PROGRAMMING AND DATA STRUCTURES

BINARY TREES (BST)

HOURIA OUDGHIRI

FALL 2023

OUTLINE

- Binary Search Trees (BST)
- Properties of the BST
- Operations on the BST
- Implementation of the BST class

STUDENT LEARNING OUTCOMES

At the end of this chapter, you should be able to:

- Describe the properties of binary search trees (BST)
- Trace operations on the BST
- Implement the BST generic data structure
- Use the BST data structure
- Evaluate the complexity of the operations on the BST

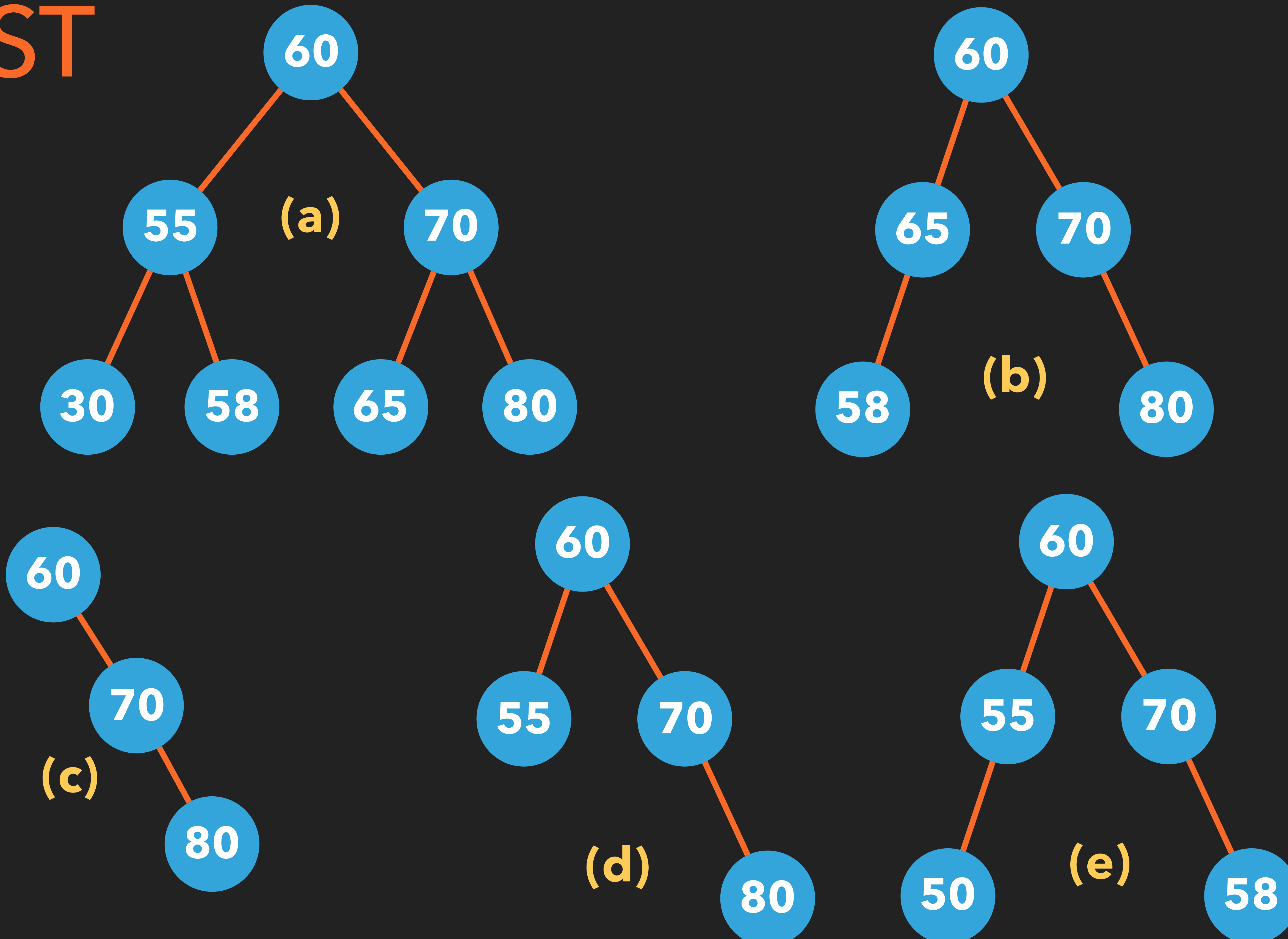
Binary Search Tree (BST)

- Special binary tree
- Used for efficient binary search in large data sets
- BST is a set (no duplicates are allowed)

BST

- Properties of the BST
  - BST has a root, a left subtree (L) and a right subtree (R)
  - The value of the root is greater than the value of every node in L
  - The value of the root is less than the value of every node in R
  - L and R are also BSTs

BST

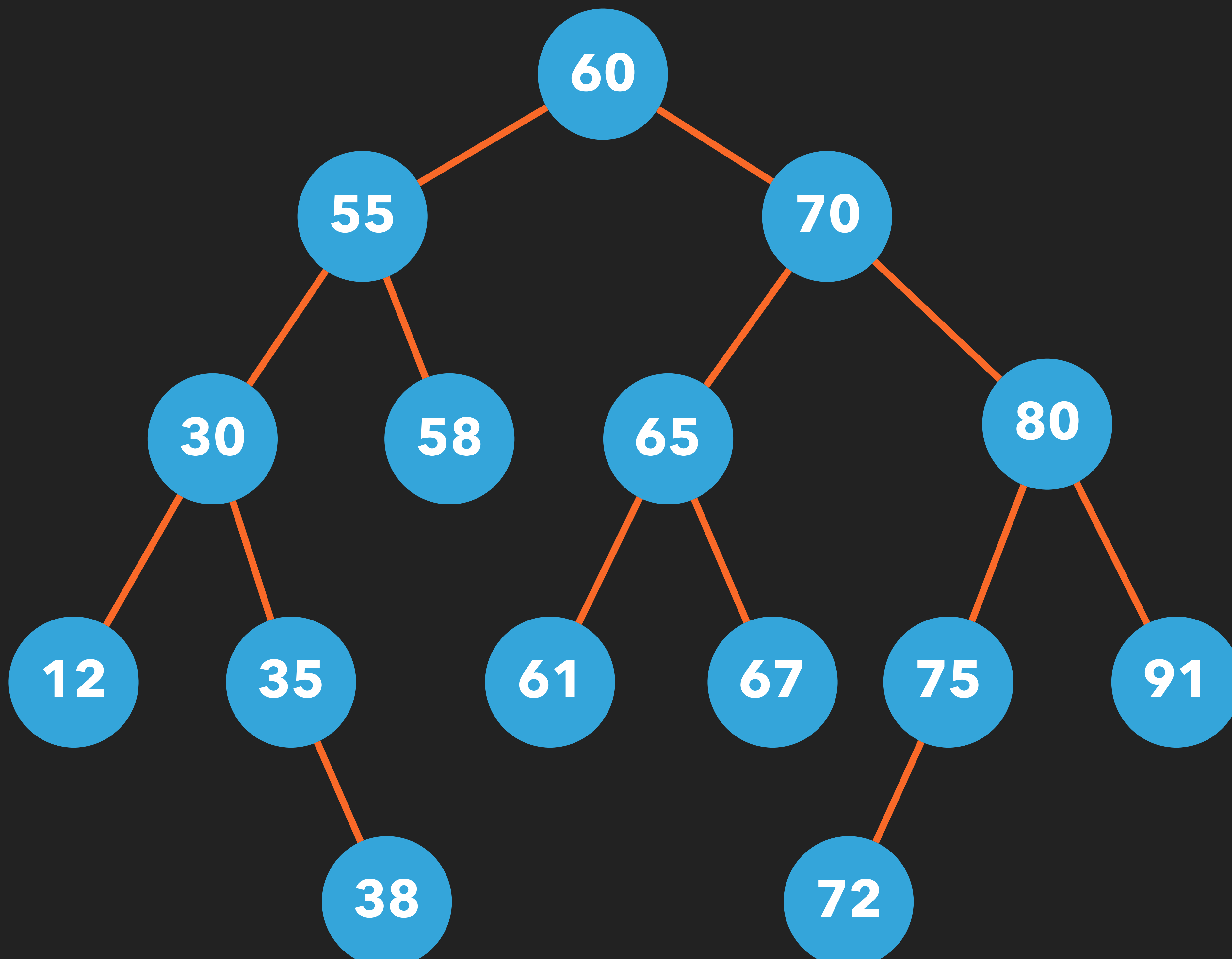

60
55
70
30
58
65
80
(a)
60
65
70
58
80
(b)
60
70
80
(c)
60
55
70
80
(d)
60
55
70
50
58
(e)

BST

- Common operations on the BST
  - Search for a specific value in the BST
  - Add a node to the BST while keeping the BST properties
  - Remove a node from the BST while keeping the BST properties
  - Traverse the BST (preorder, inorder, postorder)

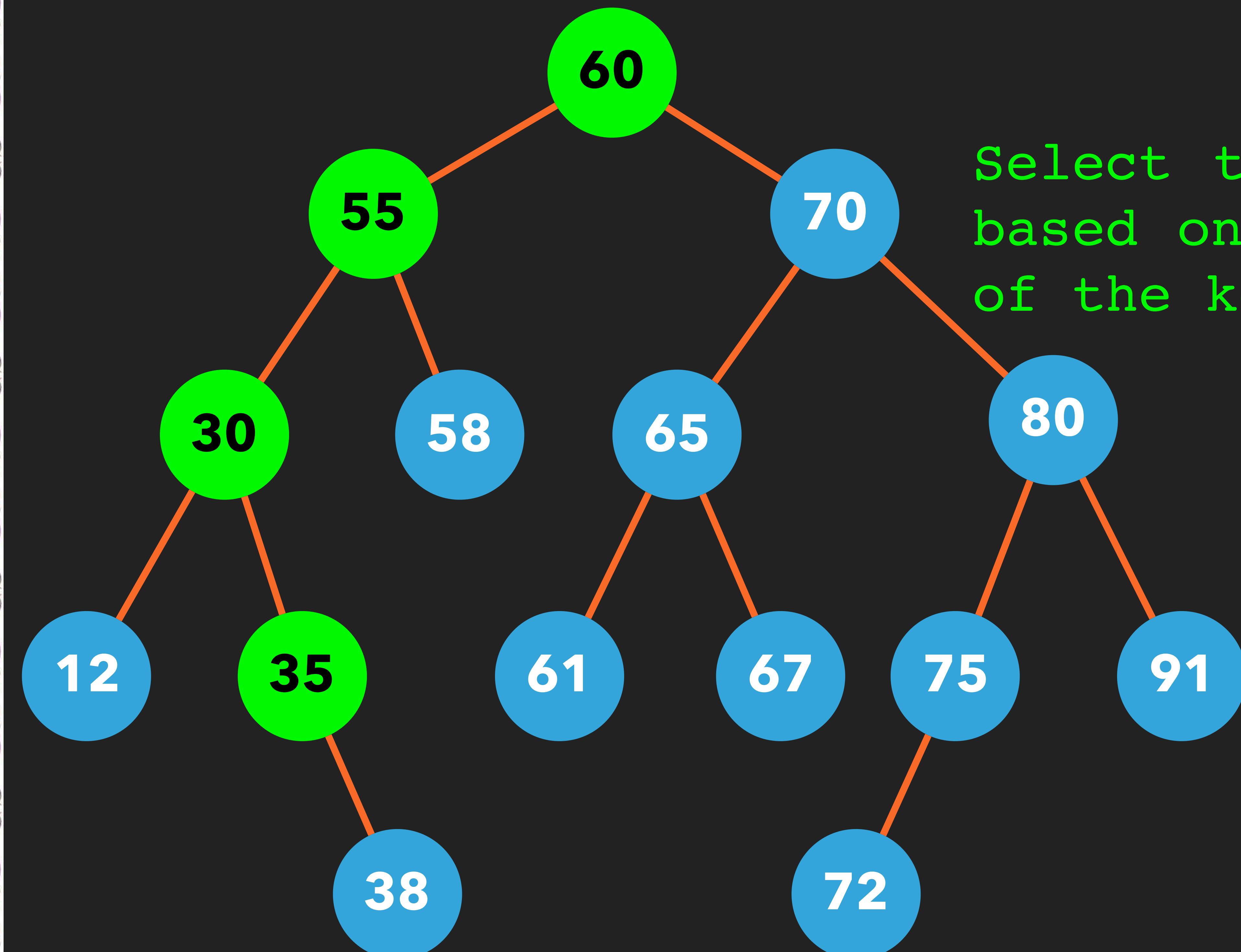
BST (Search)

