

Data Structures

Binary Search Trees

CS 225

September 18, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



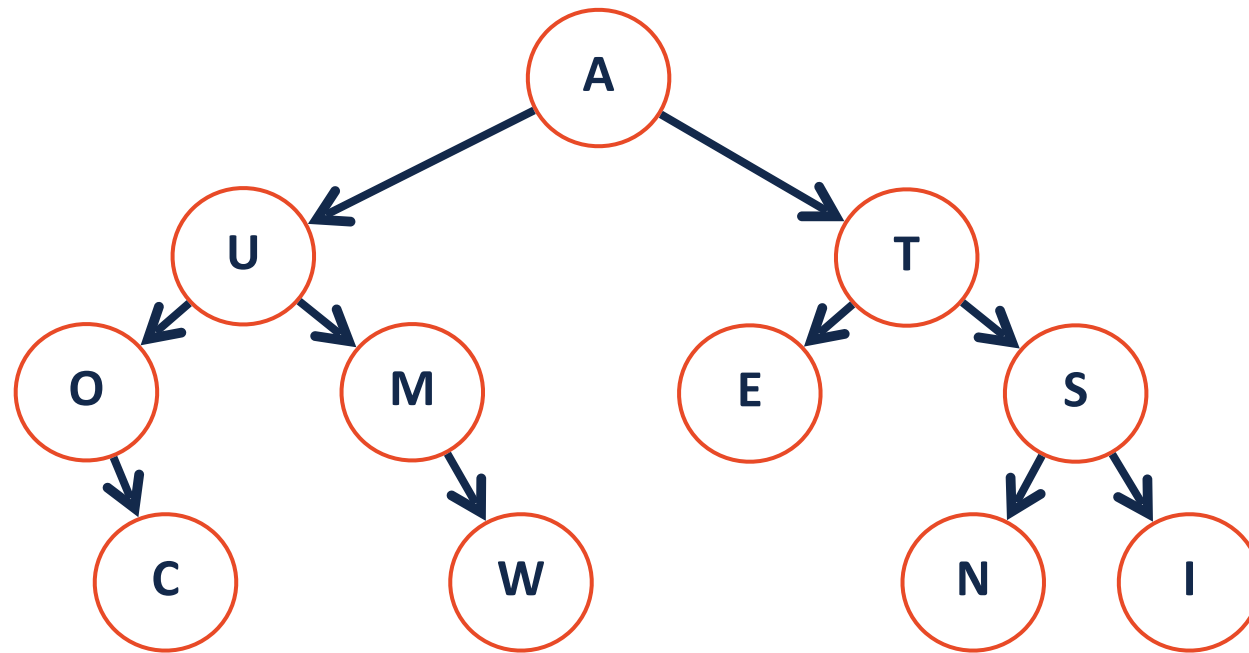
Learning Objectives

