Data Structures

Binary Search Trees

CS 225

Brad Solomon

September 20, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

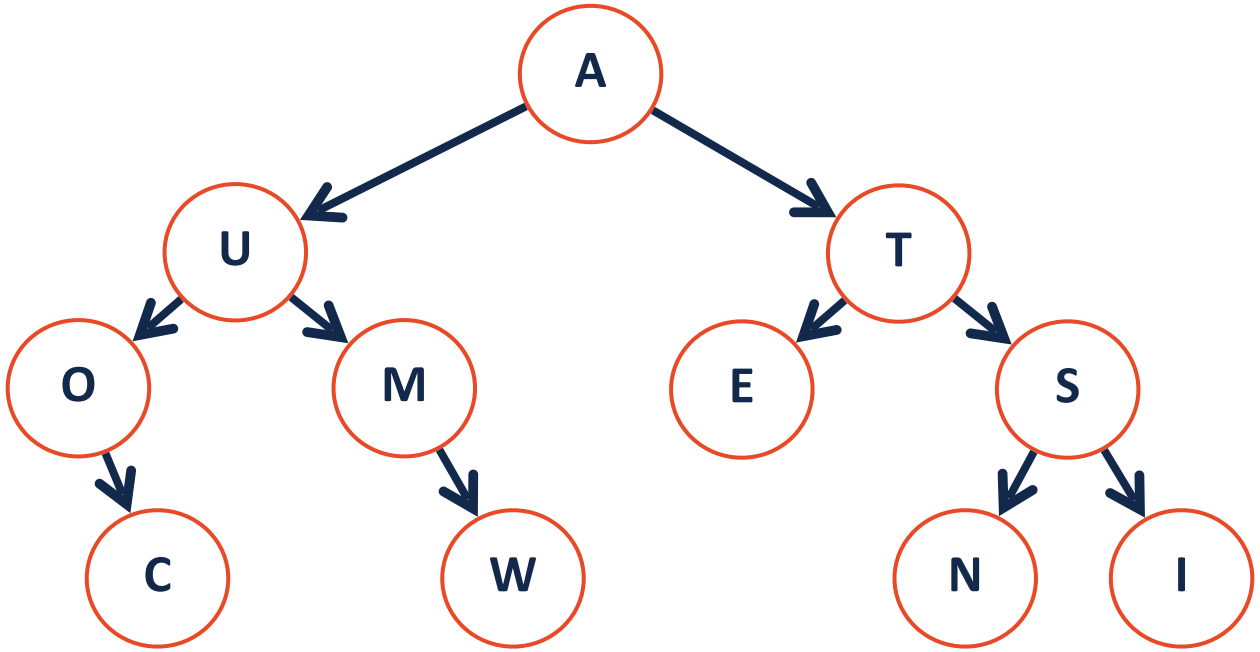
Learning Objectives

Review binary trees

Extend binary trees into binary *search* trees

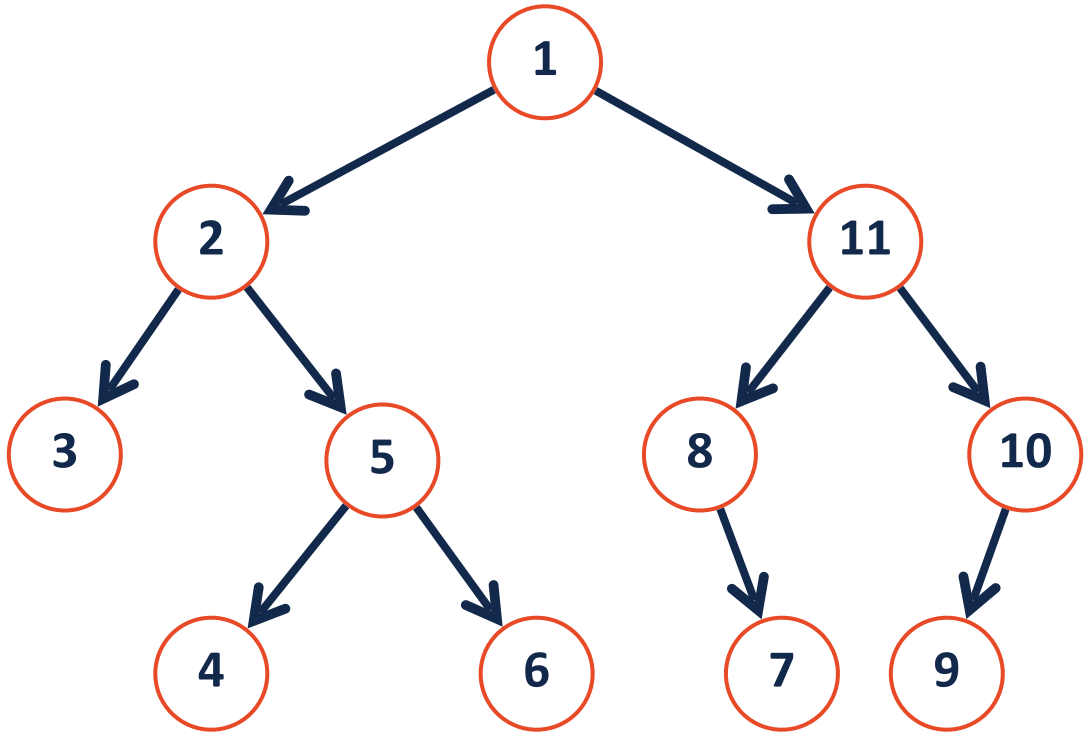
Tree Search

There are two main approaches to searching a binary tree:



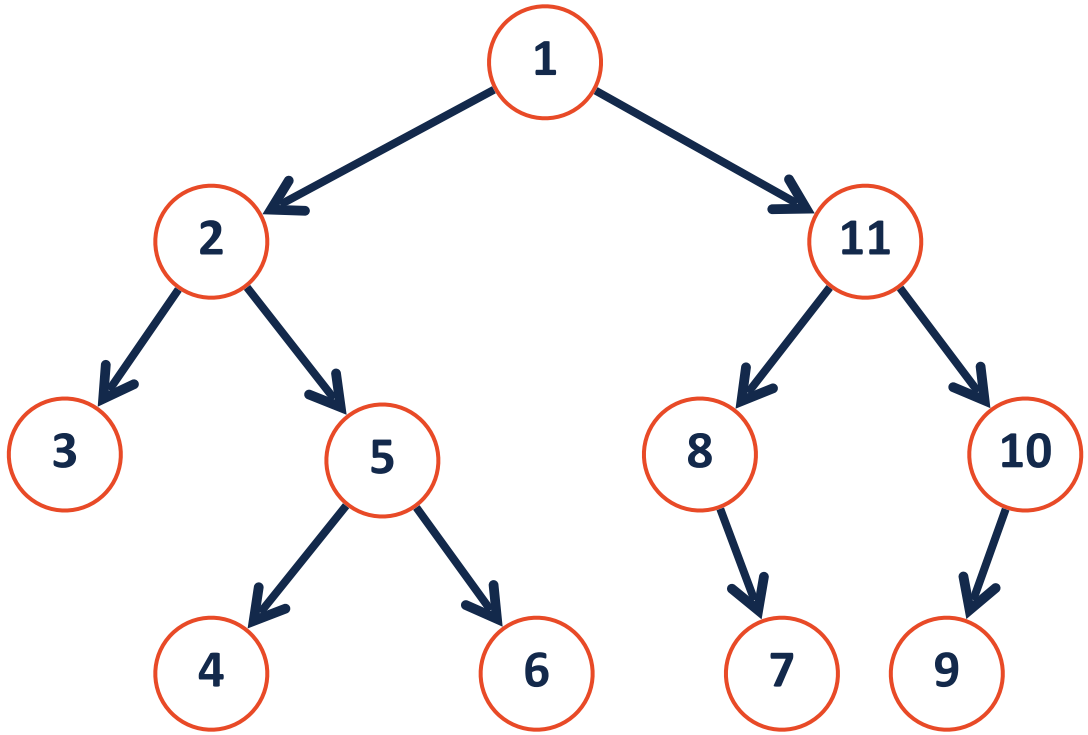
Depth First Search

Explore as far along one path as possible before backtracking

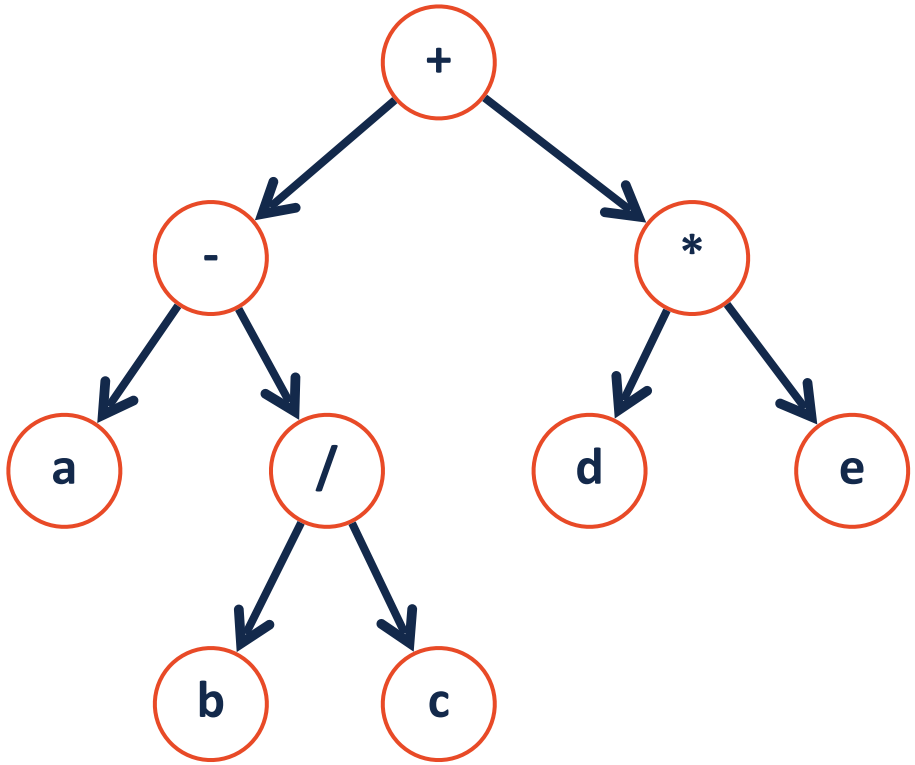


Breadth First Search

Fully explore depth i before exploring depth i+1



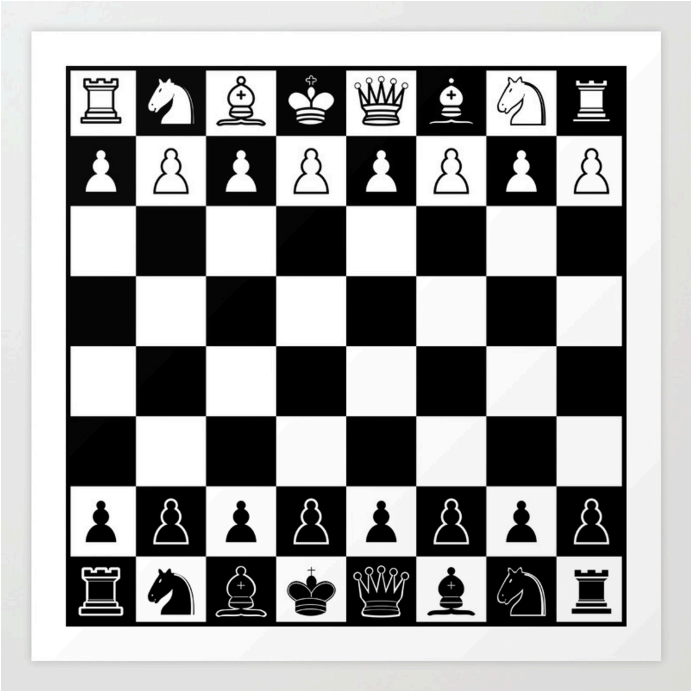
Level-Order Traversal




```
template<class T>
void BinaryTree<T>::lOrder(TreeNode * root)
{

    Queue<TreeNode*> q;