Search for the value 35



BST (Search)

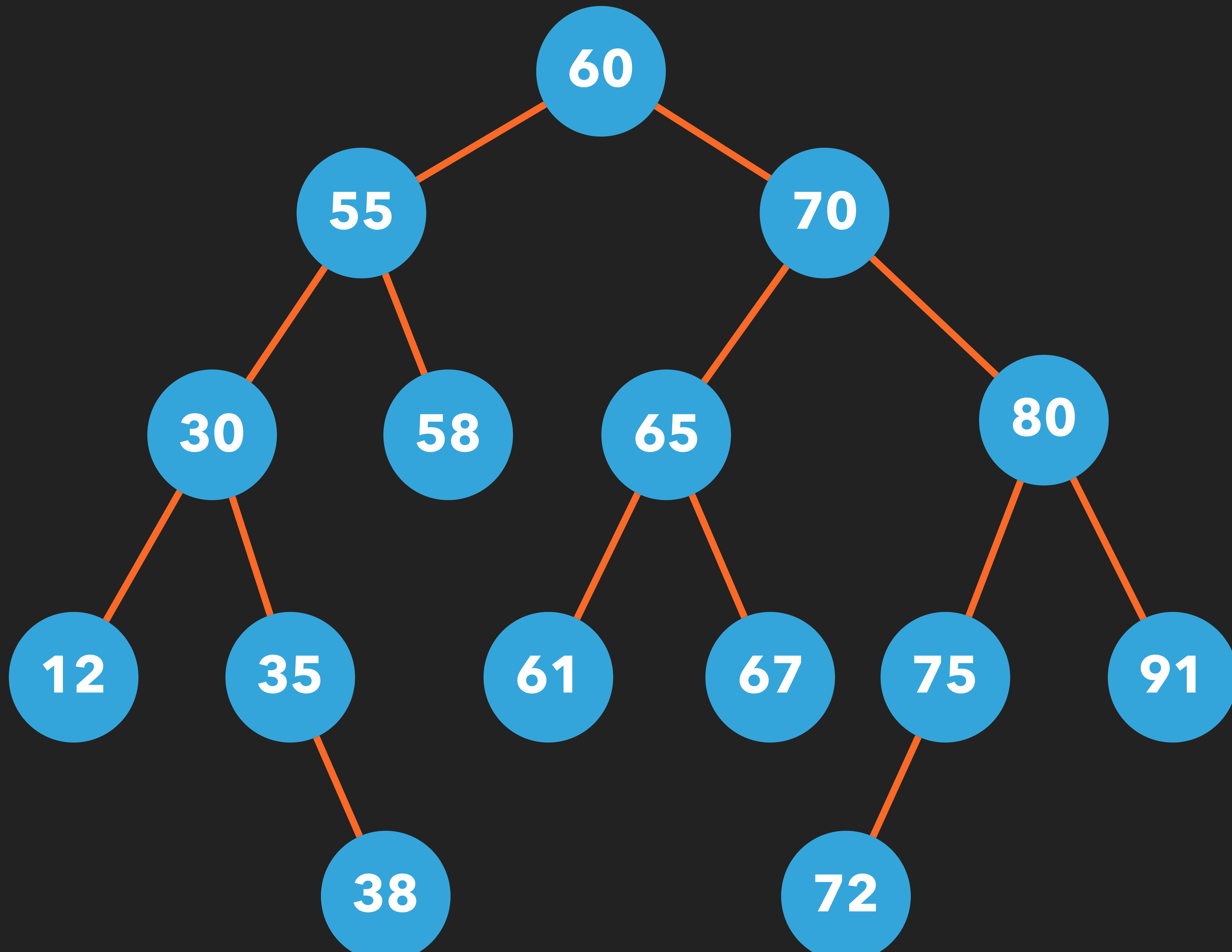
Search for the value 35



Select the subtree based on the value of the key

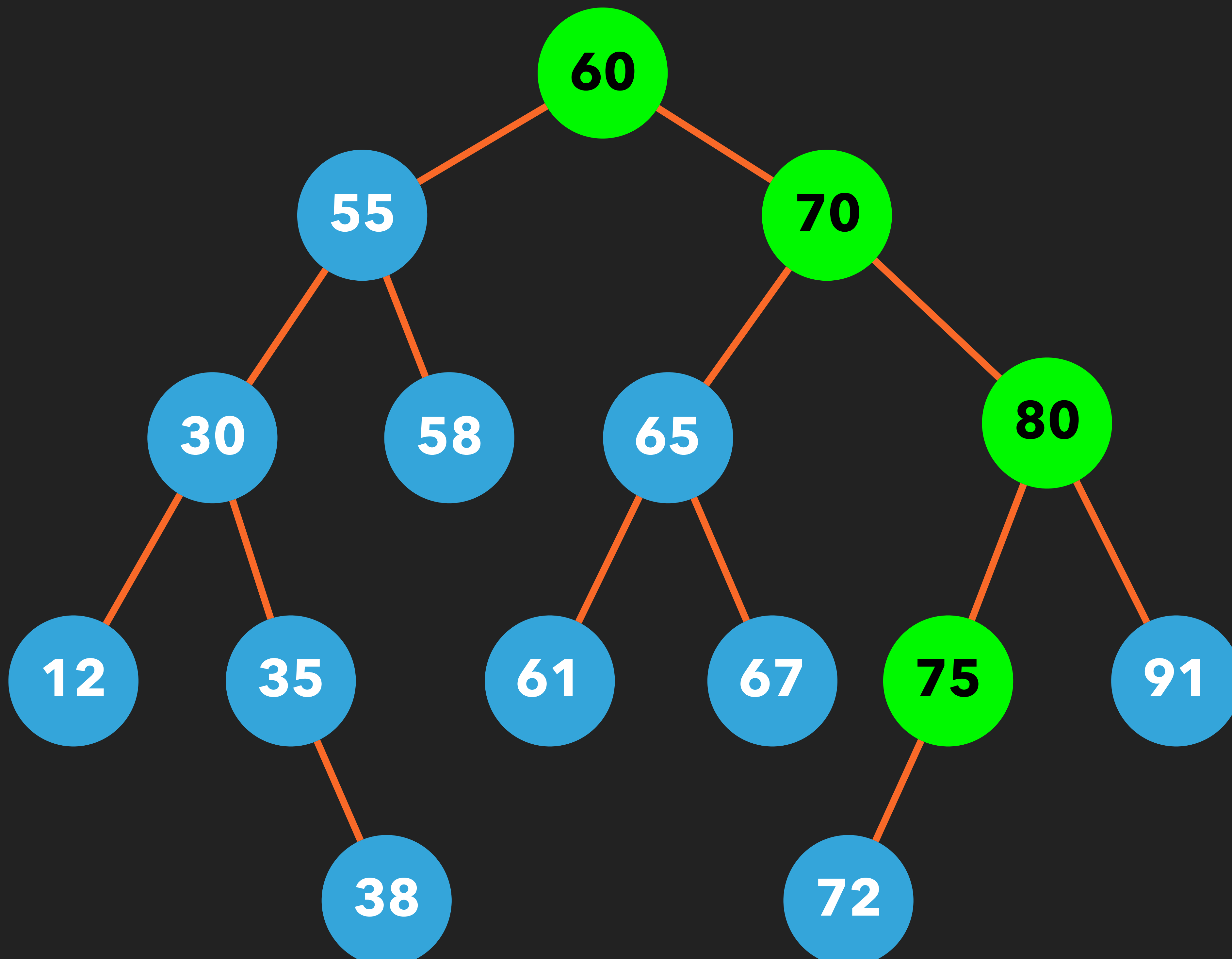
BST (Search)

Search for the value 75



BST (Search)