Review binary trees

Extend binary trees into binary *search* trees

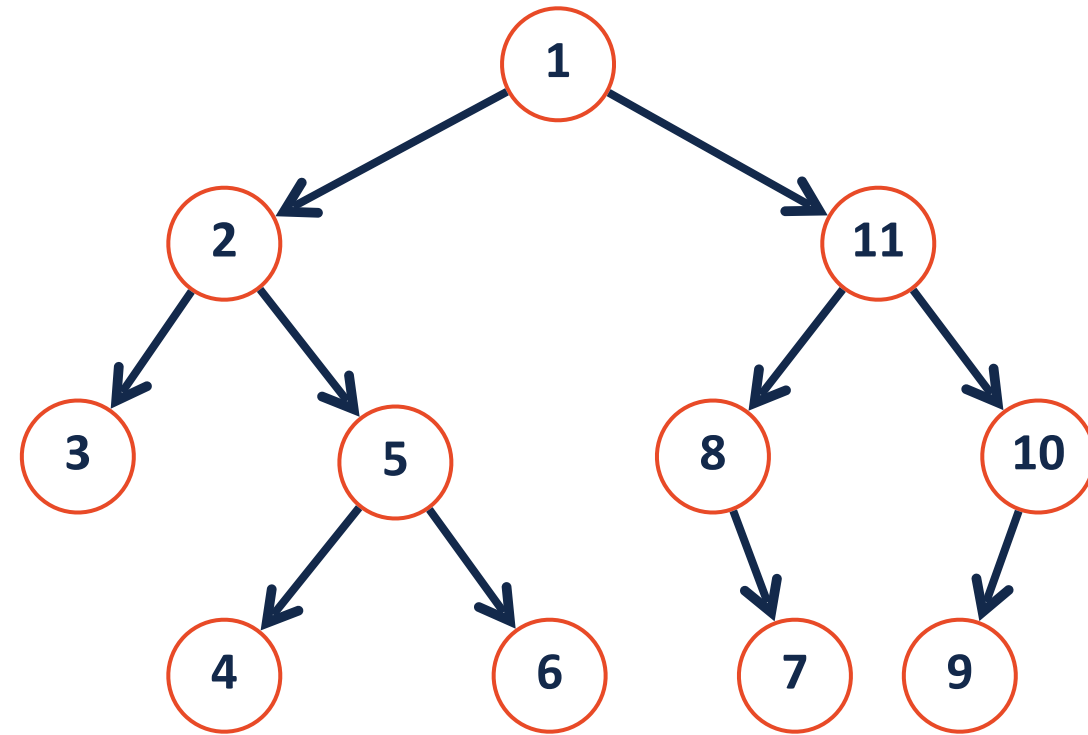
Tree Search

There are two main approaches to searching a binary tree:



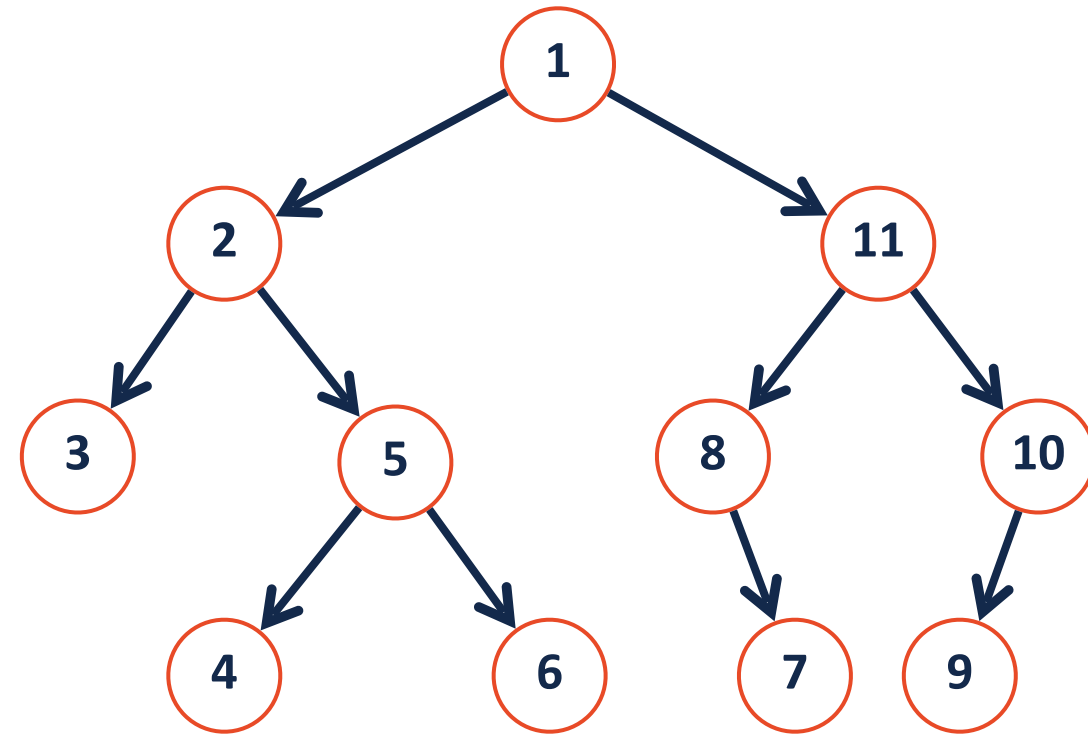
Depth First Search

Explore as far along one path as possible before backtracking

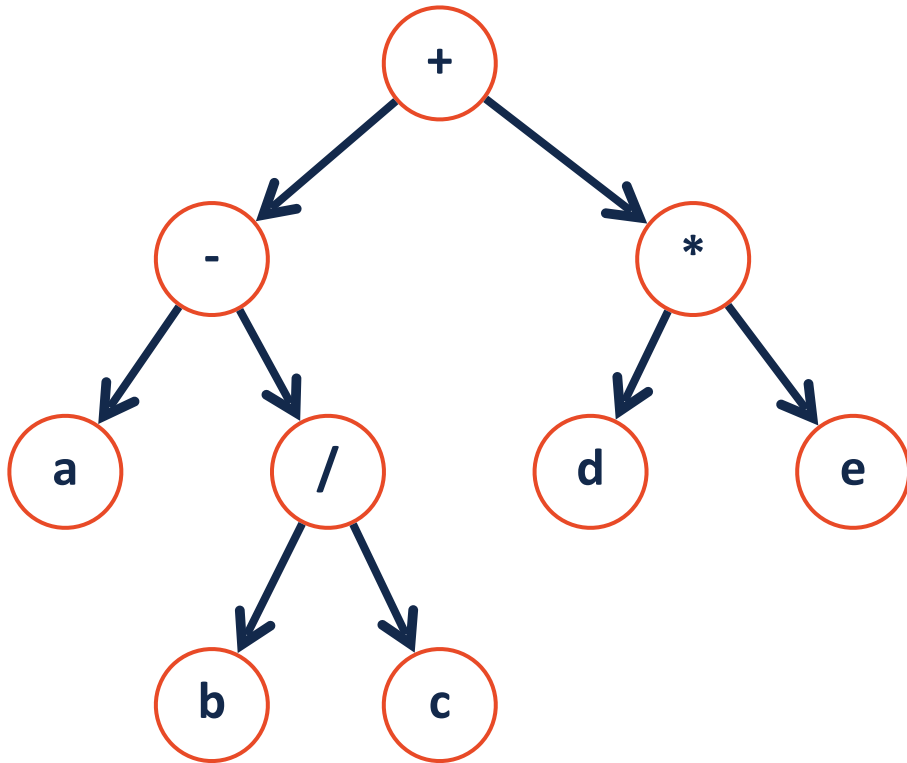


Breadth First Search

Fully explore depth i before exploring depth $i+1$



Level-Order Traversal



```
1 template<class T>
2 void BinaryTree<T>::lOrder(TreeNode * root)
3 {
4
5     Queue<TreeNode*> q;
6     q.enqueue(root);
7
8     while( q.empty() == False){
9
10        TreeNode* temp = q.head();
11        process(temp);
12
13        q.dequeue();
14
15        q.enqueue(temp->left);
16        q.enqueue(temp->right);
17
18    }
19 }
```

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?

