Binary Search Trees

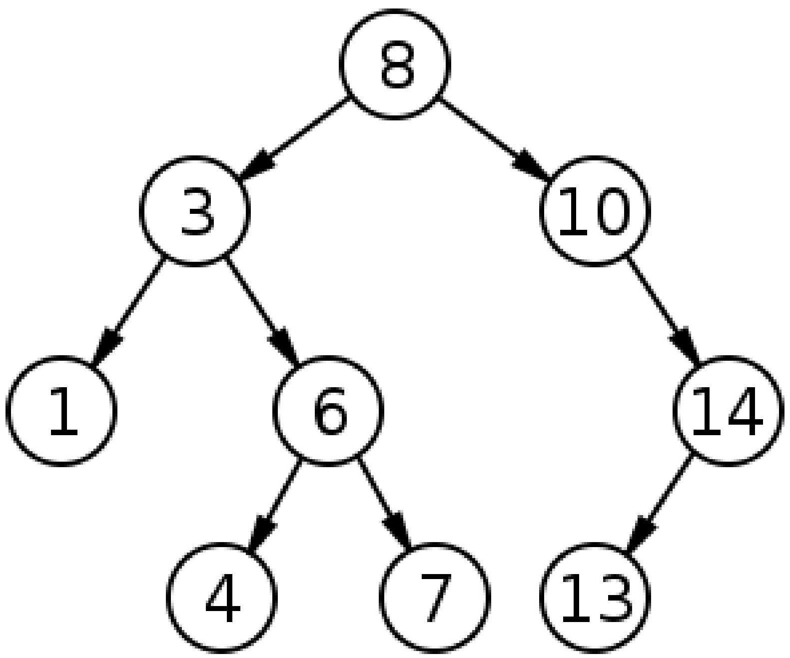
Section 6.8

Trees are efficient

- Many algorithms can be performed on trees in O(log n) time.
- Searching for elements using a binary search can work on a tree if the elements are ordered in the obvious way.
- Adding and removing elements is a little trickier.

The Binary Search Tree property

- All values in the nodes in the left subtree of a node are less than the value in the node.
- All values in the nodes in the right subtree of a node are greater than the value in the node.

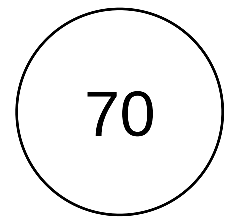


http://en.wikipedia.org/wiki/Binary_search_tree

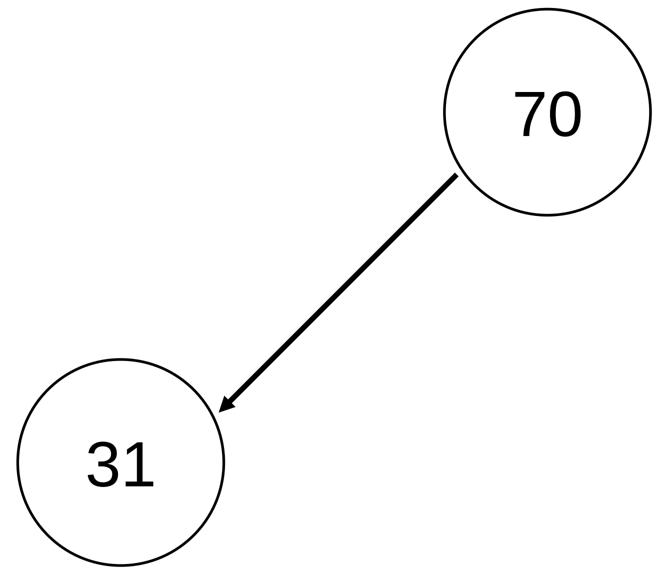
Constructing a BST

- We can go through a list of elements adding them in the order they occur.
  - e.g. 70, 31, 93, 94, 14, 23, 73