Search for the value 75

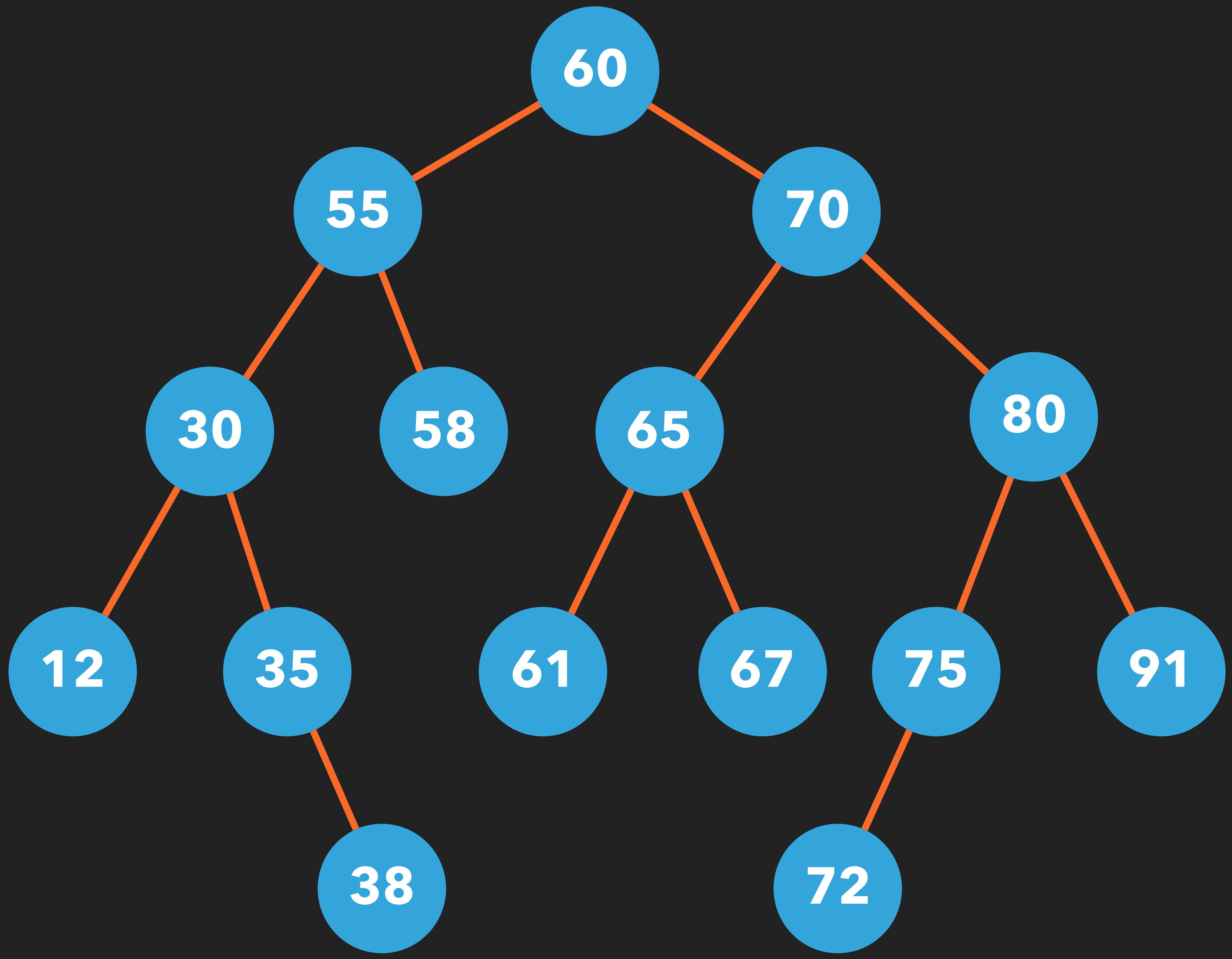


BST (Search)

```
boolean contains (value)

  current node = root // start from the root
  while(current node is not null){
     if(the value of the current node == value)
        return true
     else if (value of the current node > value)
        current node is set to its left child
     else
        current node is set to its right child
  }
  return false

end contains
```

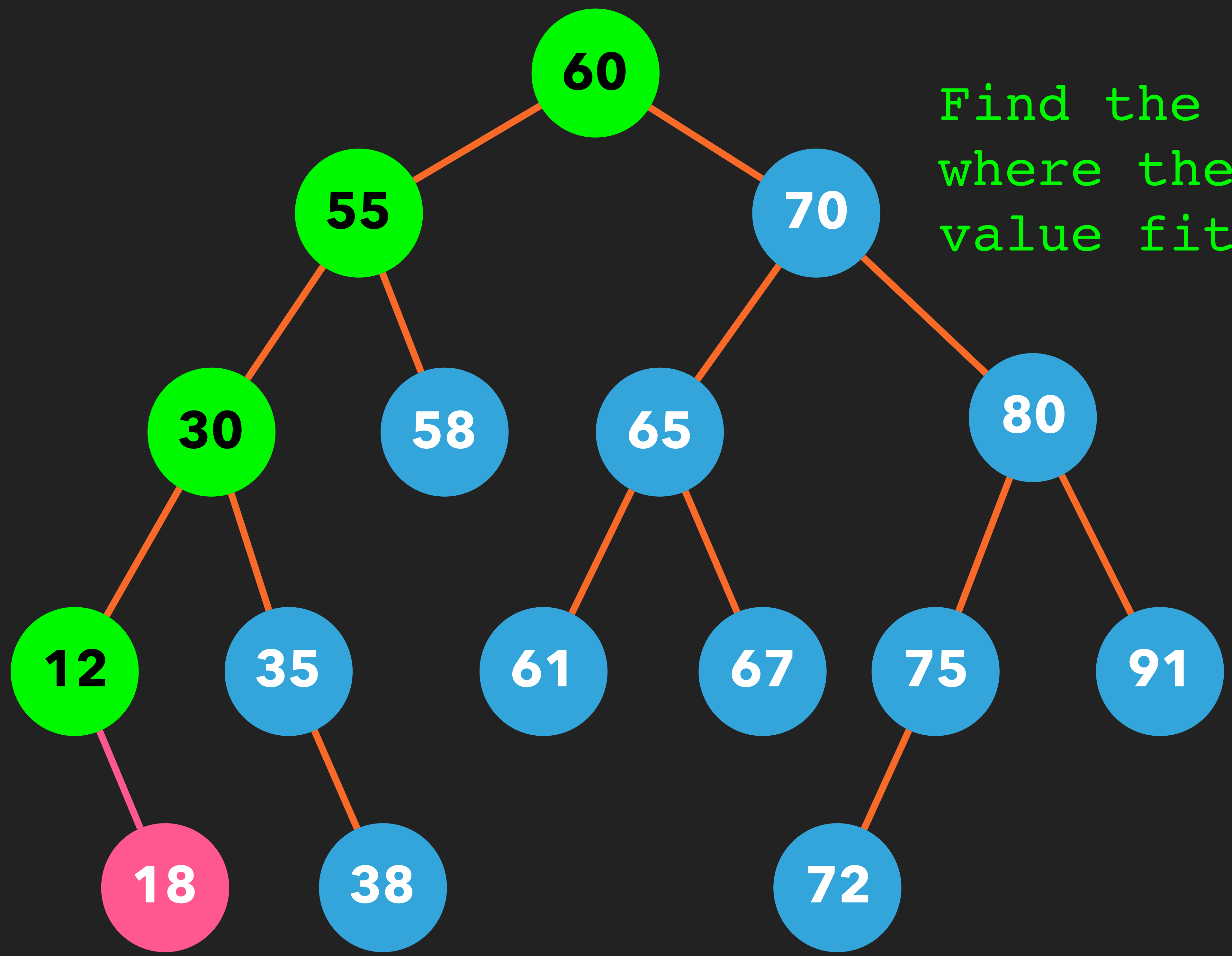
BST (Add)

Add the value 18



BST (Add)