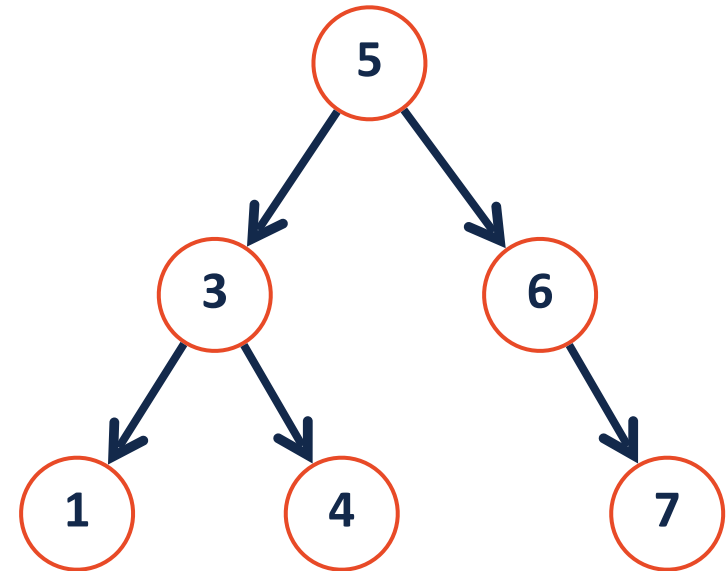
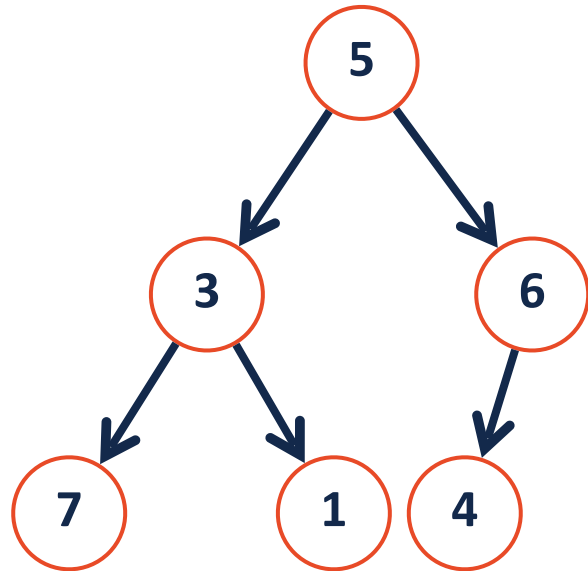
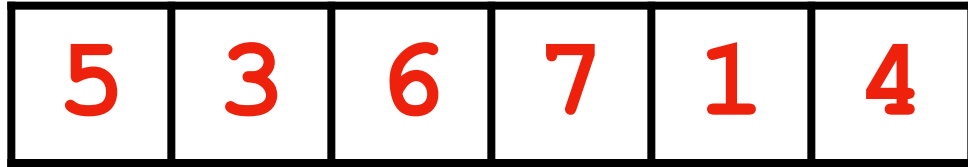


Tree Search

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



Dictionary ADT

Data is often organized into key/value pairs:

Word → Definition

Course Number → Lecture/Lab Schedule

Node → Incident Edges

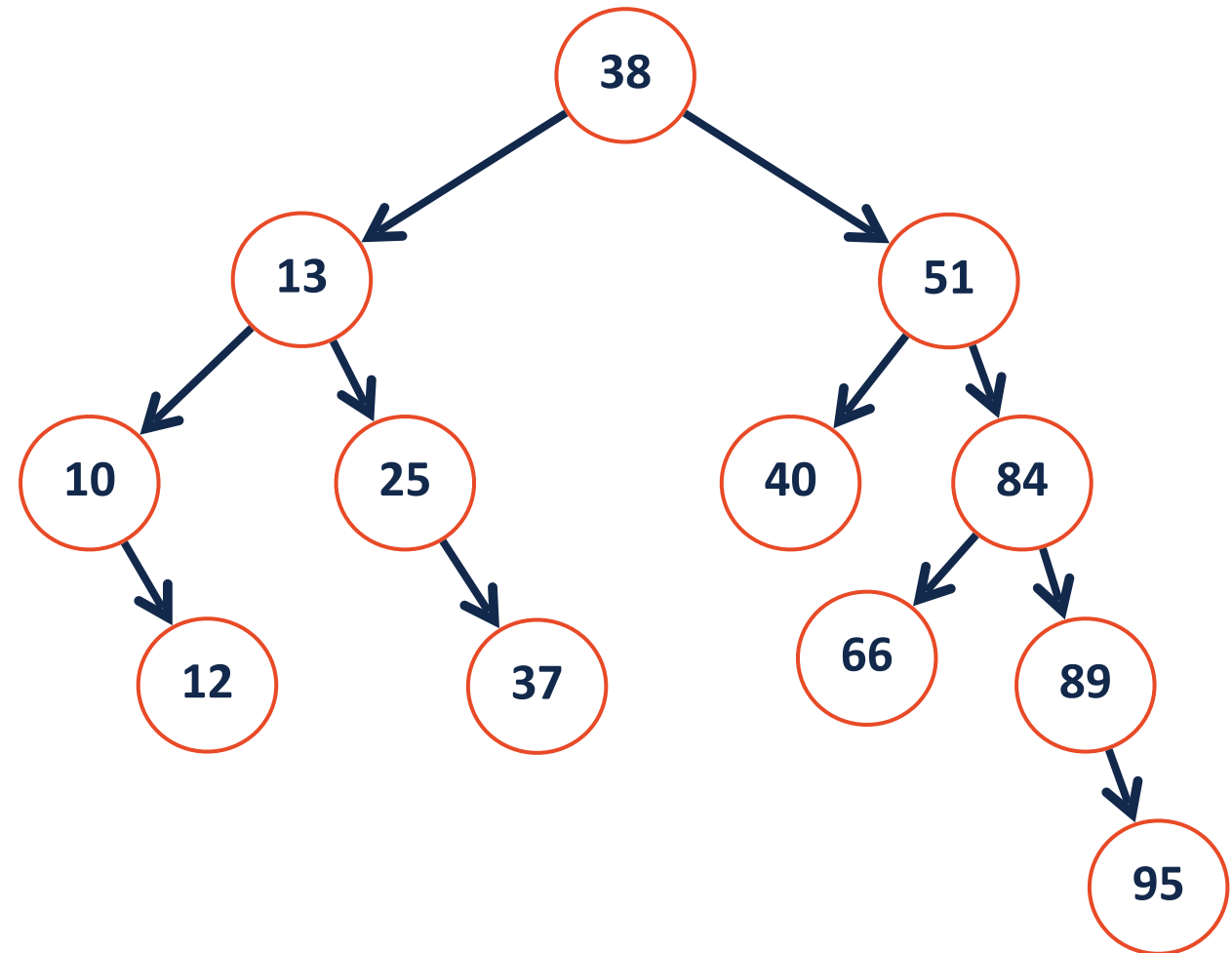
Flight Number → Arrival Information

URL → HTML Page

...

Binary Search Tree (BST)

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



BST.h

```
1 #pragma once
2
3 template <typename K, typename V>
4 class BST {
5     public:
6         /* ... */
7     private:
8         class TreeNode {
9             K & key;
10            V & value;
11
12            TreeNode *left, *right;
13
14            TreeNode(K & k, V & v) :
15            key(k), value(v),
16            left(NULL), right(NULL) { }
17            };
18
19            TreeNode *root_;
20            /* ... */
21 };
22
23
```

Tree.h

```
1 #pragma once
2
3 template <typename T>
4 class BinaryTree {
5     public:
6         /* ... */
7     private:
8         class TreeNode {
9             T & data;
10
11            TreeNode * left;
12
13            TreeNode * right;
14
15            TreeNode(T & data) :
16            data(data), left(NULL),
17            right(NULL) { }
18
19            };
20
21            TreeNode *root_;
22            /* ... */
23 };