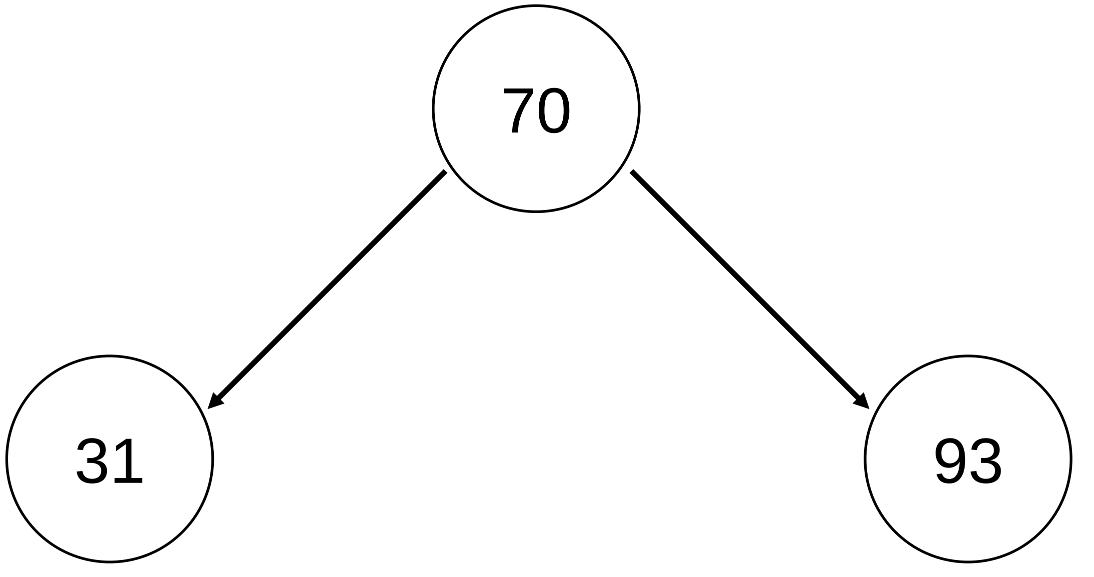
70, 31, 93, 94, 14, 23, 73



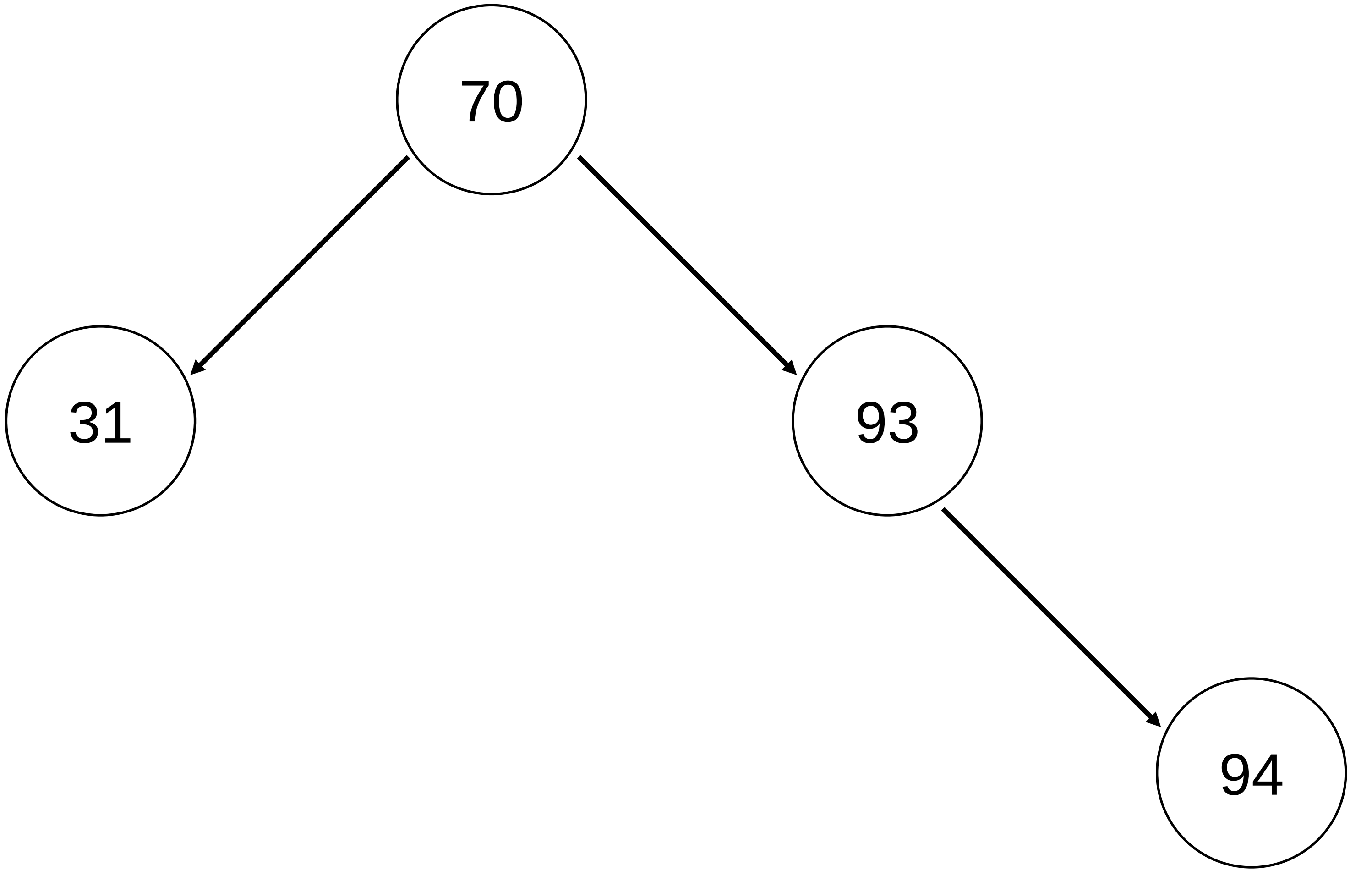
70, **31**, 93, 94, 14, 23, 73



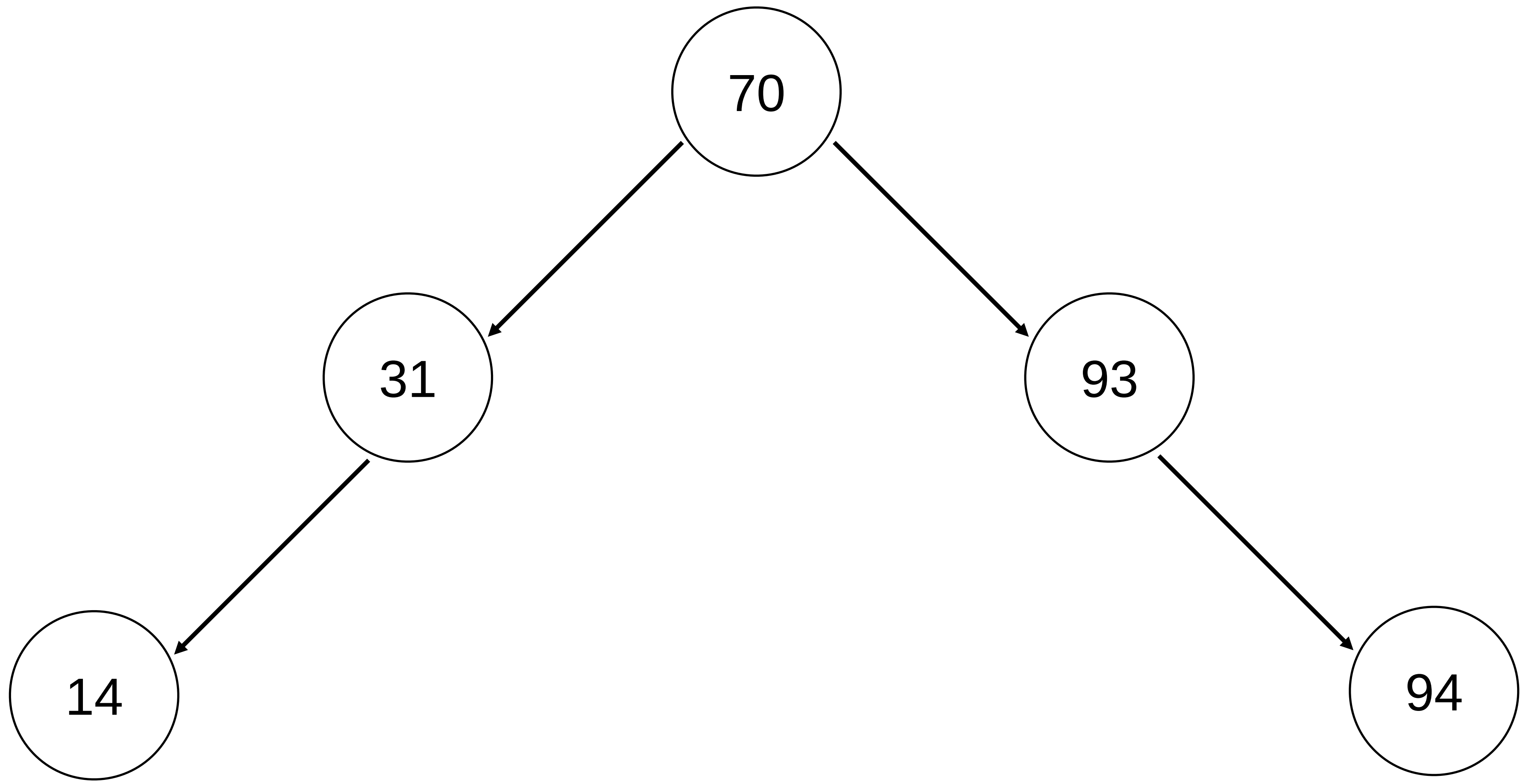
70, 31, **93**, 94, 14, 23, 73