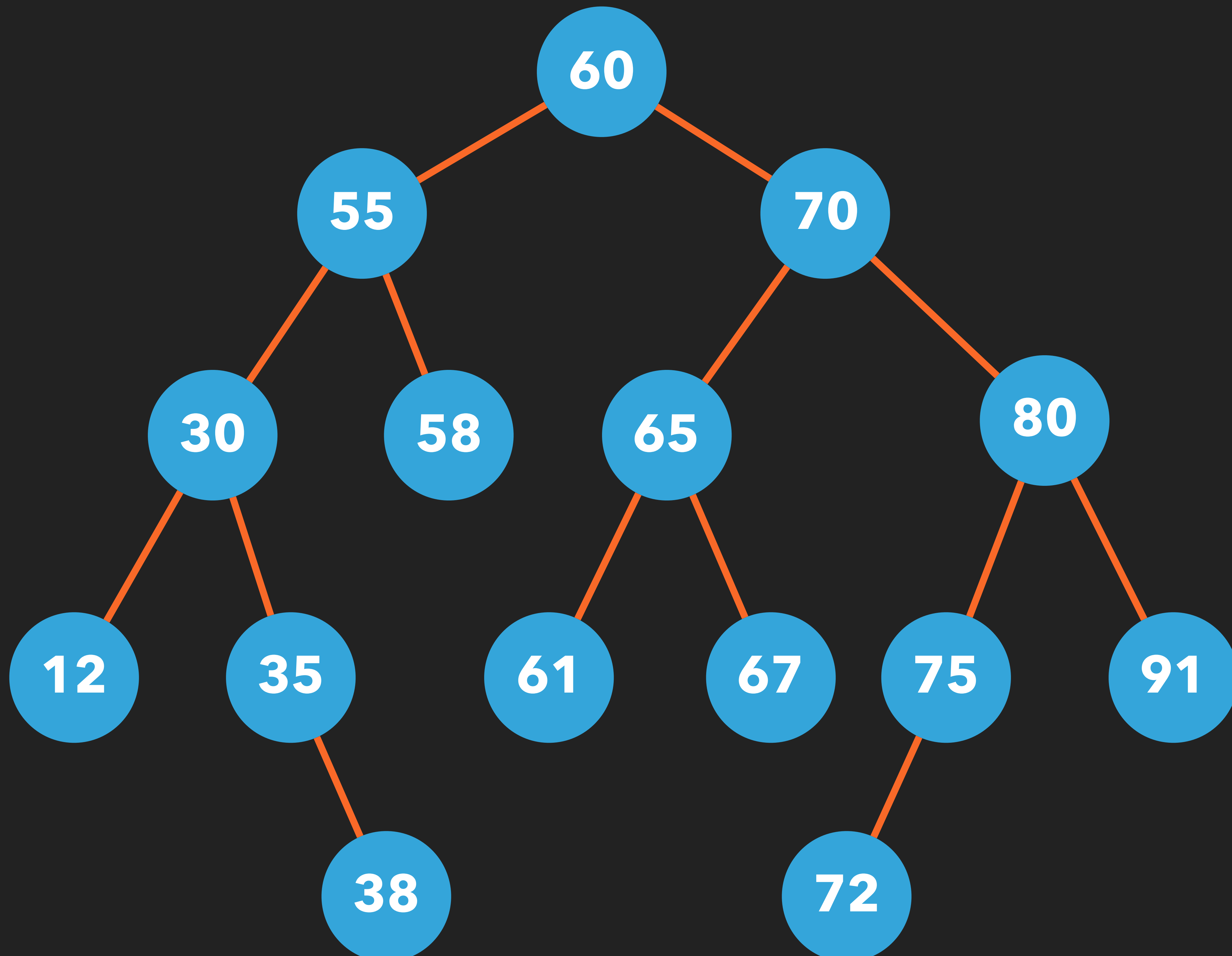
Add the value 18



Find the subtree where the new value fits

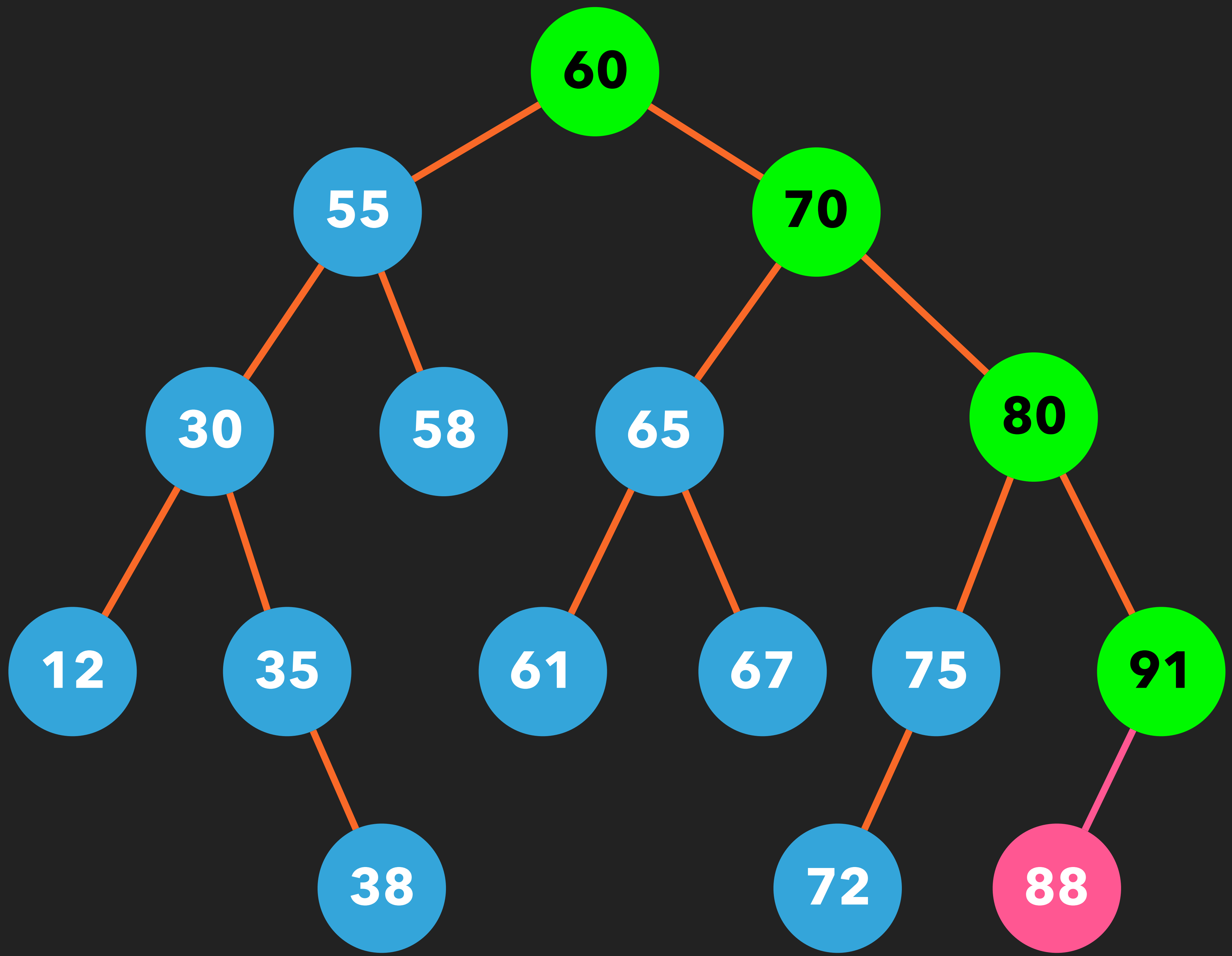
BST (Add)

Add the value 88



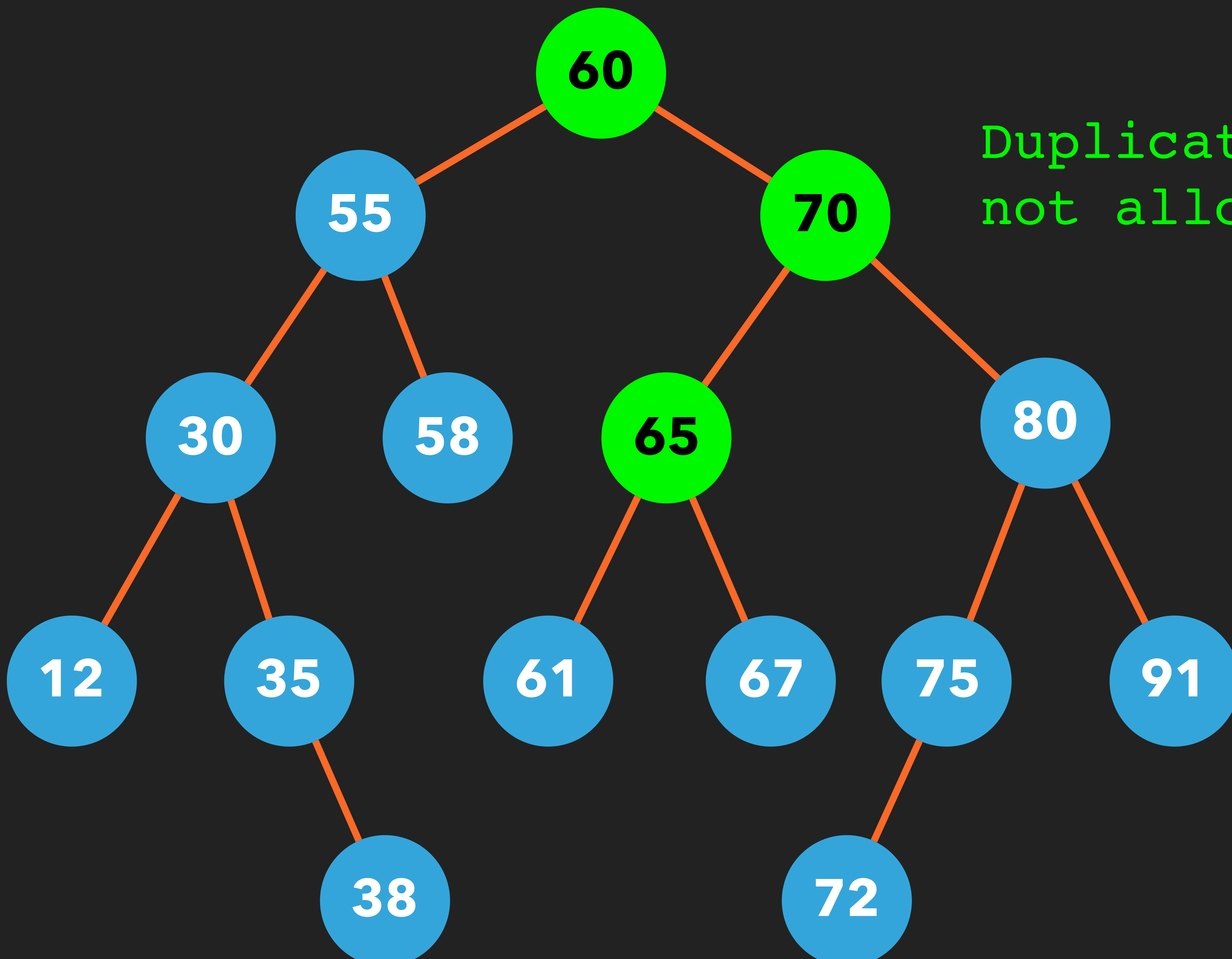
BST (Add)

Add the value 88



BST (Add)

Add the value 65



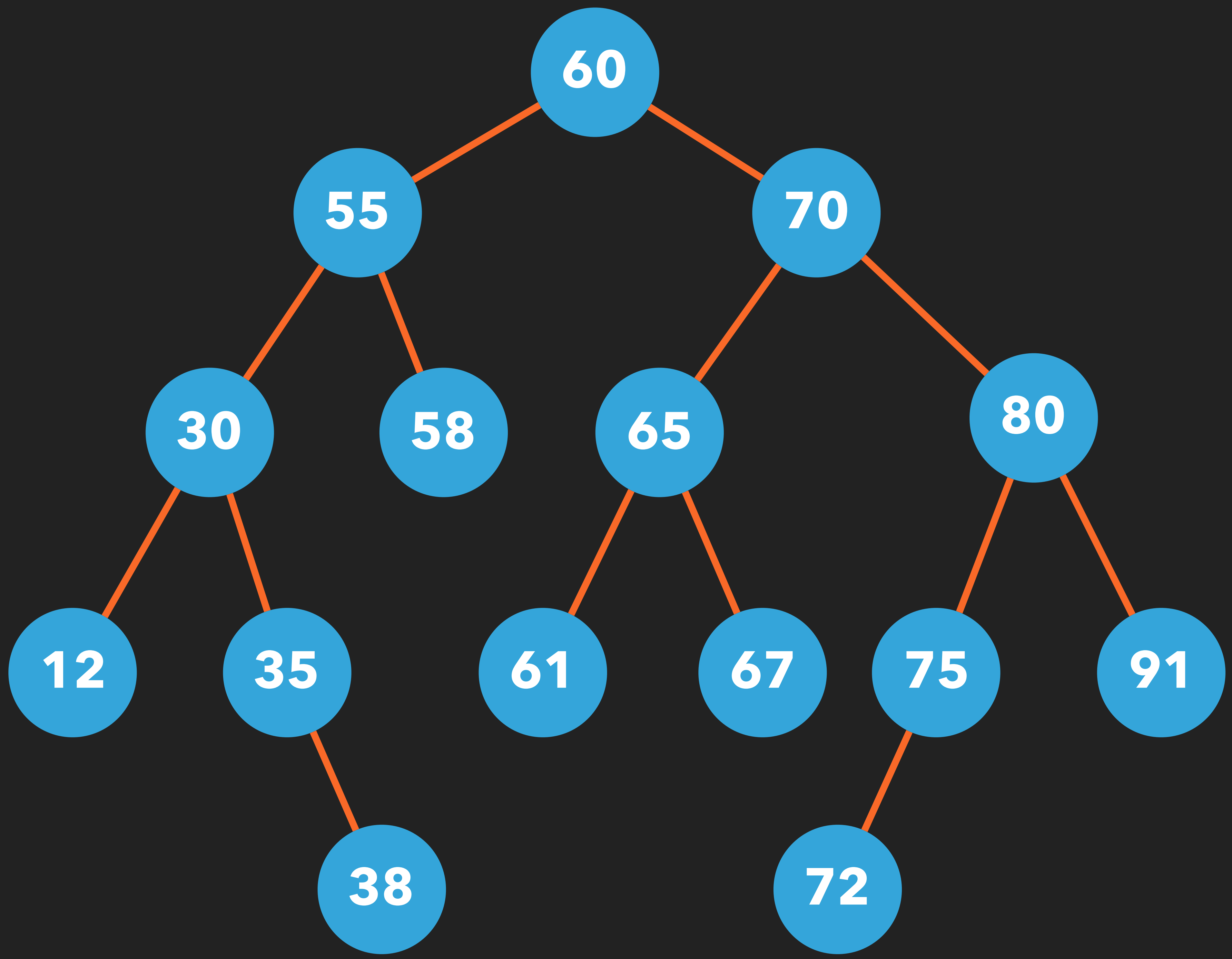
Duplicates are not allowed

BST (Add)

```
boolean add (value)
   currentNode = root
   while(currentNode is not null){
     parentNode = currentNode
     if(the value of currentNode == value)
        return false (duplicates are not allowed)
     else if (value of currentNode > value)
        currentNode is set to its left child
     else
        currentNode is set to its right child
   }
   if (the value of parentNode > value)
      Add a left child with value to parentNode
   else
      Add a right child with value to parentNode
   end if
   return true
end add
```