```

Binary Search Tree ADT



Insert

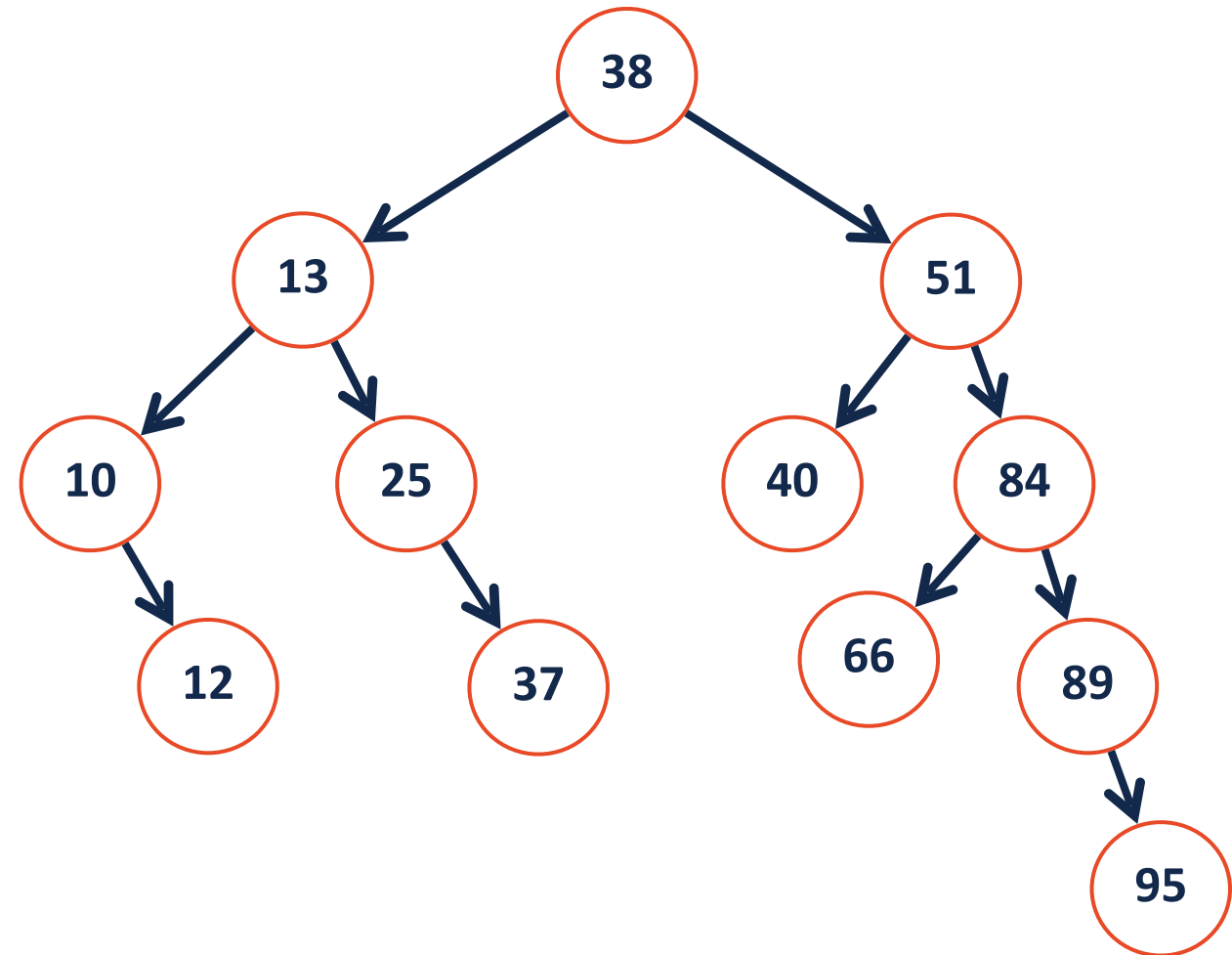
Remove

Find

Traverse