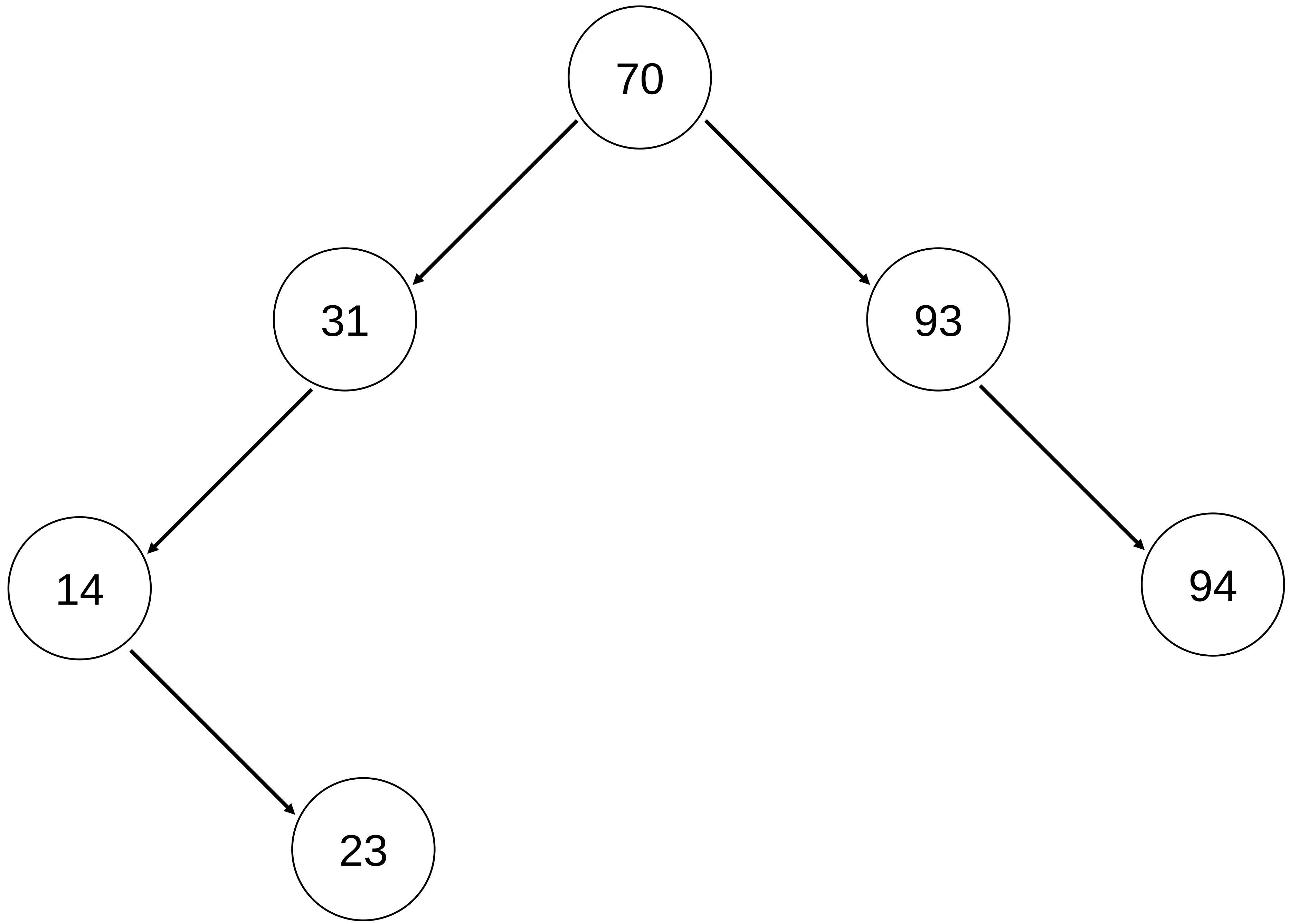
70, 31, 93, **94**, 14, 23, 73



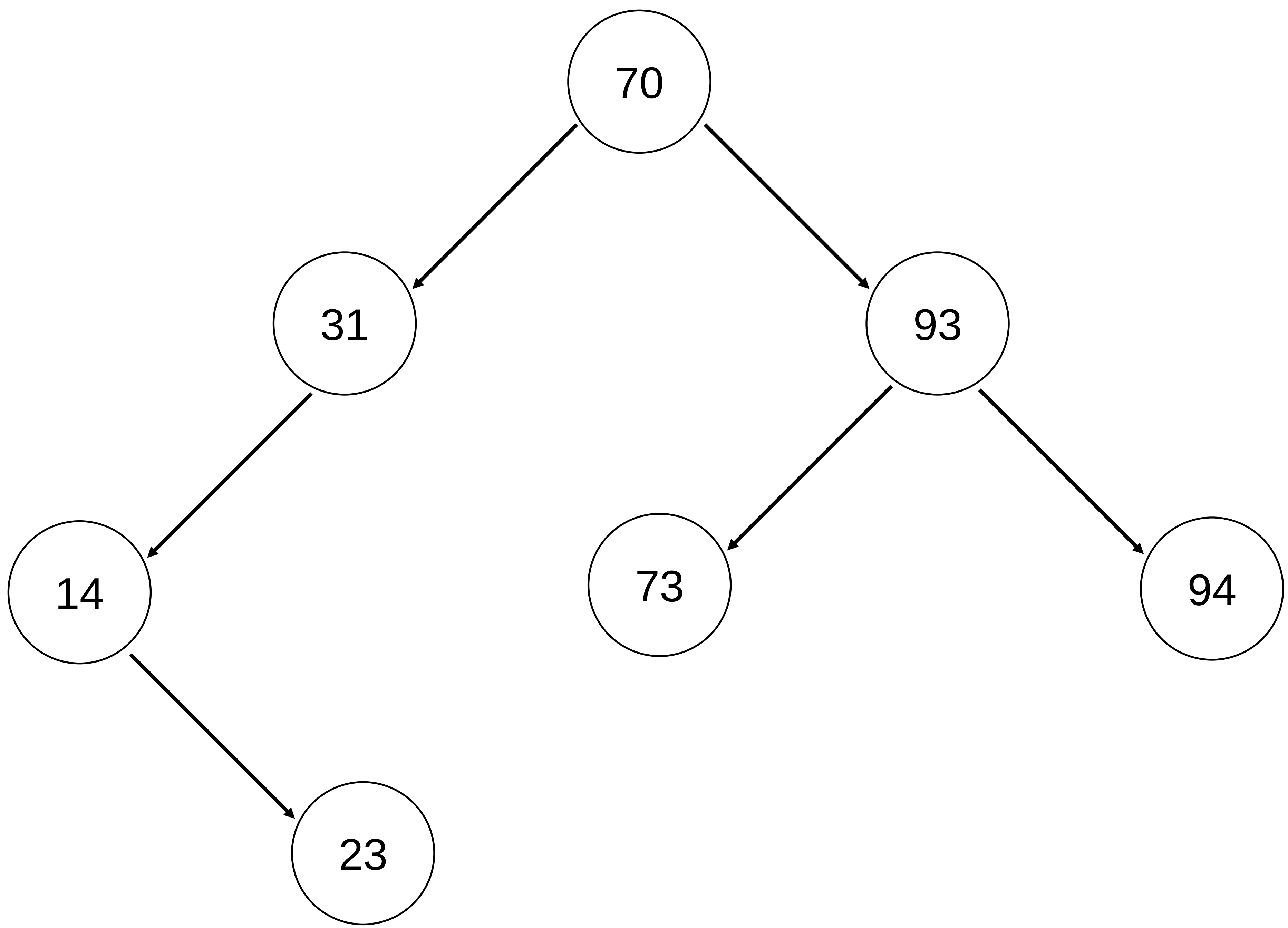
70, 31, 93, 94, **14**, 23, 73



70, 31, 93, 94, 14, **23**, 73



70, 31, 93, 94, 14, 23, 73



Your turn

- Add the elements 17, 5, 25, 2, 11, 35, 9, 16, 29, 38, 7 to a binary search tree

Map ADT and BSTs

- If we use a key as the ordering component in our BSTs we can also store a separate value.
- We can then use a BST as a Map with functions such as:
  - put(key, value) - stores value using key
  - get(key) - returns the value found from key
- The text book does this.
- Most introductions just use a value - this is what I will use.