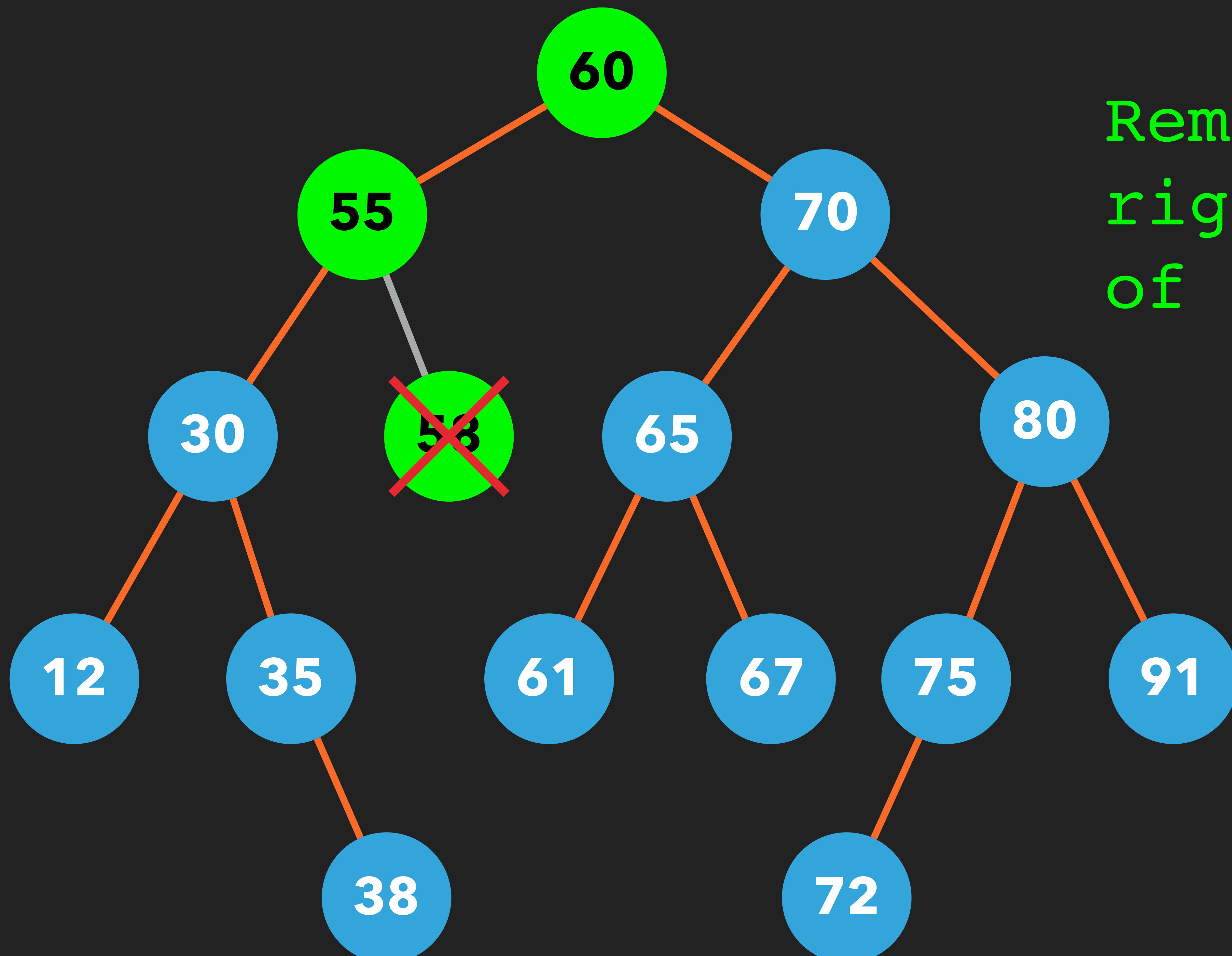
BST (Remove)

Remove the value 58 (Leaf)



BST (Remove)

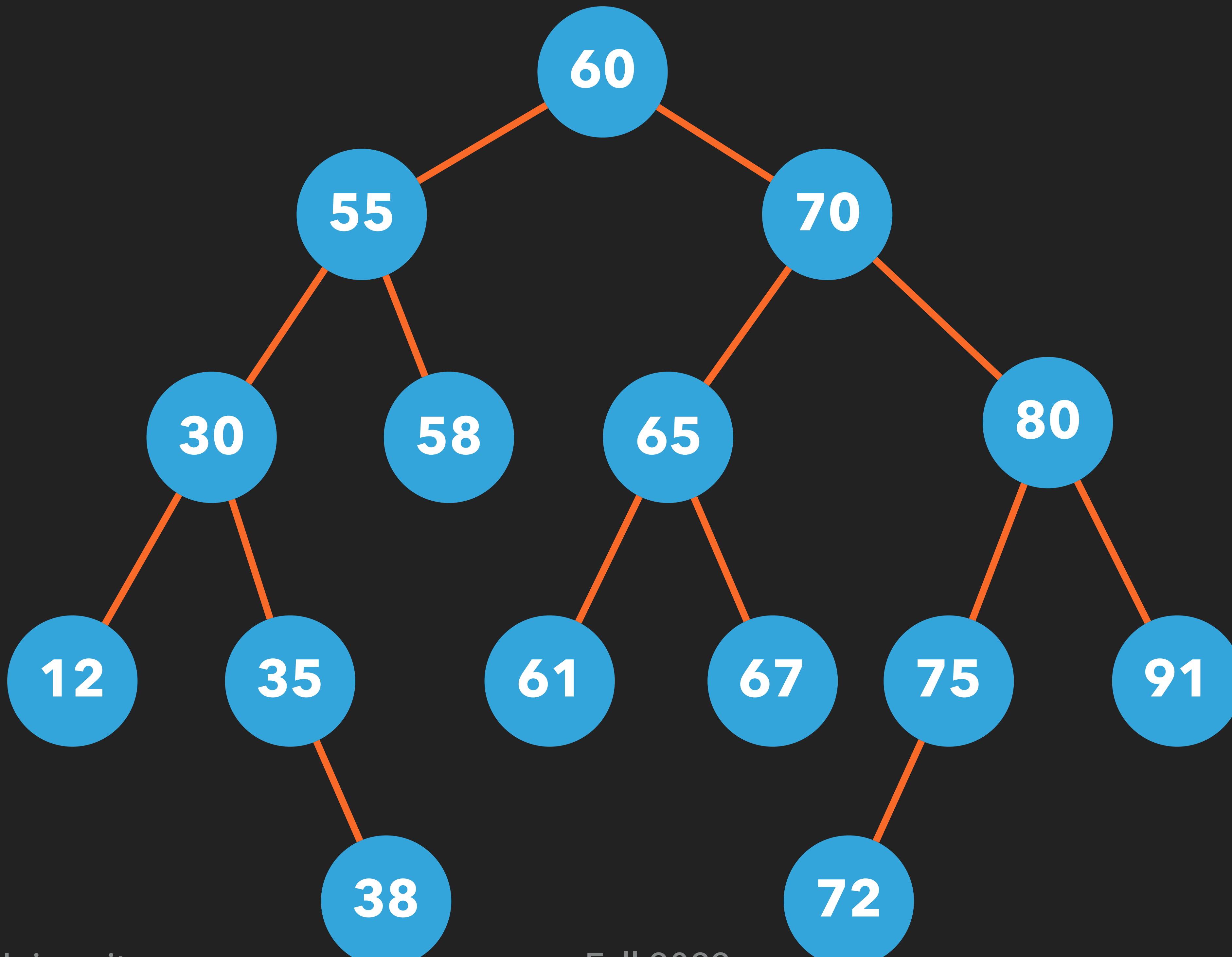
Delete the value 58 (Leaf)




```
Remove the
right child
of 55
```

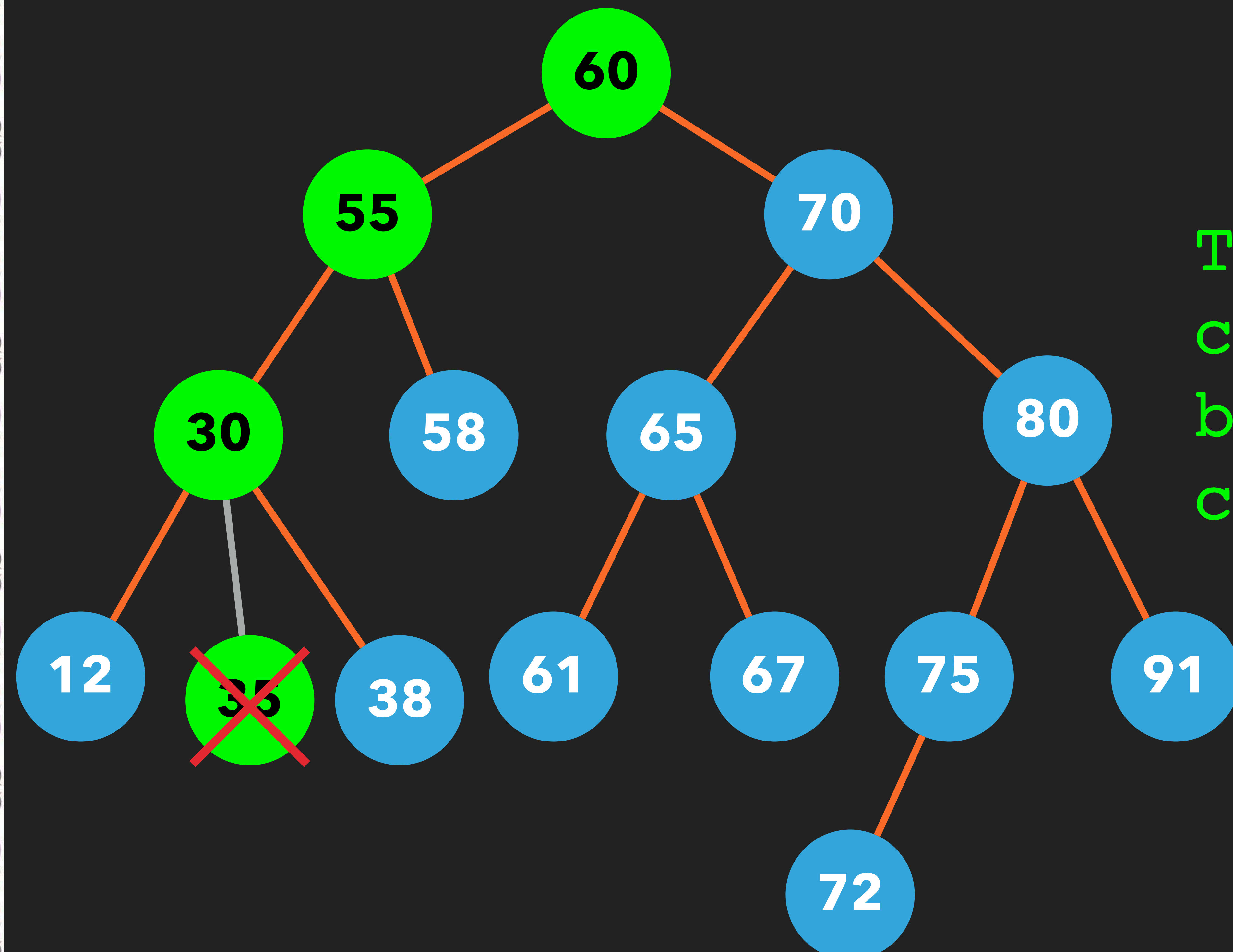
BST (Remove)