    q.enqueue(root);

    while( q.empty() == False){

        TreeNode* temp = q.head();
        process(temp);

        q.dequeue();

        q.enqueue(temp->left);
        q.enqueue(temp->right);

    }
}
```

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?



Tree Search

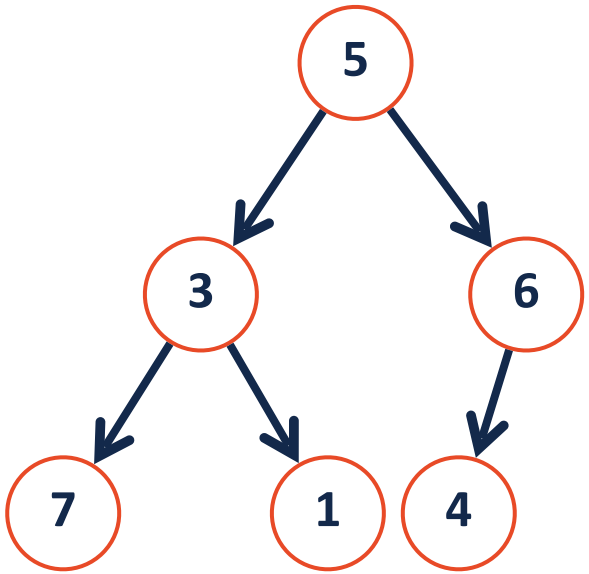
How can we improve our ability to search a binary tree?