Binary Search Tree code

```
class BST:
    """A Binary Search Tree (BST) class."""

    def __init__(self, value, parent=None):
        """Construct a BST.

        value -- the value of the root node
        parent -- the parent node (of this BST subtree)
        """
        self.value = value
        self.left = None
        self.right = None
        self.parent = parent # useful for some operations
```

Inserting a value

```
def insert(self, value):
    """Insert value into the BST."""
    if value == self.value: # already in the tree
        return
    elif value < self.value:
        if self.left:
            self.left.insert(value)
        else:
            self.left = BST(value, parent=self)
    else:
        if self.right:
            self.right.insert(value)
        else:
            self.right = BST(value, parent=self)
```

A Factory Method

```
    def create(a_list):
        """Create a BST from the elements in a_list."""
        bst = BST(a_list[0])
        for i in range(1, len(a_list)):
            bst.insert(a_list[i])
        return bst

# A factory method is one which creates and returns
# a new object.

# e.g. this would be called like this
bst = BST.create([70, 31, 93, 94, 14, 23, 73])
```

Doing something in order

```
def inorder(self, function):
    """Traverse the BST in order performing function."""
    if self.left: self.left.inorder(function)
    function(self.value)
    if self.right: self.right.inorder(function)

# for example:
bst = BST.create([70, 31, 93, 94, 14, 23, 73])
bst.inorder(print)

# The output is 14 23 31 70 73 93 94
```

Your turn

- Write a __contains__ method which returns True if the BST contains the value, otherwise False.

```
def __contains__(self, value):
```

Deleting a value

- We need to find the node the value is stored in.
- There are three cases
  - the node has no children
  - the node has one child
  - the node has two children

Finding the node

```
def locate(self, value):
    """Return the node holding value."""
    if value == self.value:
        return self
    elif value < self.value and self.left:
        return self.left.locate(value)
    elif value > self.value and self.right:
        return self.right.locate(value)
    else:
        return None
```