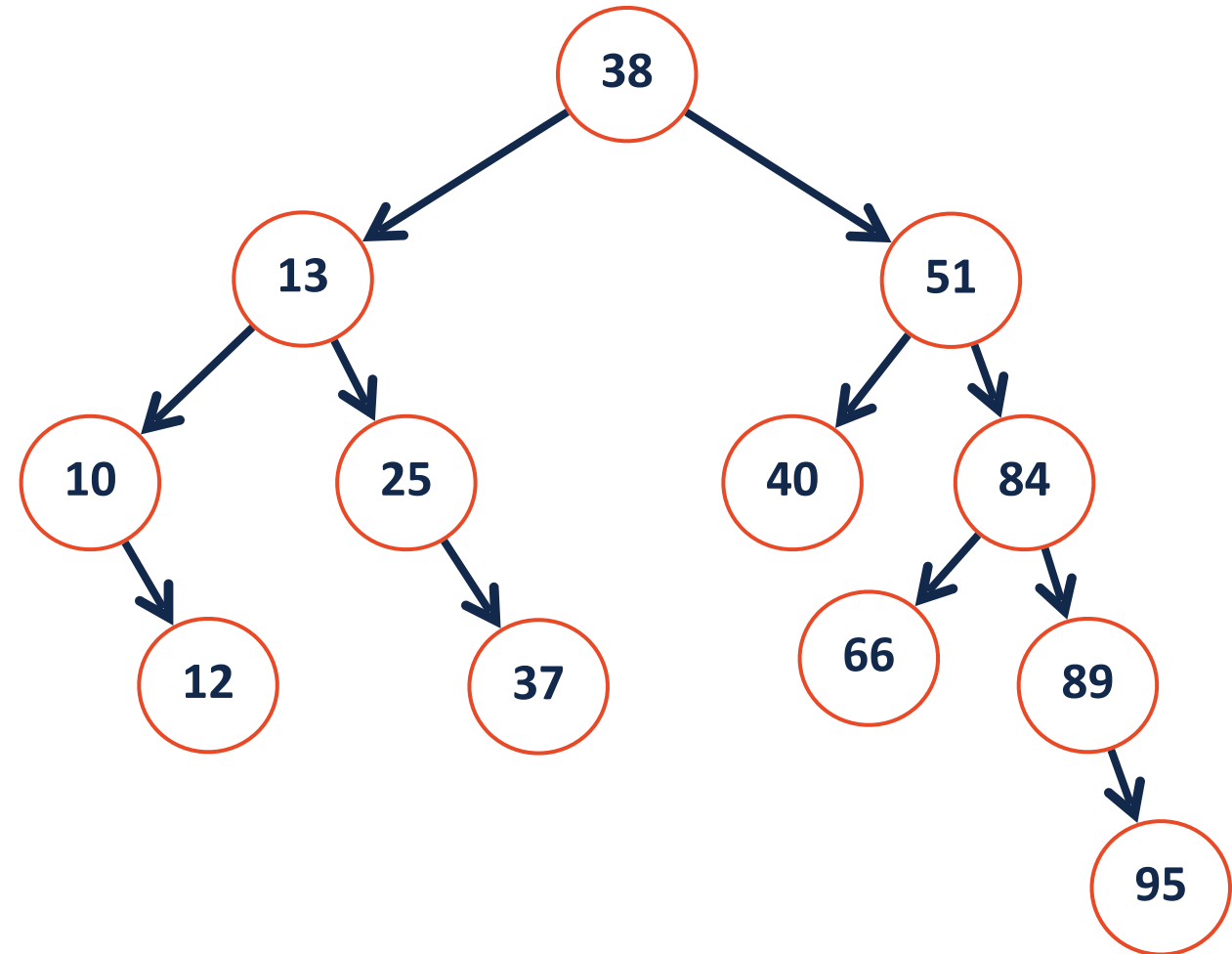
BST Find

find(66)



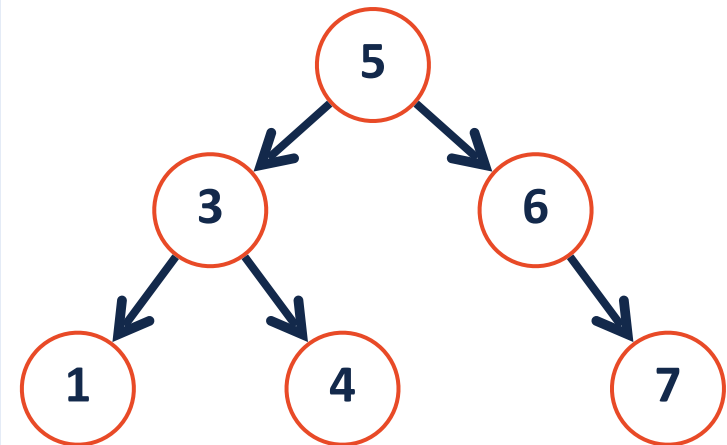
BST Find

find(9)



