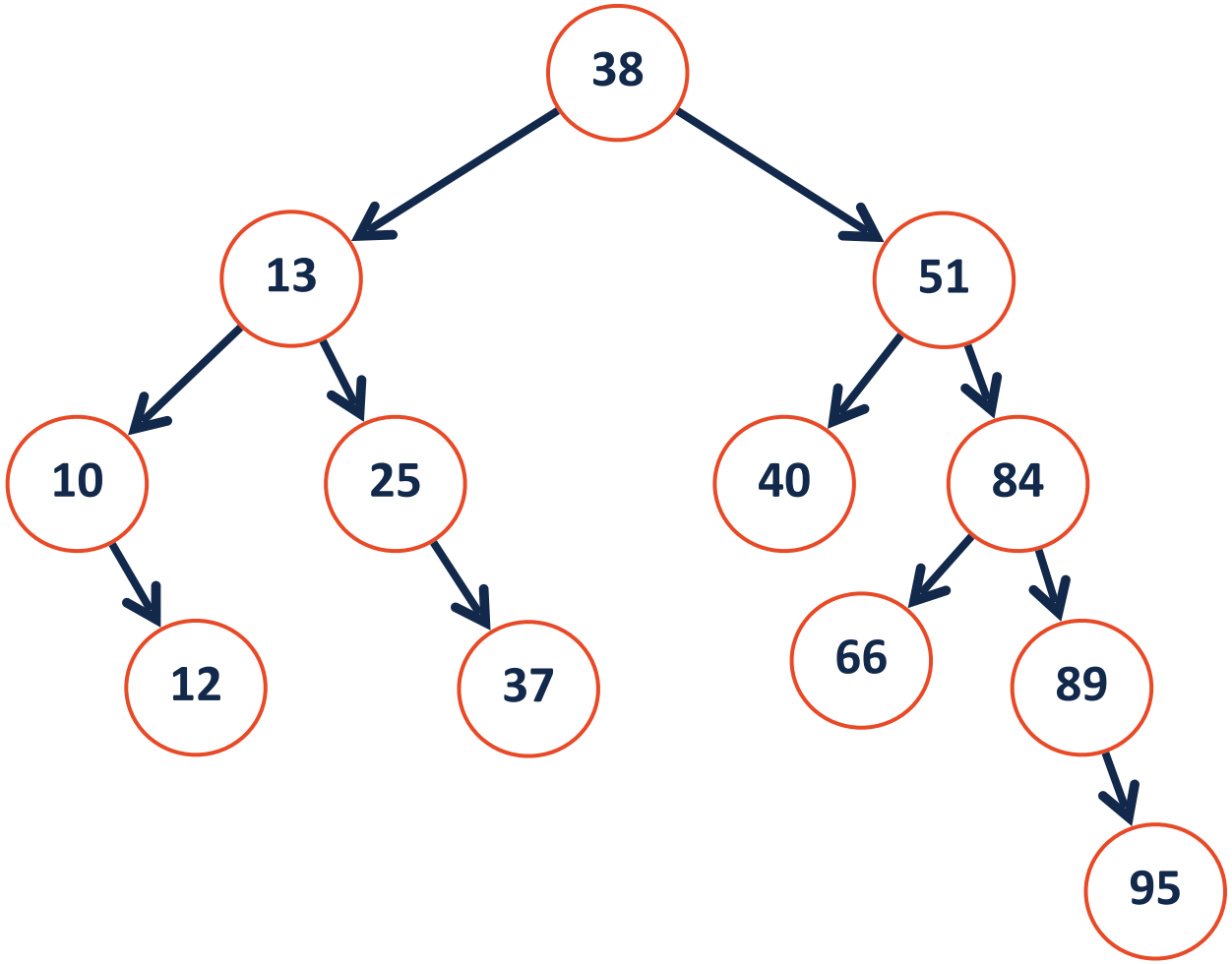
BST Remove

`remove(40)`



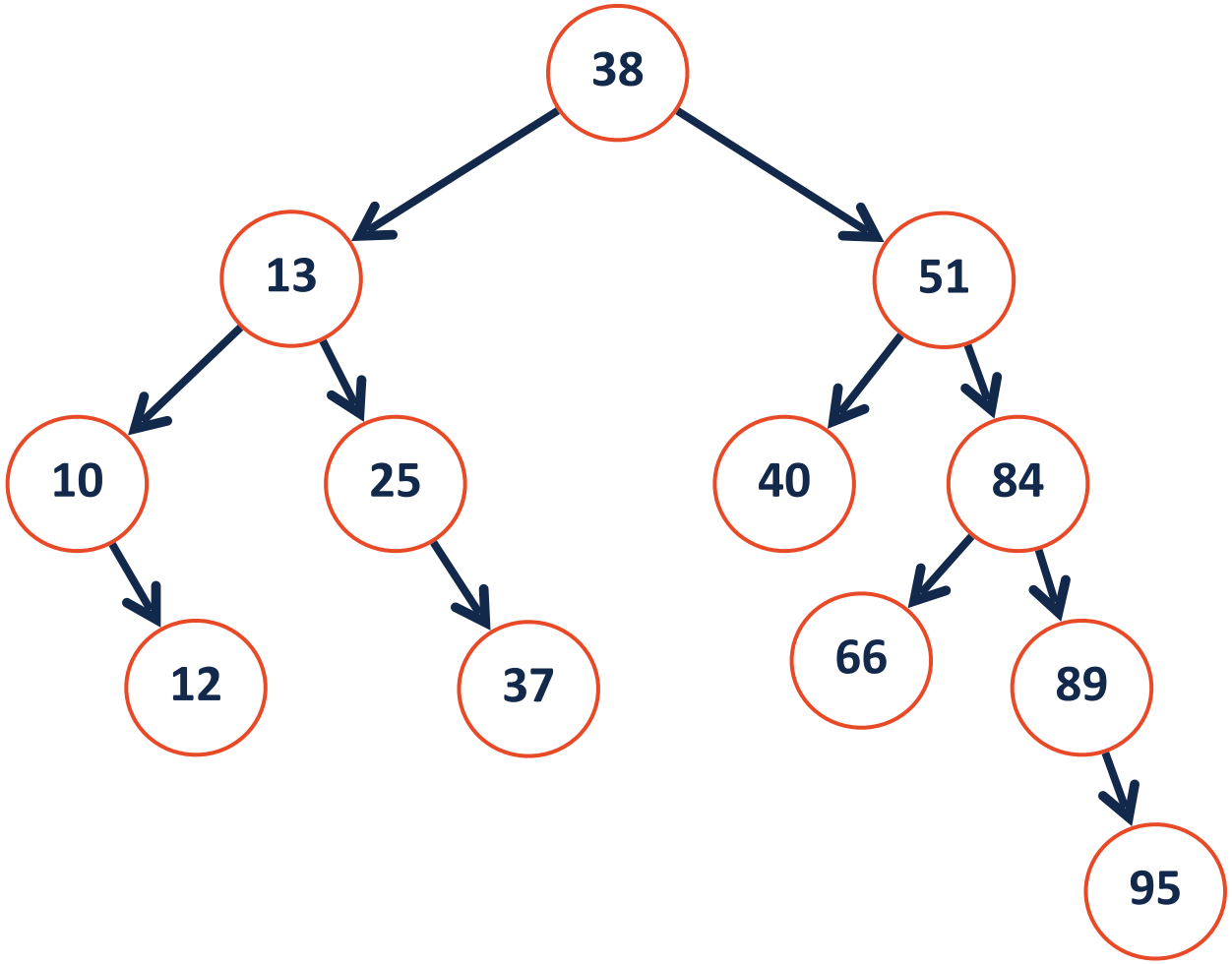
BST Remove

`remove(25)`



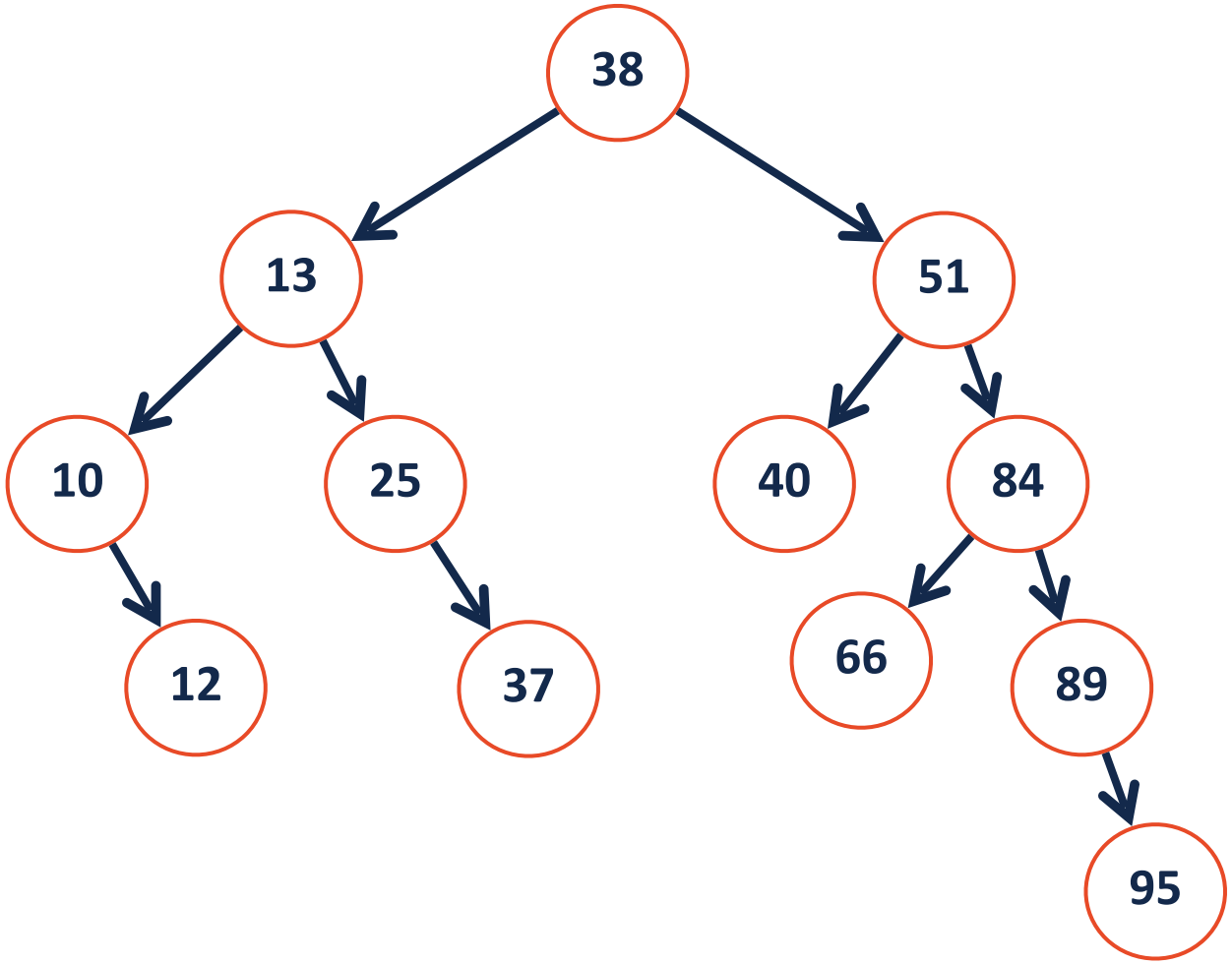