No children

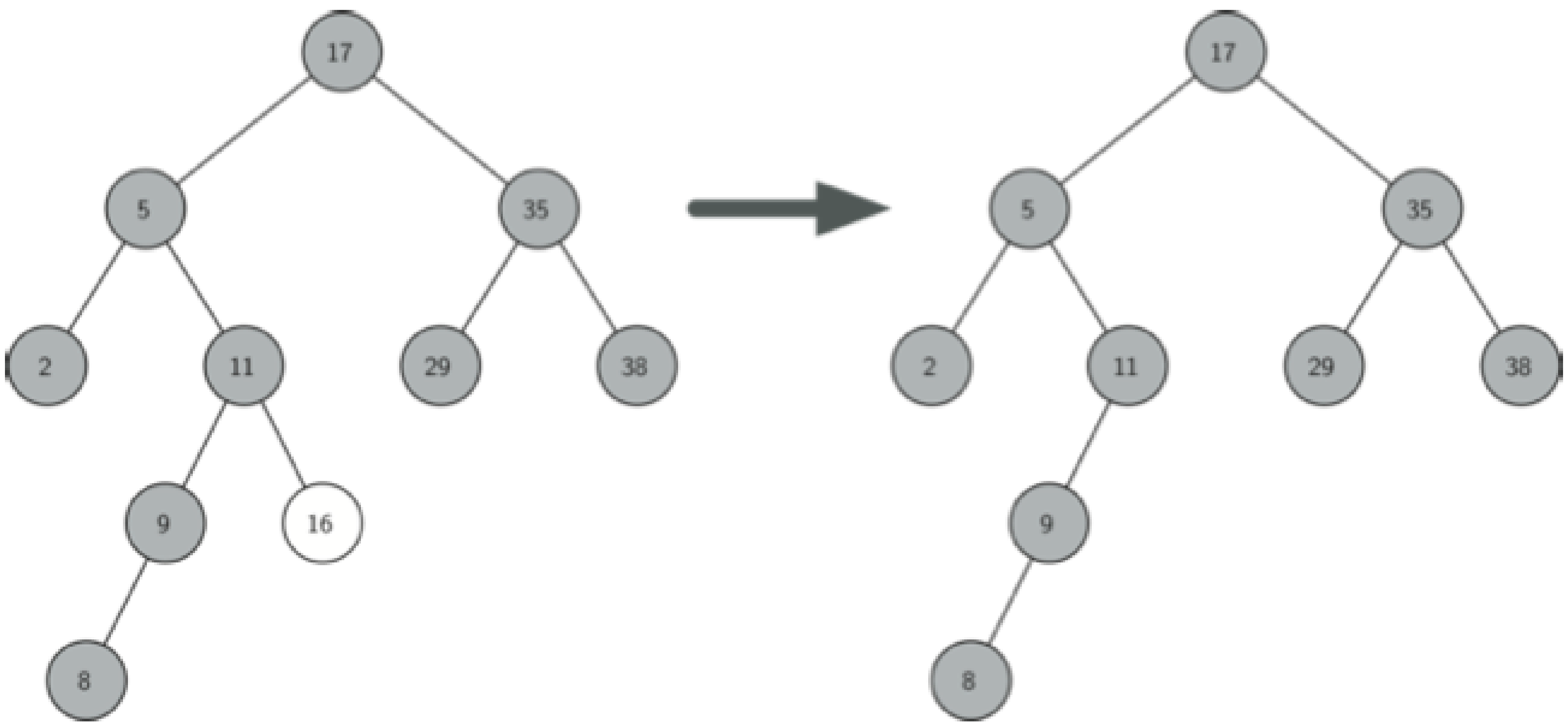


Figure 6.20: Deleting Node 16, a Node without Children

No children

- Just delete the node and fix up its parent.

One child

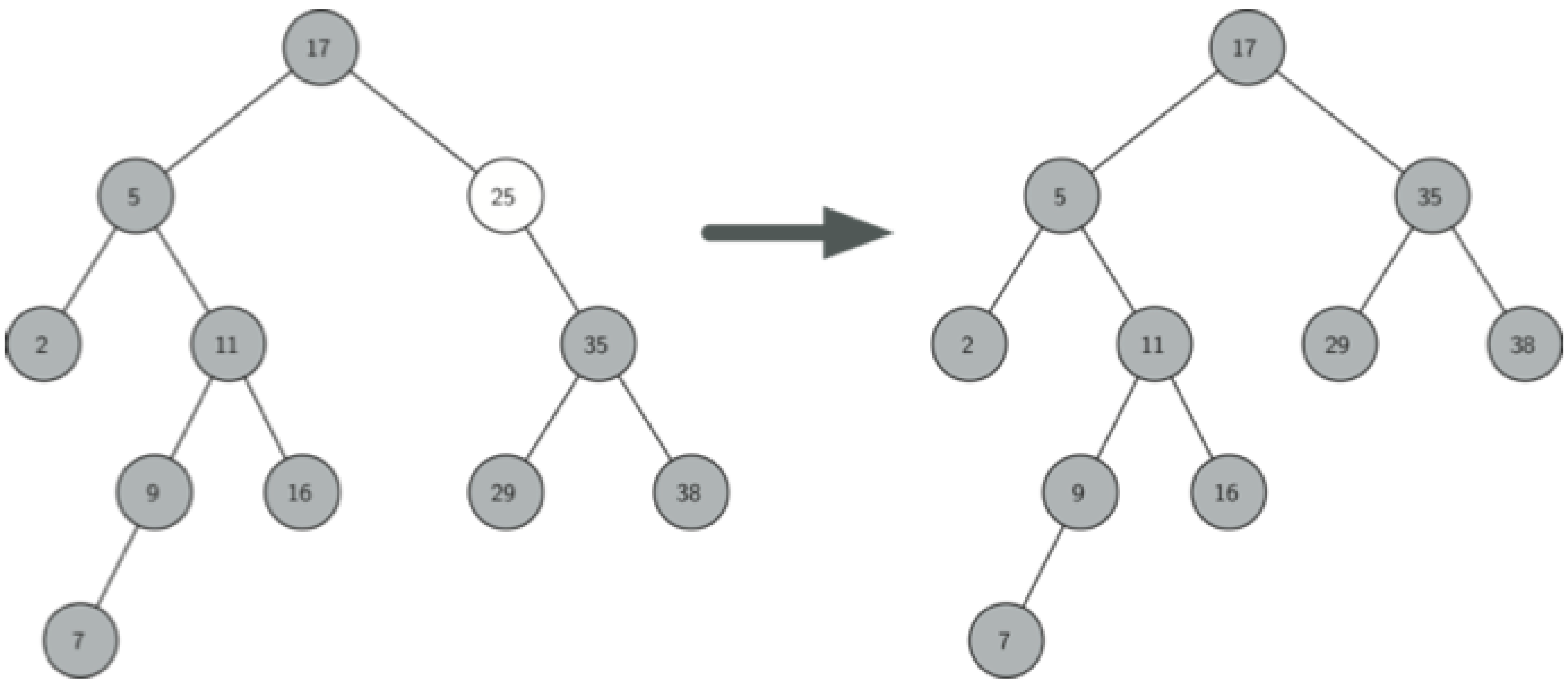


Figure 6.21: Deleting Node 25, a Node That Has a Single Child

One child

- Delete the node and shift its child up to take its place by changing the parent.