Delete the value 35 (one child)



BST (Remove)

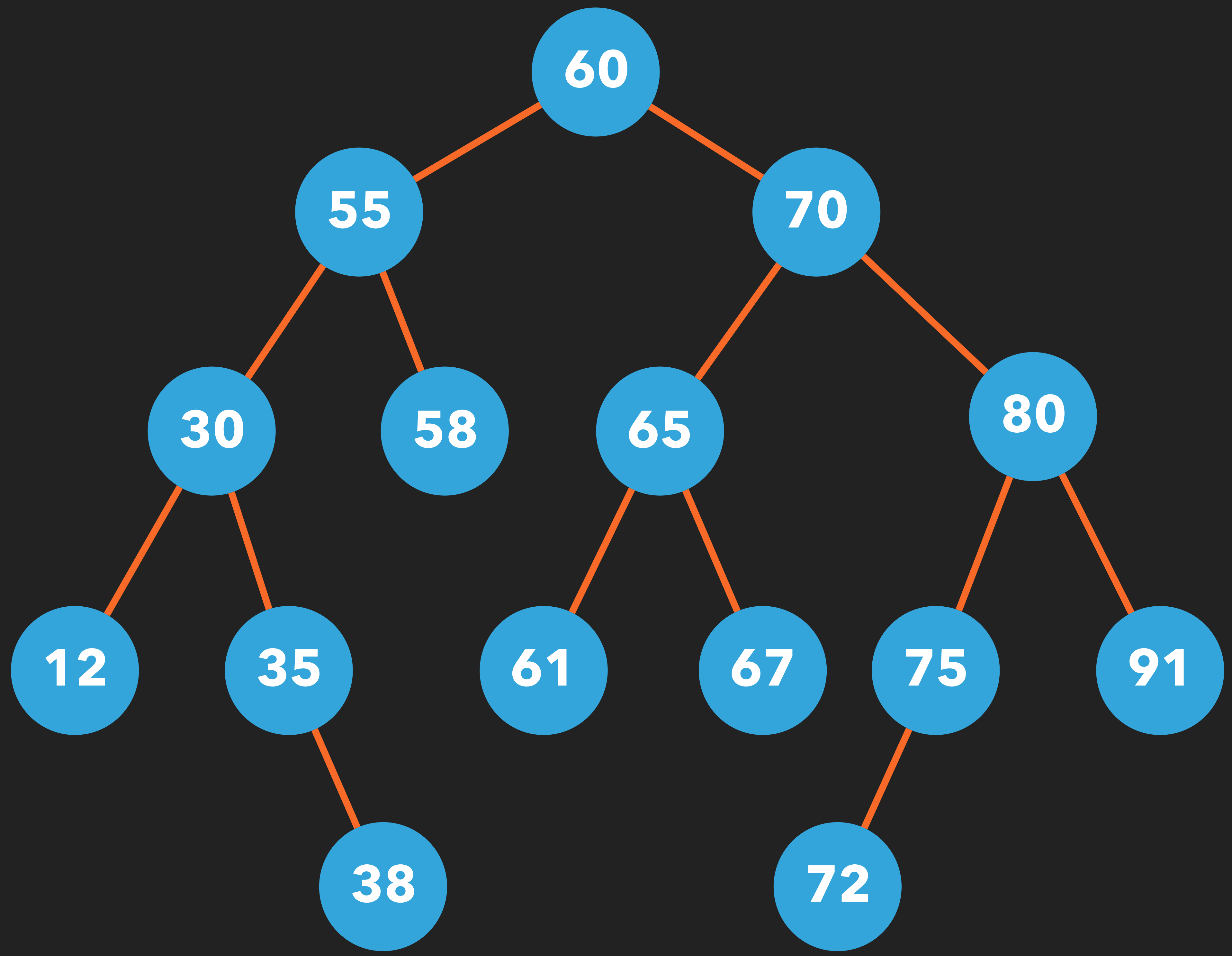
Delete the value 35 (one child)



The only child of 35 becomes the child of 30

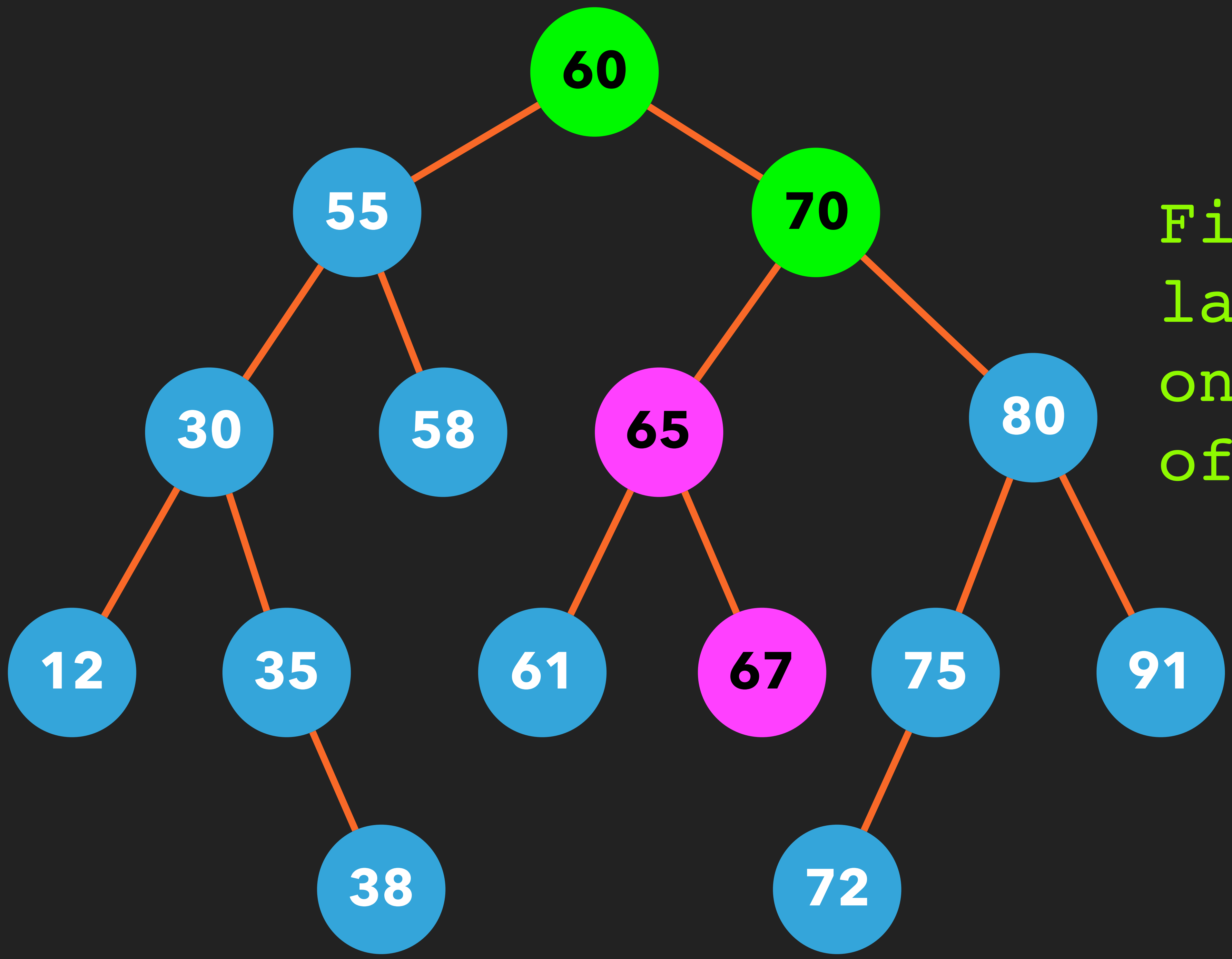
BST (Remove)

Delete the value 70 (two children)



BST (Remove)