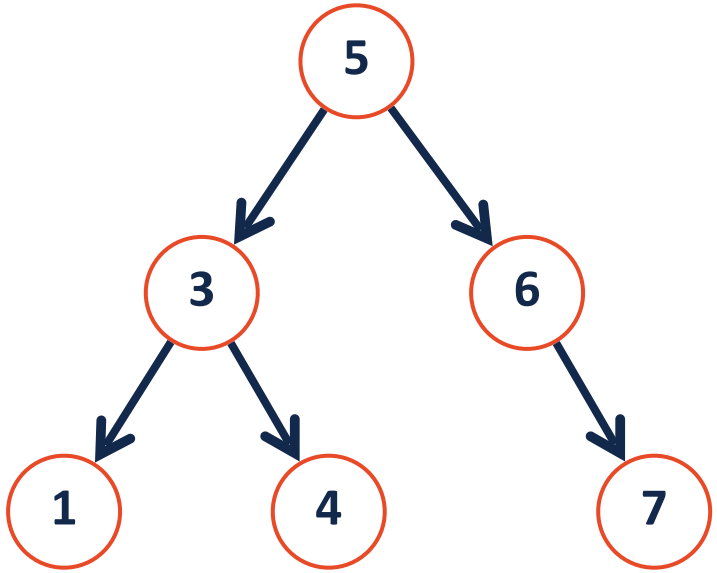
What do we trade in order to do so?

Improved search on a binary tree

5	3	6	7	1	4

1	3	4	5	6	7





Dictionary ADT

Data is often organized into key/value pairs:

Word → Definition

Course Number → Lecture/Lab Schedule

Node → Incident Edges

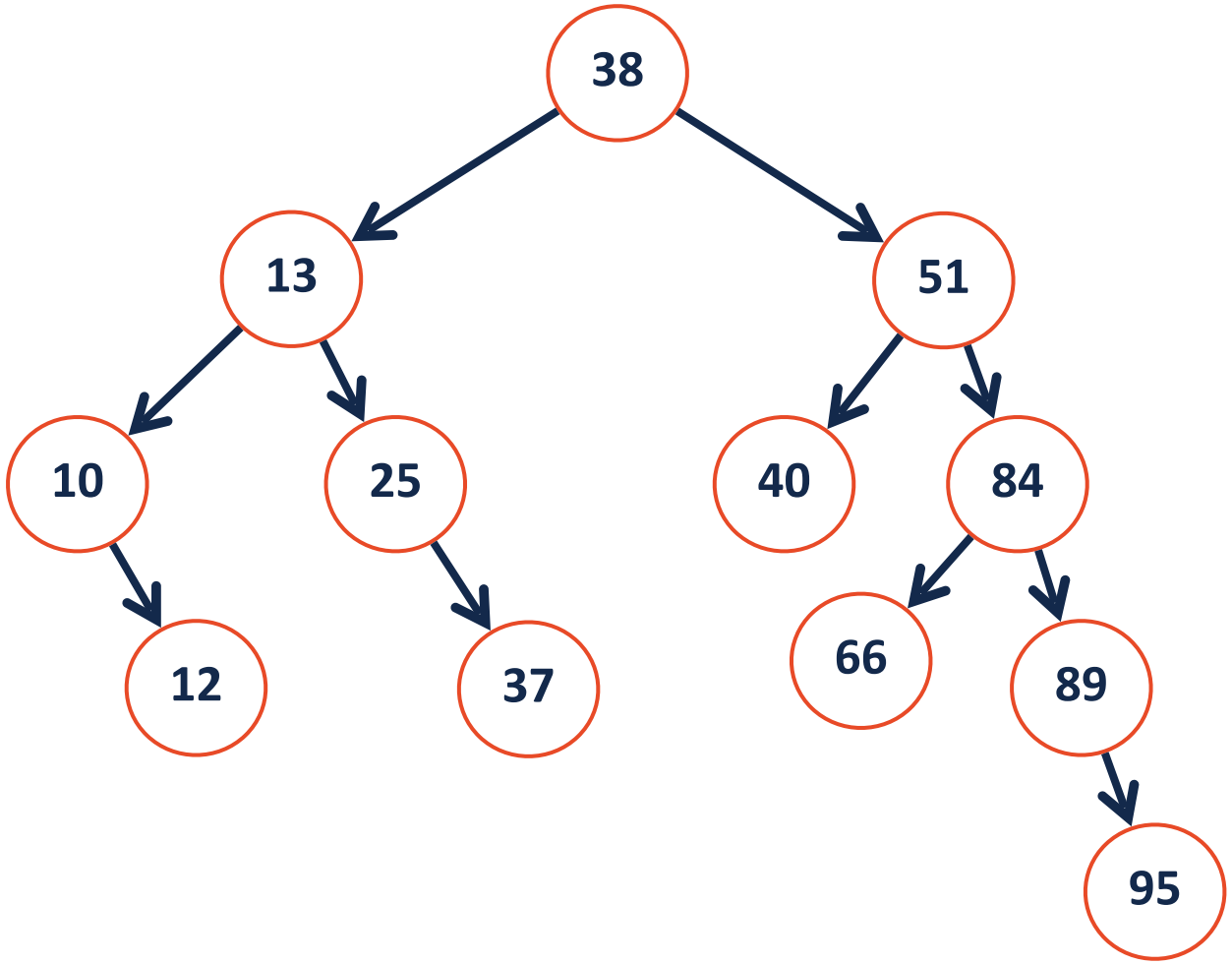
Flight Number → Arrival Information

URL → HTML Page

…

Binary Search Tree (BST)

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



BST.h

```
#pragma once

template <typename K, typename V>
class BST {
  public:
   /* ... */
  private:
     class TreeNode {
        K & key;
        V & value;

        TreeNode *left, *right;

        TreeNode(K & k, V & v) :
key(k), value(v),
left(NULL), right(NULL) { }
     };

     TreeNode *root_;
    /* ... */
};
```

Tree.h

```
#pragma once

template <typename T>
class BinaryTree {
  public:
     /* ... */
  private:
     class TreeNode {
        T & data;

        TreeNode * left;

        TreeNode * right;

        TreeNode(T & data) :
         data(data), left(NULL),
right(NULL) { }

     };

     TreeNode *root_;
     /* ... */
};
```