Two children

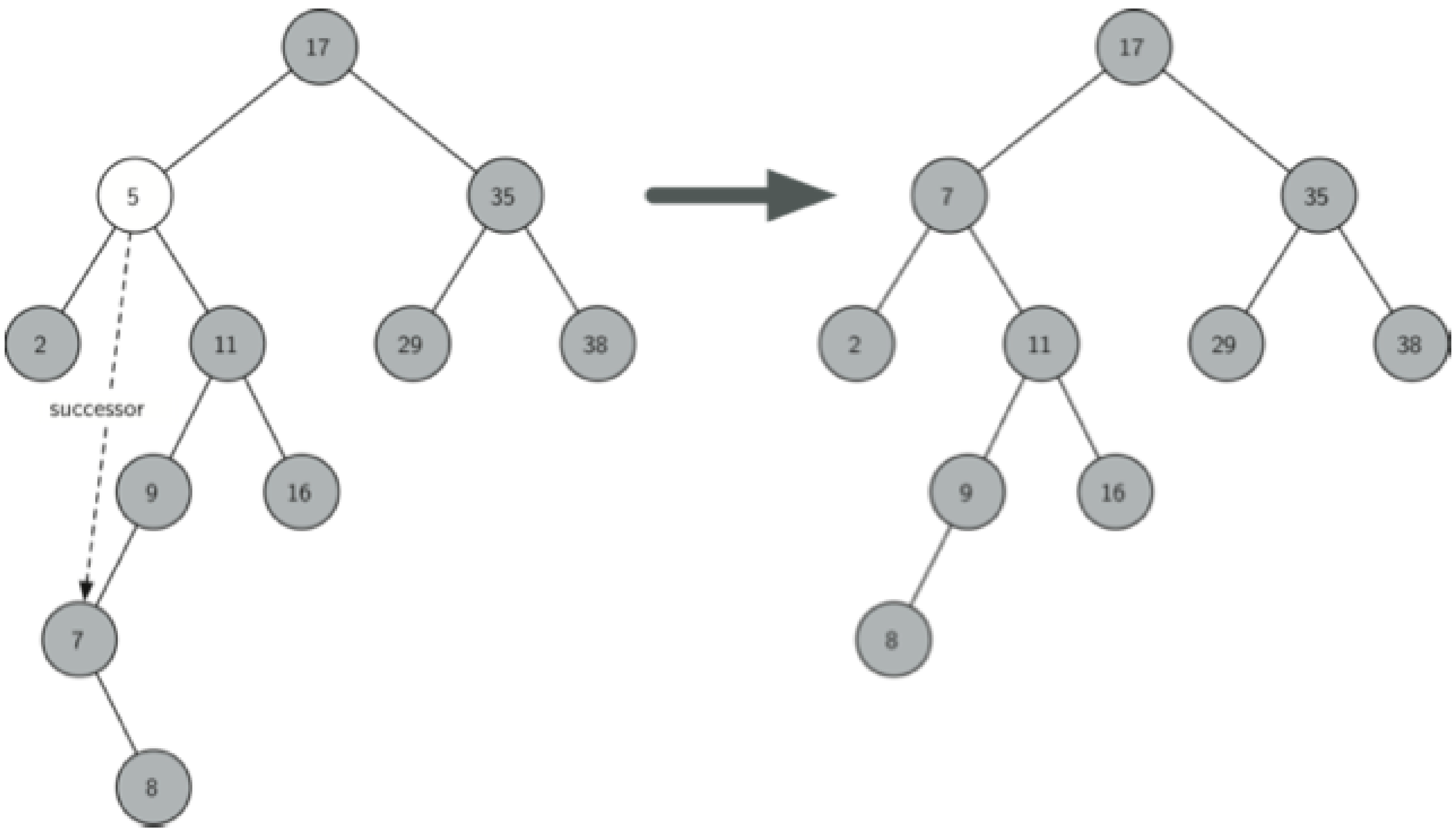


Figure 6.22: Deleting Node 5, a Node with Two Children

Two children

- Replace the value in the node with its inorder successor.
- We also have to delete the inorder successor node.
  - But this can’t have more than one child.
    - Why not?