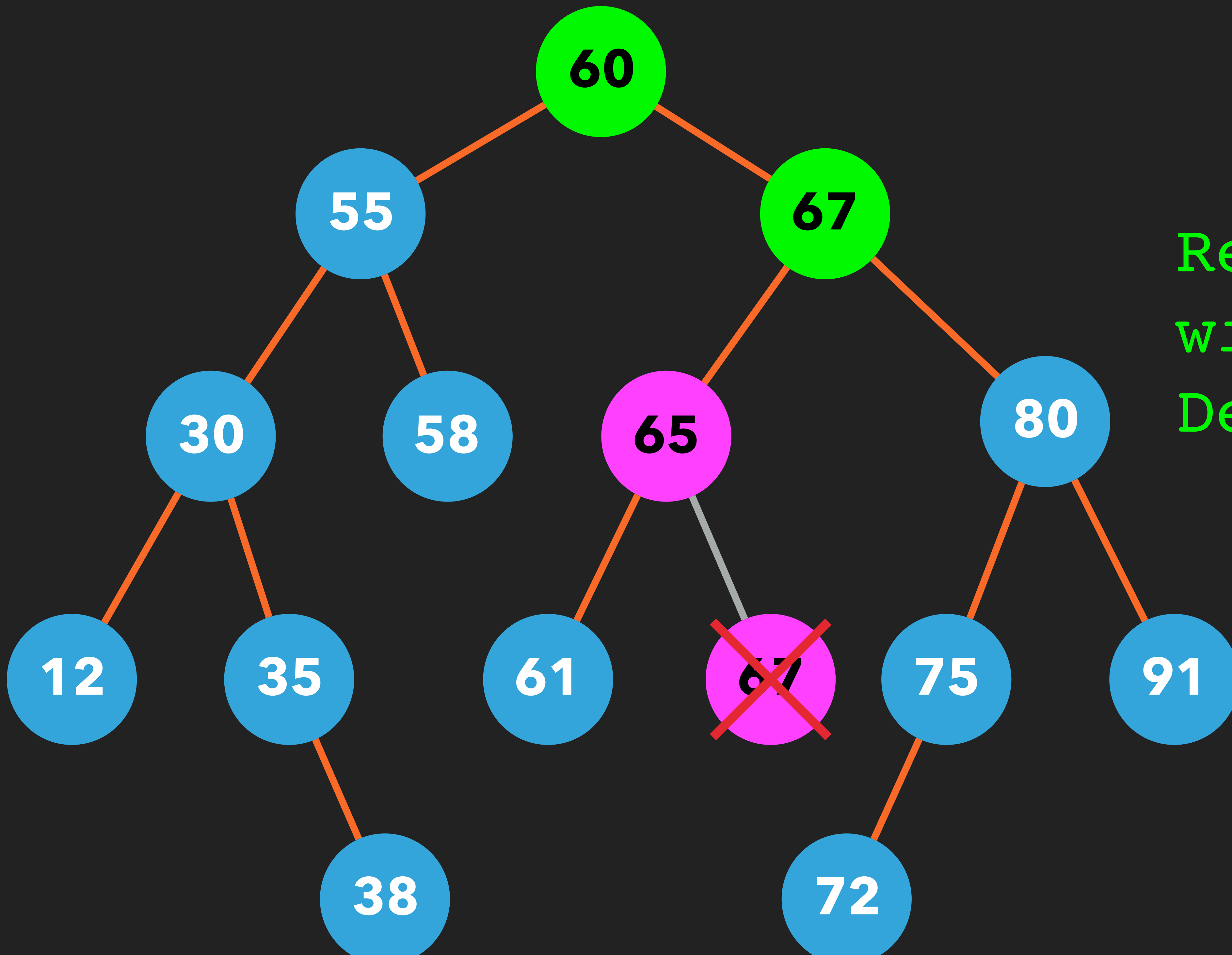
Delete the value 70 (two children)



Find the largest node on the left of 70

BST (Remove)

Delete the value 70 (two children)



Replace 70 with 67 and Delete 67

BST (Remove)

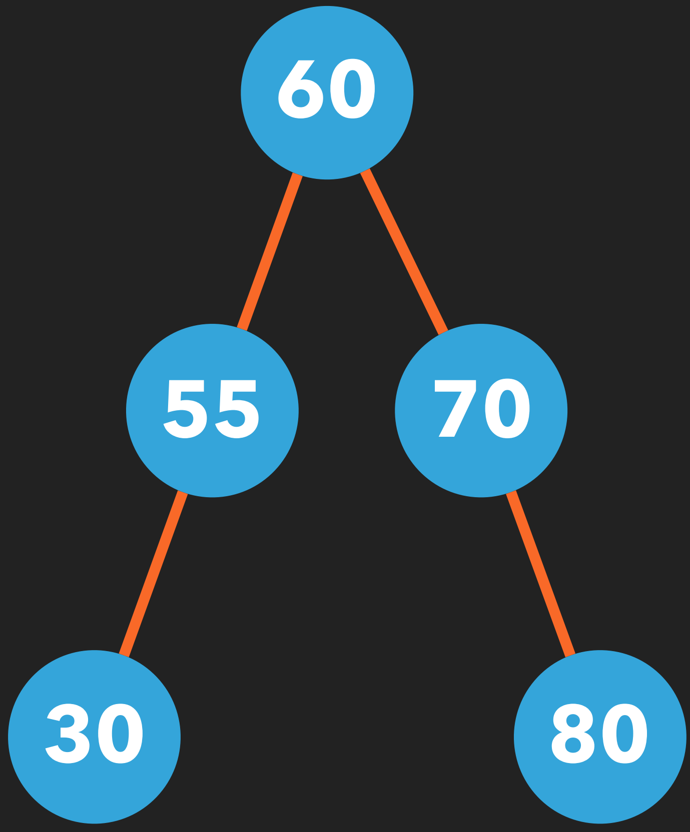
```
boolean remove (value)
  node = search(value) // find node with value first
  if (node == null)
     return false (value not found in the BST)
  else
   if (node has no children)
      remove link to node (parent points to null)
   else if (node has one child)
      replace node with its child
   else if (node has two children)
      find the largest node on the left subtree of node
      copy the value of the largest node to node
      remove the largest node
   end if
  end if
  return true
end remove
```

Traversals (Preorder)

```
preorder(){
  preorder(root)
}
preorder(node){
  if(node not null){
   print node
   preorder(left child of node)
   preorder(right child of node)
  }
}
```

Traversals (Preorder)

```
preorder(){
  preorder(60)
}
preorder(60){
   print 60                                                     60
   preorder(55) -> preorder(55){
                     print  55                                   55
                     preorder(30)  -> preorder(30){
                                       print 30                  30
                                       preorder(null)
                                       preorder(null)
                                      }
                     preorder(null)
                   }
   preorder(70) -> preorder(70){
                    print 70                                     70
                    preorder(null)
                    preorder(80)   -> preorder(80){
                                       print 80                  80
                                       preorder(null)
                                       preorder(null)
                                     }
                   }
}
```

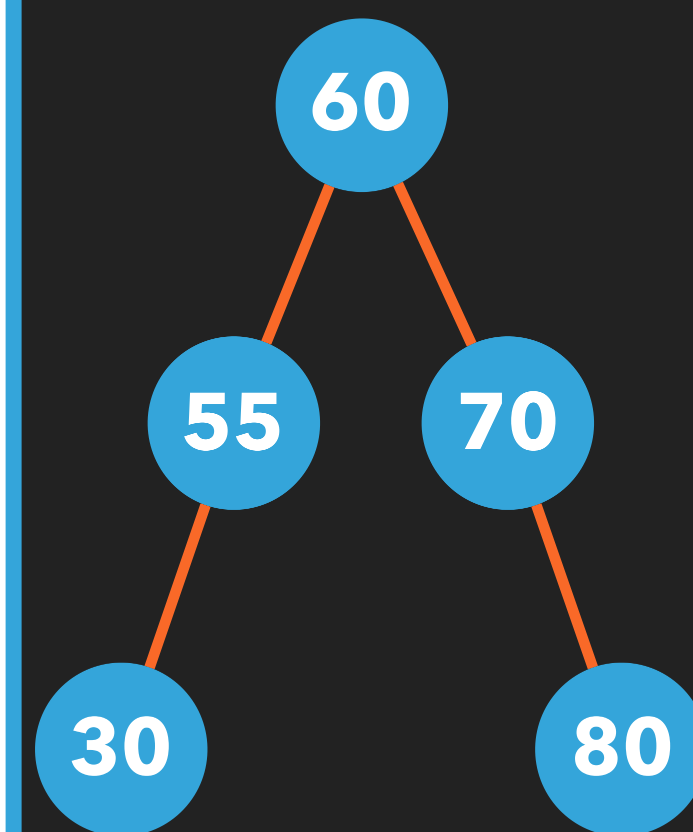


Traversals (Inorder)

```
inorder(){
  inorder(root)
}
inorder(node){
  if(node not null){
   inorder(left child of node)
   print node
   inorder(right child of node)
  }
}
```

Traversals (Inorder)

```
inorder(){
  inorder(60)
}
inorder(60){
   preorder(55) —> preorder(55){
                     preorder(30)  —> preorder(30){
                                       preorder(null)
                                       print 30            30
                                       preorder(null)
                                      }
                     print  55                             55
                     preorder(null)
                    }
   print 60                                                60
   preorder(70) —> preorder(70){
                    preorder(null)
                    print 70                               70
                    preorder(80)   —> preorder(80){
                                       preorder(null)
                                       print 80            80
                                       preorder(null)
                                     }
                    }
}
```

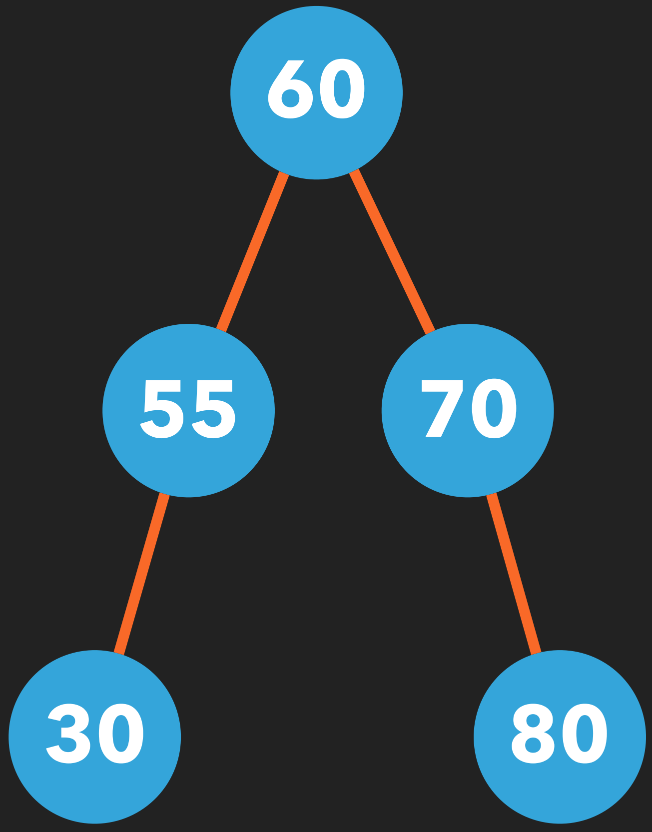


Traversals (Postorder)

```
postorder(){
  postorder(root)
}
postorder(node){
  if(node not null){
    postorder(left child of node)
    postorder(right child of node)
    print node

  }
}
```

Traversals (Preorder)

```
postorder(){
  postorder(60)
}
postorder(60){
   postorder(55) –> postorder(55){
                     postorder(30)  –> postorder(30){
                                        postorder(null)
                                        postorder(null)
                                        print 30            30
                                       }

                      postorder(null)
                      print  55                             55
                   }

   postorder(70) –> postorder(70){
                    postorder(null)
                    postorder(80)   –> postorder(80){
                                        postorder(null)
                                        postorder(null)
                                        print 80            80
                                       }
                   }
                   print 70                                 70
   print 60                                                 60
}
```