Binary Search Tree ADT

Insert

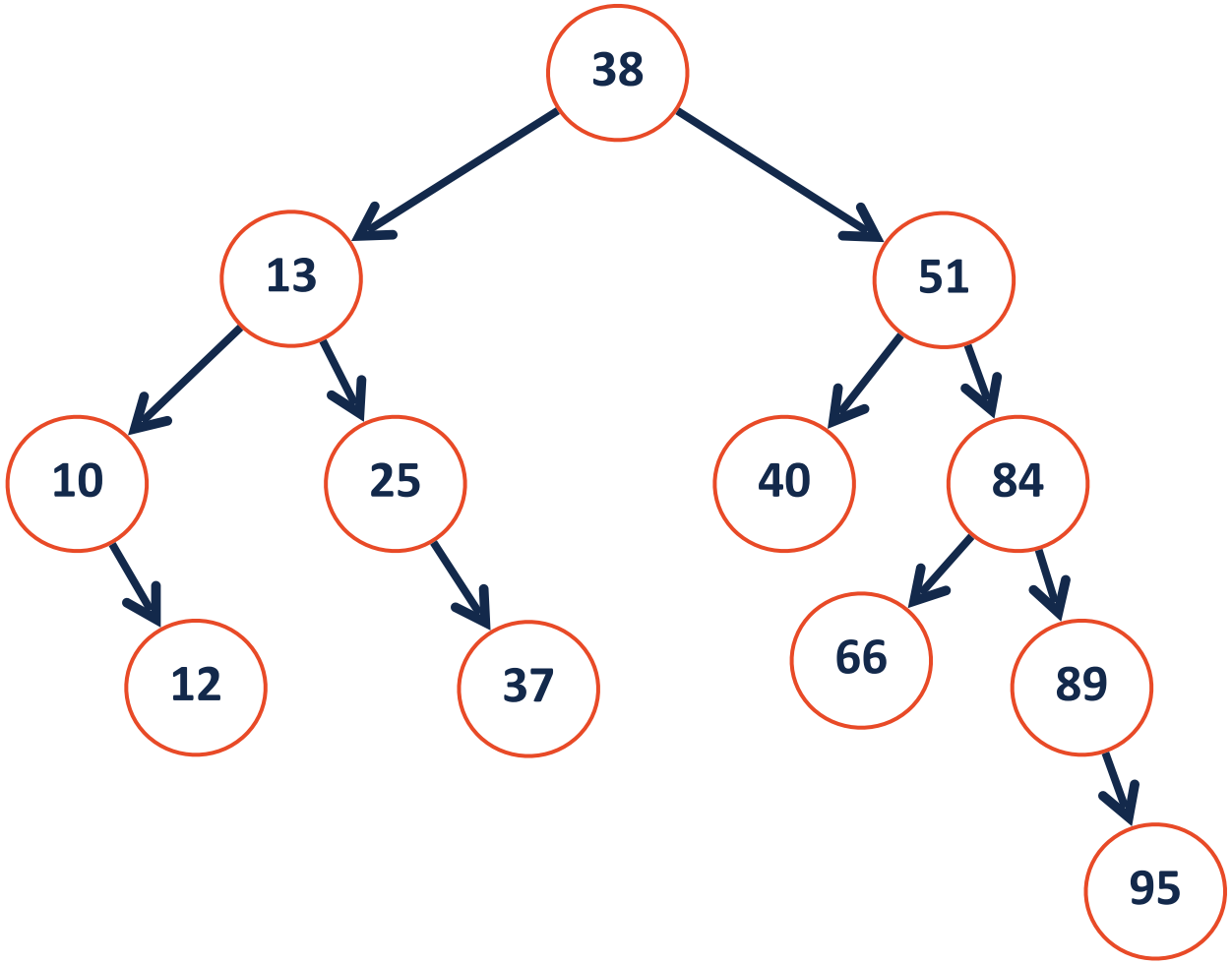
Remove

Find

Traverse

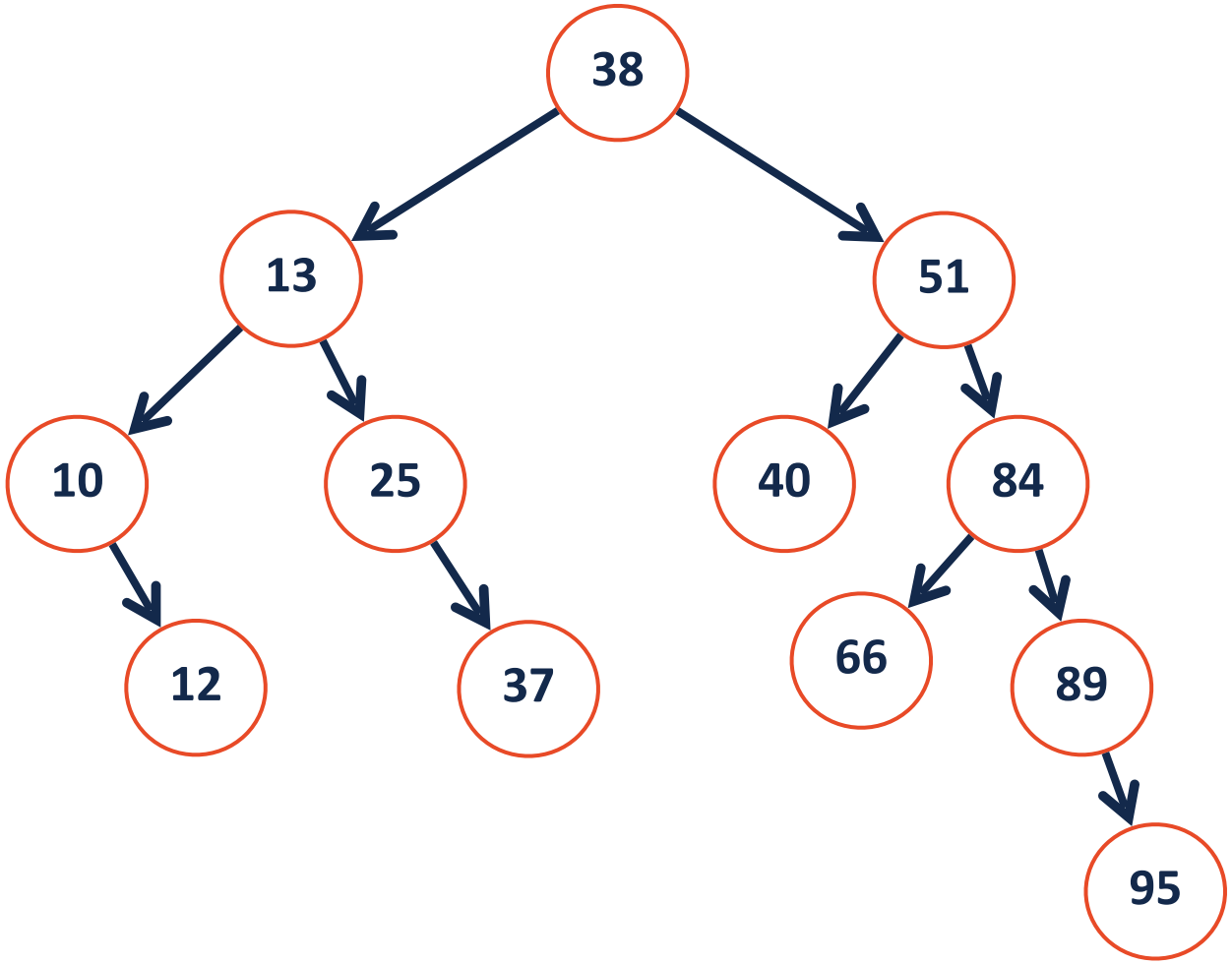
BST Find

`find(66)`

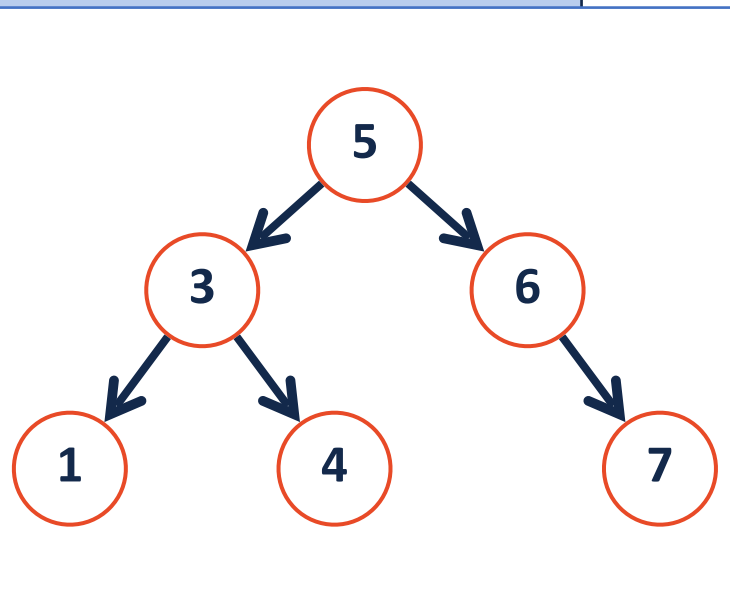


BST Find

`find(9)`

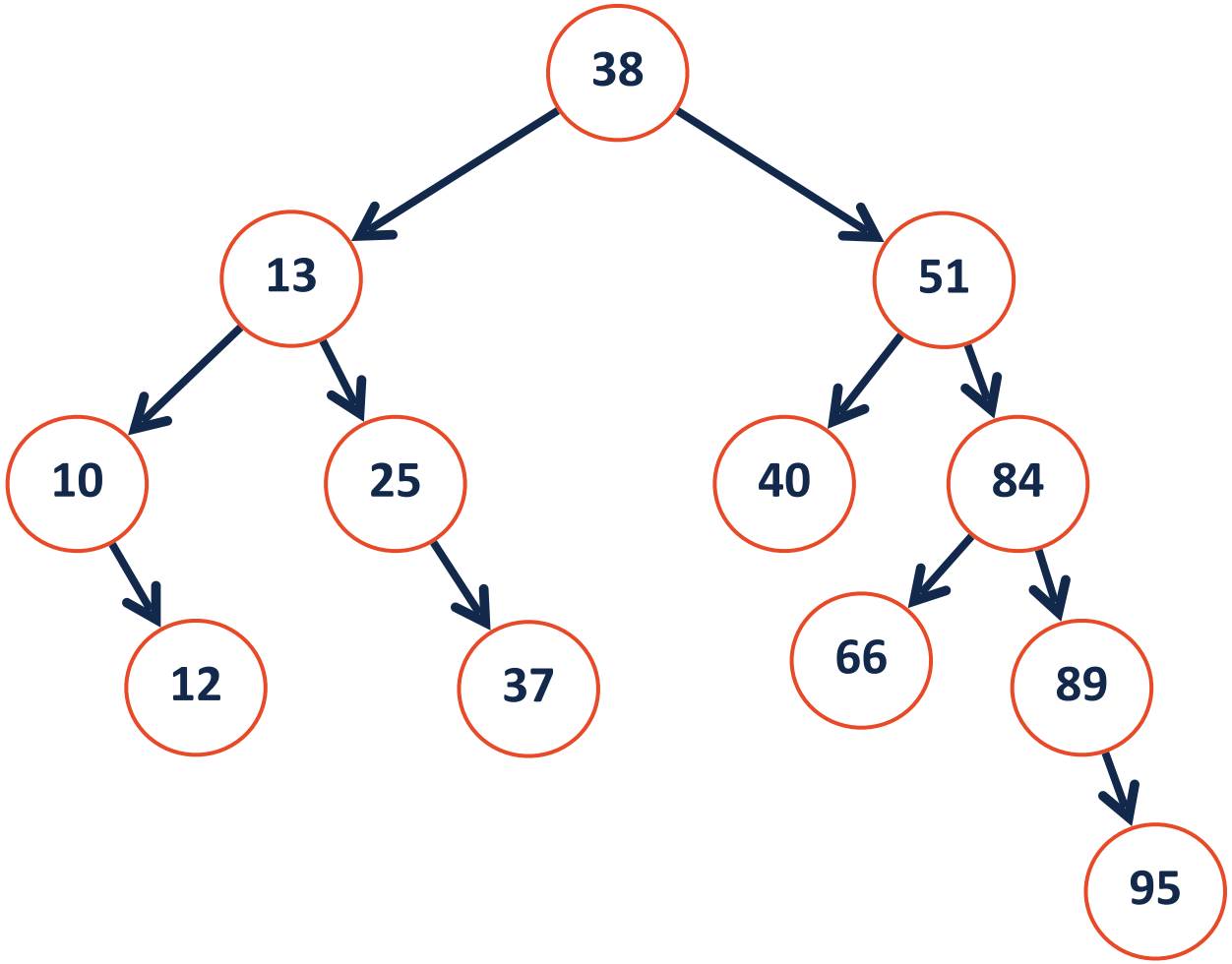


```
1  template<typename K, typename V>
2