BST Remove

`remove(13)`



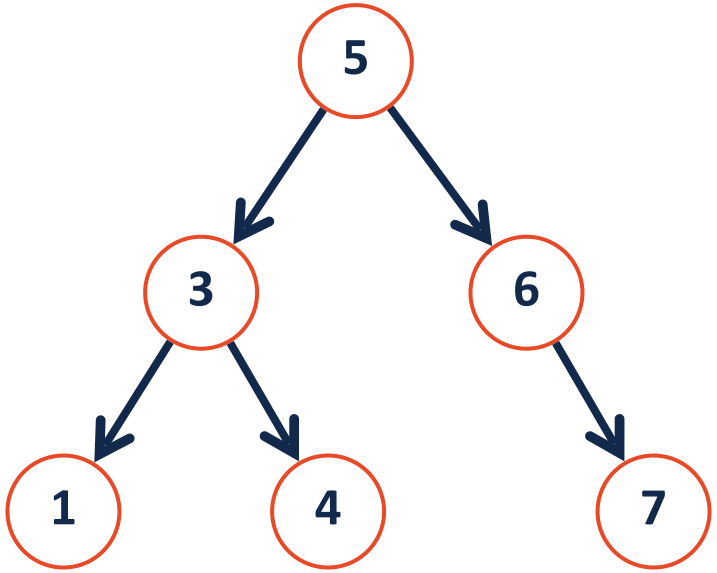
BST Remove

`remove(51)`



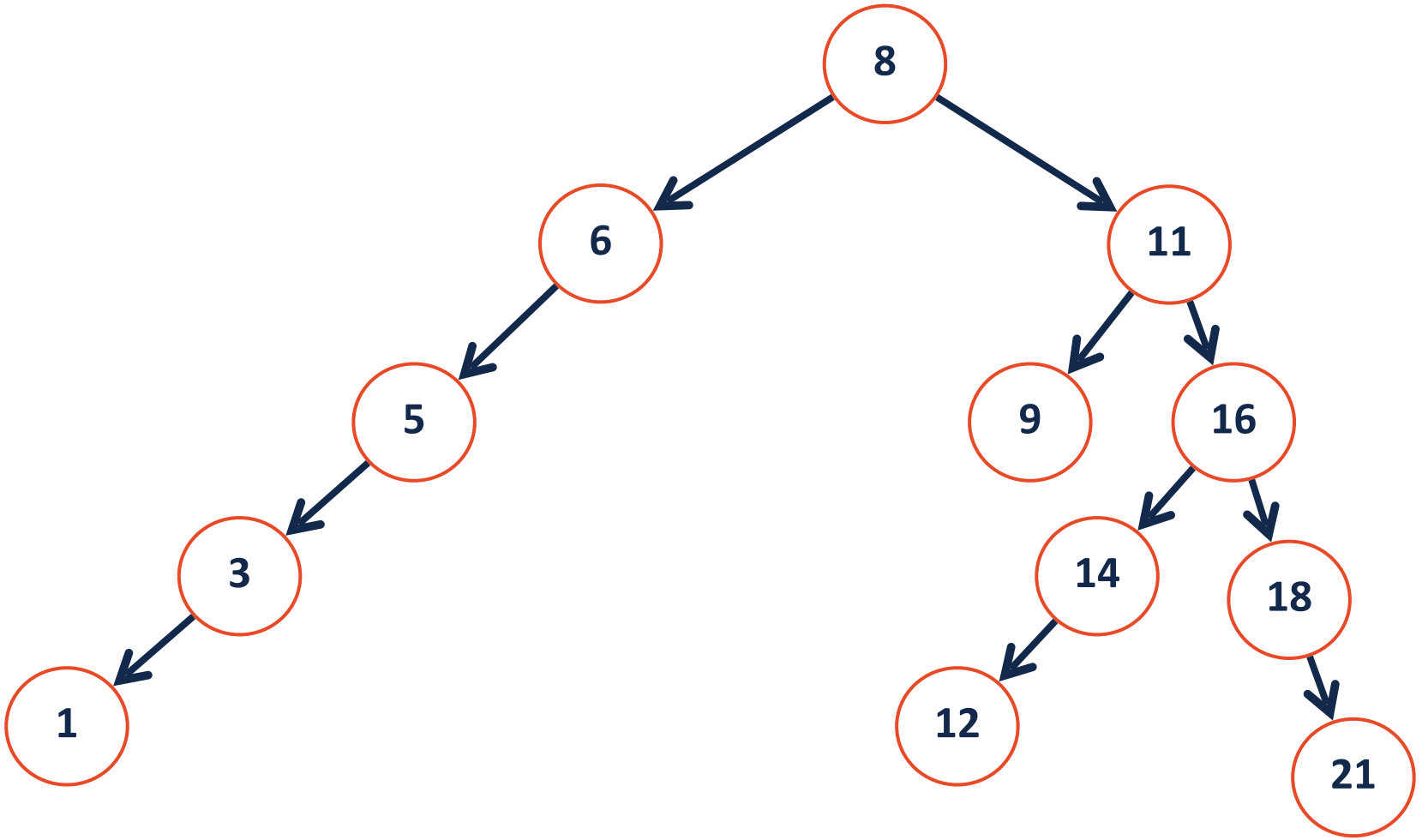
BST Remove

```
1  def remove(root, key):
2      root = remove_helper(root, key)
3      return root
4
5  def remove_helper(node, key):
6
7
8
9
10
11
12
13
14
15
16
17
18
19 def findIOP(node):
20     pass
21
22 def findIOS(node):
23     pass
```



BST Remove

What will the tree structure look like if we remove node 16 using IOS?



BST Analysis – Running Time

Operation	BST Worst Case
find	
insert	
delete	
traverse	