BST implementation

- BST may be implemented in two ways
  - Array Based BST
  - Linked BST

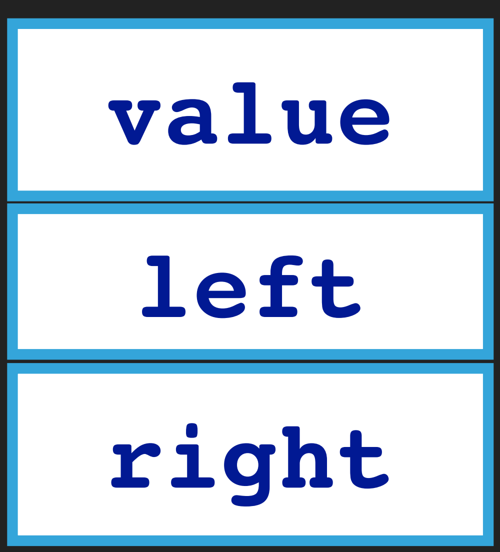
BST implementation (ArrayList)

- Nodes of the tree are stored in an array
- Children of a node follow the node (at specific indices)
- Waste of space if the BST is not full

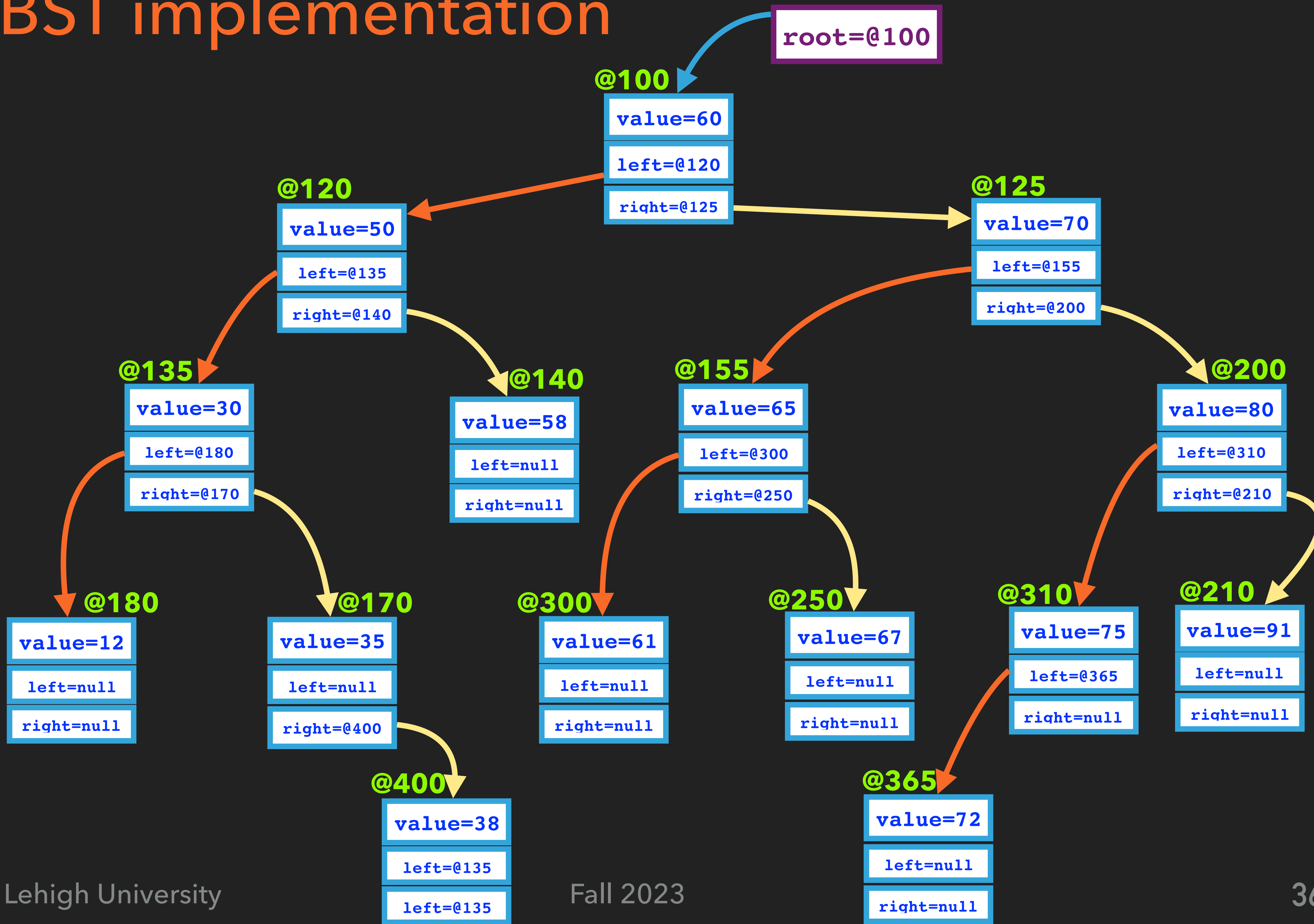
BST implementation (Linked nodes)

- Nodes of the tree are linked
- Every node has a value and two references, one to the left child and one to the right child

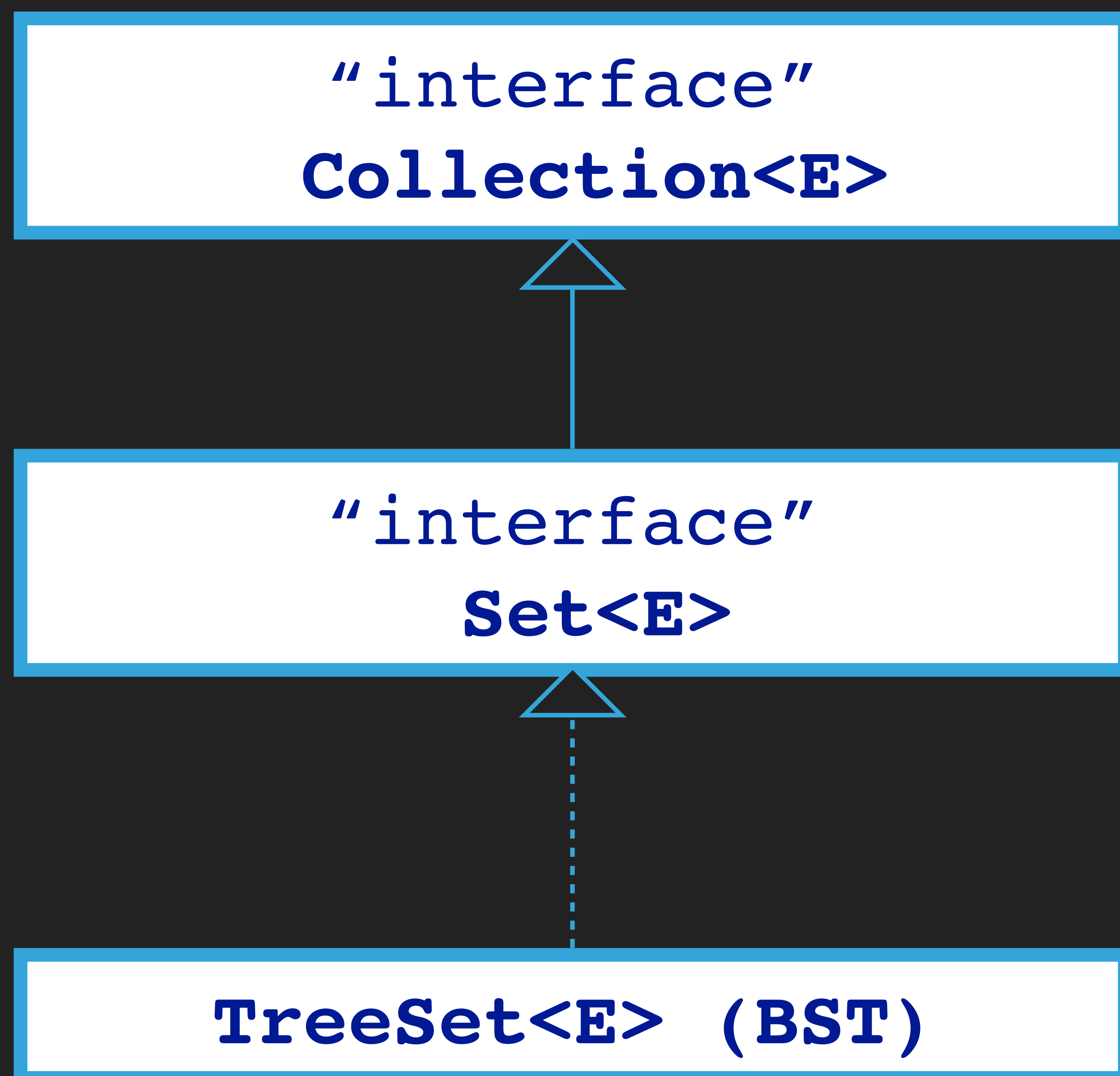
TreeNode



BST implementation



BST implementation



BST implementation

```
BST<E extends Comparable<E>>
-root: TreeNode
-size: int
+BST()
+size(): int
+isEmpty(): boolean
+clear(): void
+contains(E): boolean
+add(E): boolean
+remove(E): boolean
+inorder(): void
+preorder(): void
+postorder(): void
```

has

```
TreeNode
value: E
Left: TreeNode
Right: TreeNode
TreeNode(E val)
```

BST

Test.java

```
public class Test {
    public static void main(String[] args) {
        BST<String> bst = new BST<>();
        bst.add("Kiwi");
        bst.add("Strawberry");
        bst.add("Apple");
        bst.add("Banana");
        bst.add("Orange");
        bst.add("Lemon");
        bst.add("Watermelon");
        System.out.print("BST:  ");
        bst.inorder();
        System.out.println();
        System.out.println("BST contains Banana? " + bst.contains("Banana"));
        bst.remove("Banana");
        System.out.println("BST contains Banana? " + bst.contains("Banana"));
        System.out.print("BST:  ");
        bst.inorder();
        System.out.println();
        bst.remove("Orange");
        System.out.print("BST:  ");
        bst.inorder();
        System.out.println();
        bst.remove("Kiwi");
        System.out.print("BST:  ");
        bst.inorder();
        System.out.println();
    }
}
```

BST

The order in which the values are added to the BST affects its balance (shape)

```
public class Test {
    public static void main(String[] args) {
        BST<String> bst = new BST<>();
        bst.add("Apple");
        bst.add("Banana");
        bst.add("Kiwi");
        bst.add("Lemon");
        bst.add("Orange");
        bst.add("Strawberry");
        bst.add("Watermelon");
        System.out.print("BST: ");
        bst.inorder();
    }
}
```

BST

- Complexity of the BST operations

Method	Complexity	Method	Complexity
BST()	O(1)	remove(E)	O(log n) O(n)
size()	O(1)	contains(E)	O(log n) O(n)
clear()	O(1)	inorder()	O(n)
isEmpty()	O(1)	**preorder()**	O(n)
add(E)	O(log n) O(n)	**postorder()**	O(n)

Summary

- ✦ Binary Search Tree
- ✦ Operations: Search, Add, Remove, Traversals
- ✦ Implementation - Linked Nodes
- ✦ The order in which data is added has an effect on the shape of the BST (balance)
- ✦ Balanced BSTs: AVL trees, Red-Black trees