3  ____________________________ _find(TreeNode *& root, const K & key) {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```



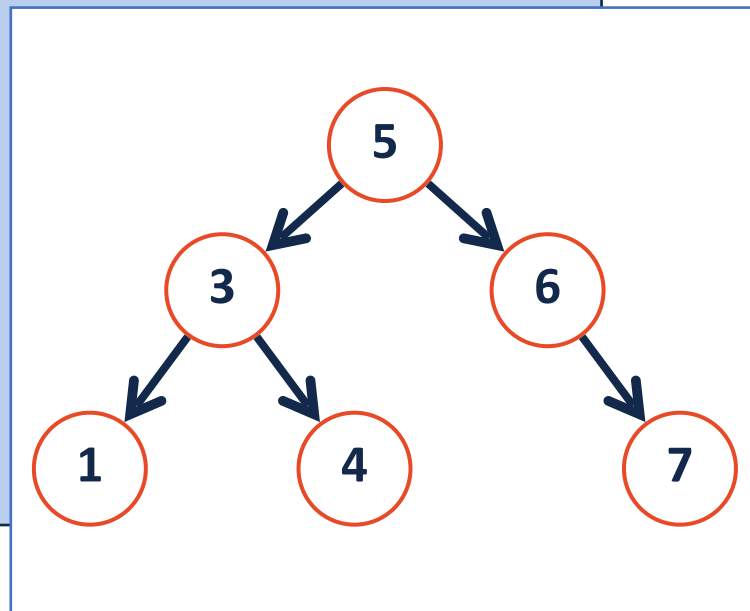
BST Insert

`insert(33, v)`



```
1  template<typename K, typename V>
2
3  void _insert(const K & key, const V & val) {
4
5      return _insert(root, key, val);
6  }
7
```

```
1  template<typename K, typename V>
2
3  void _insert(TreeNode *& root, const K & key, const V & val) {
4
5
6
7
8
9
10
11
12
13
14
15
16 }
```



BST Insert

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

BST Remove

What should our tree look like after the following…

```
remove(40)

remove(13)

remove(51)
```