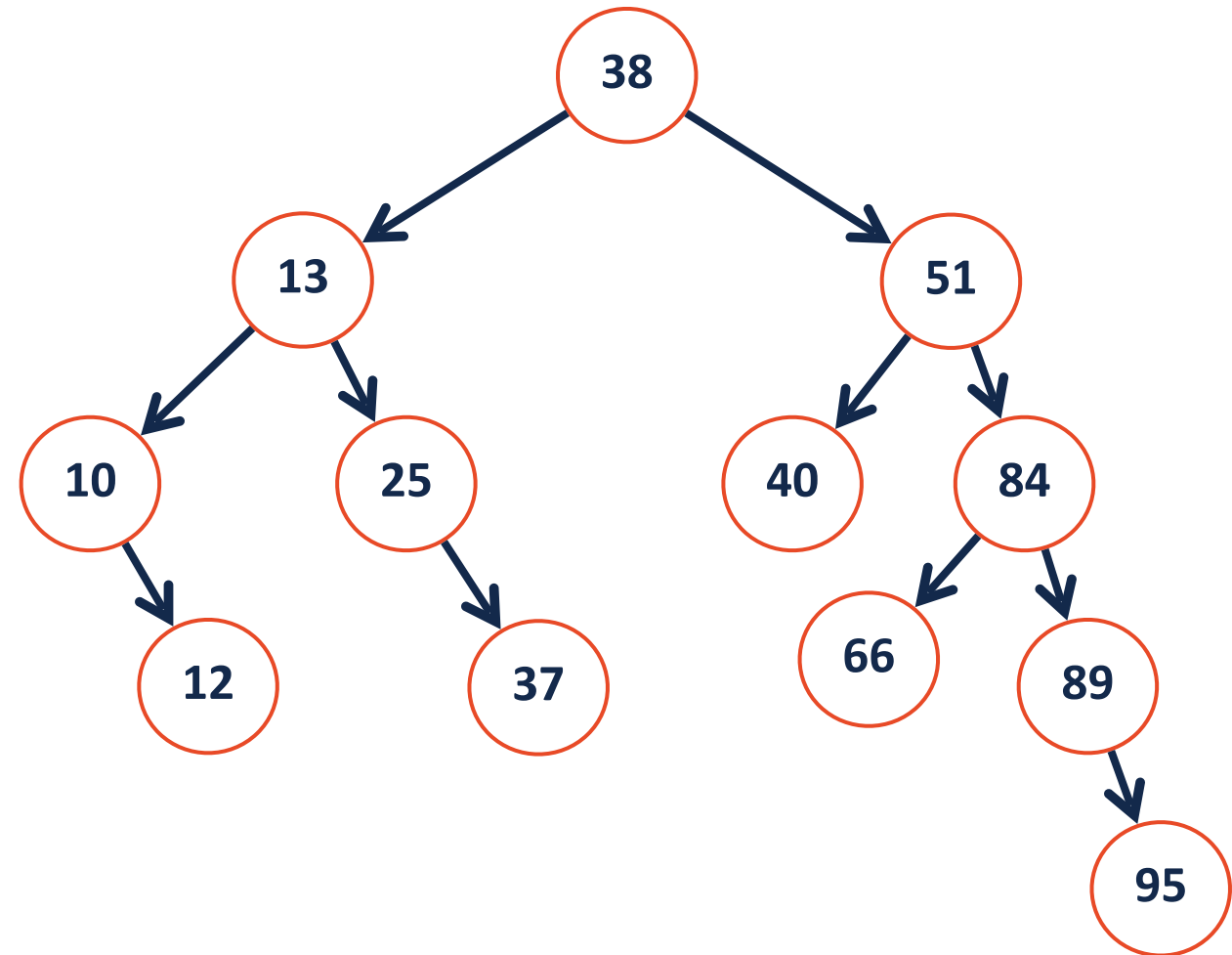


```
1 template<typename K, typename V>
2
3     __find(TreeNode *& root, const K & key) {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```



BST Insert

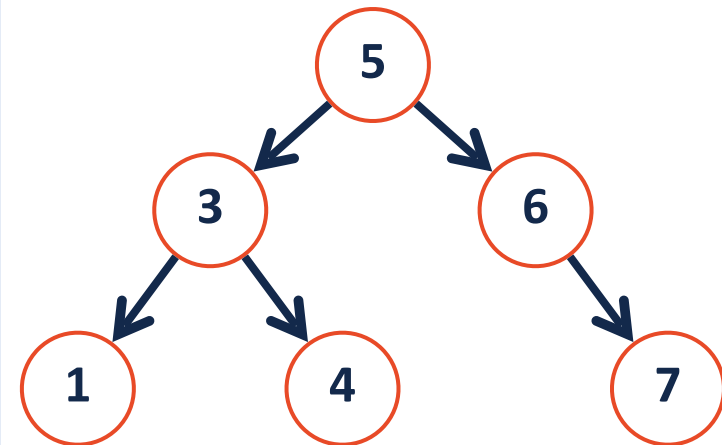
insert(33)





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

```
1 template<typename K, typename V>
2
3 void _insert(TreeNode *& root, const K & key) {
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, \emptyset]

BST Remove

What should our tree look like after the following...

remove (40)

remove (13)

remove (51)

