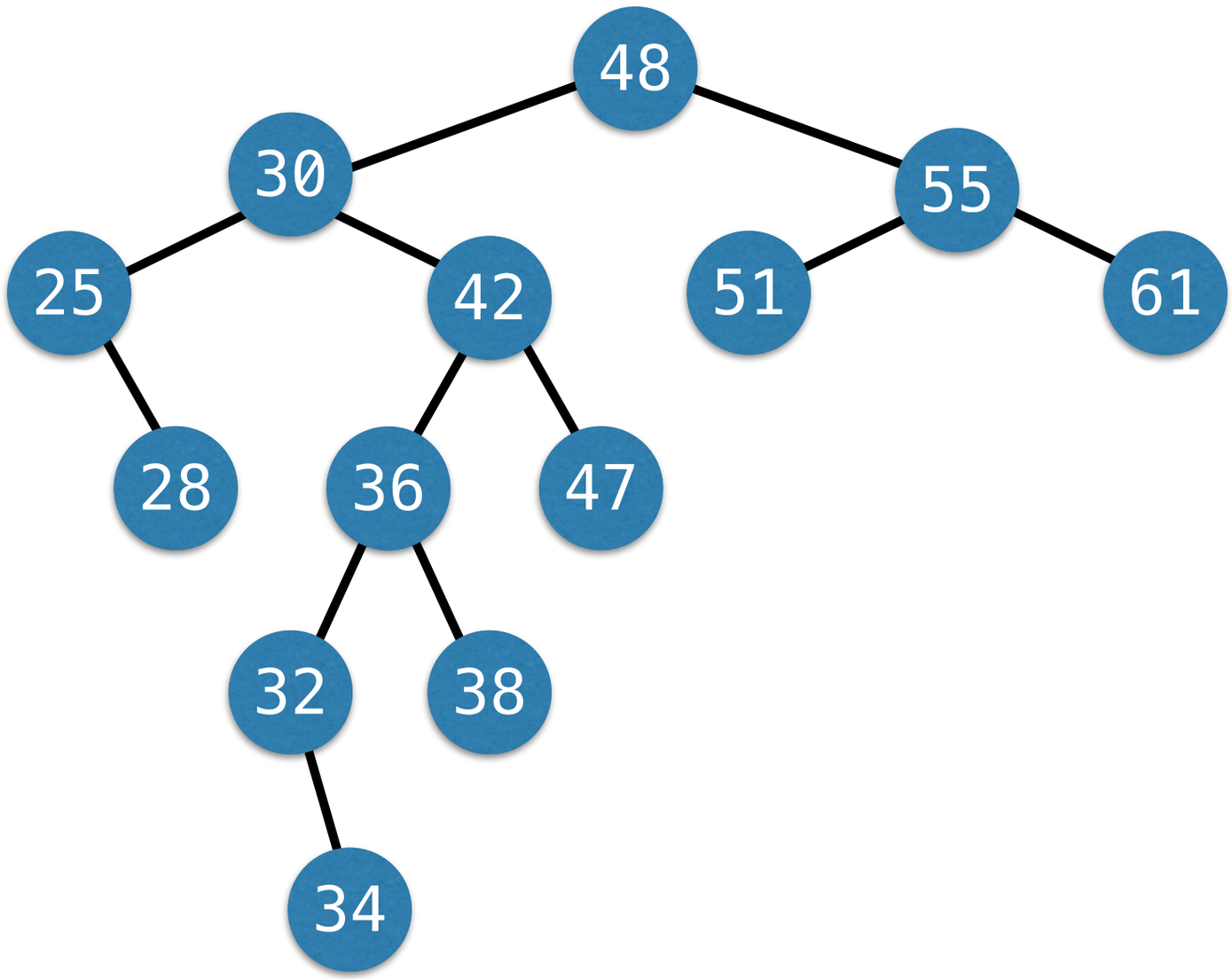
Deletion

`remove()`

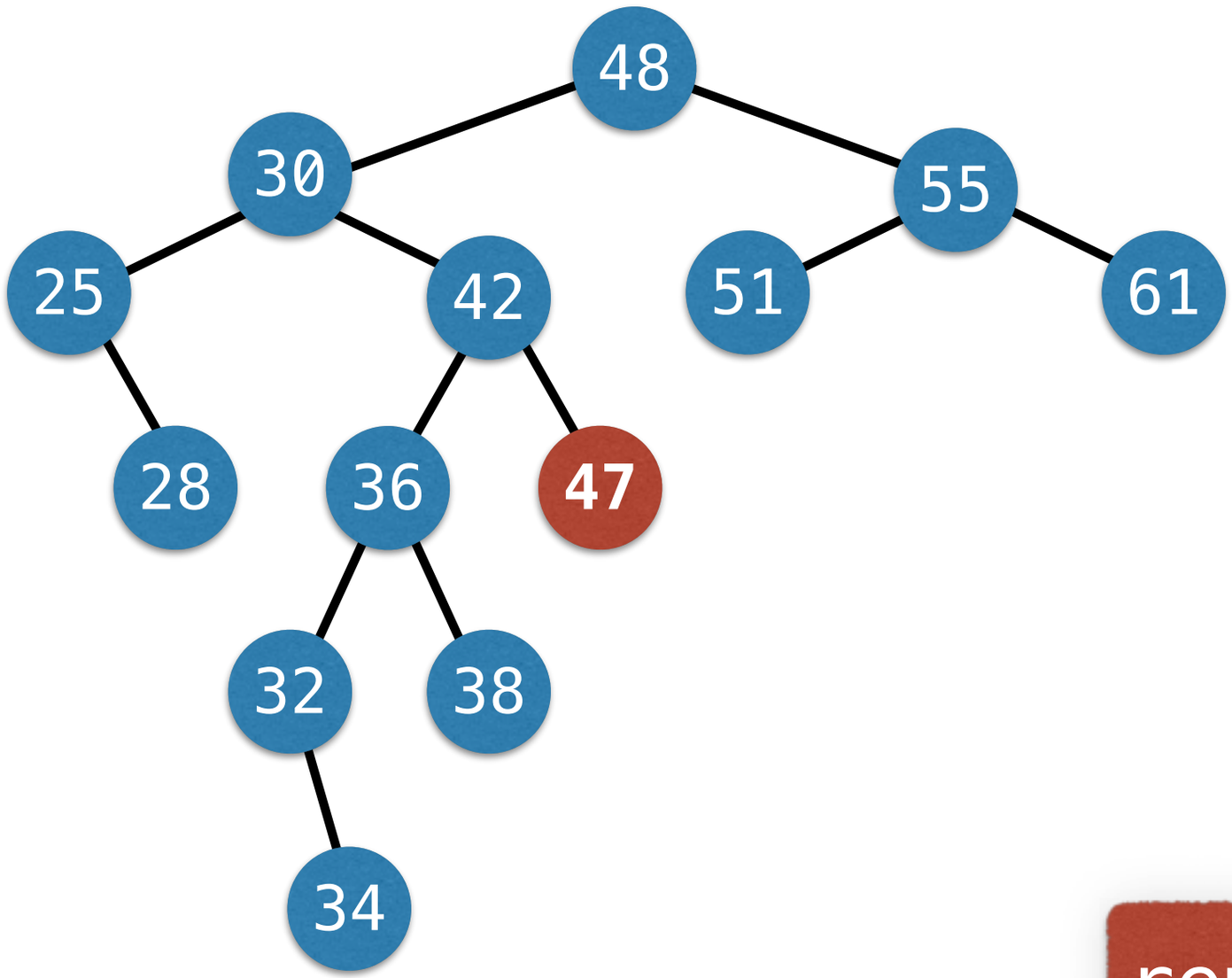
- Find the node `n` containing the key.
  - If not found, return `false`.
- Unlink `n`.
  - Return `true`.

```
public boolean remove(Object obj)
{
   E key = (E) obj;
   Node n = findEntry(key);
   if (n == null)
   {
      return false;
   }
   unlinkNode(n);
   return true;
}
```

Deleting a Leaf

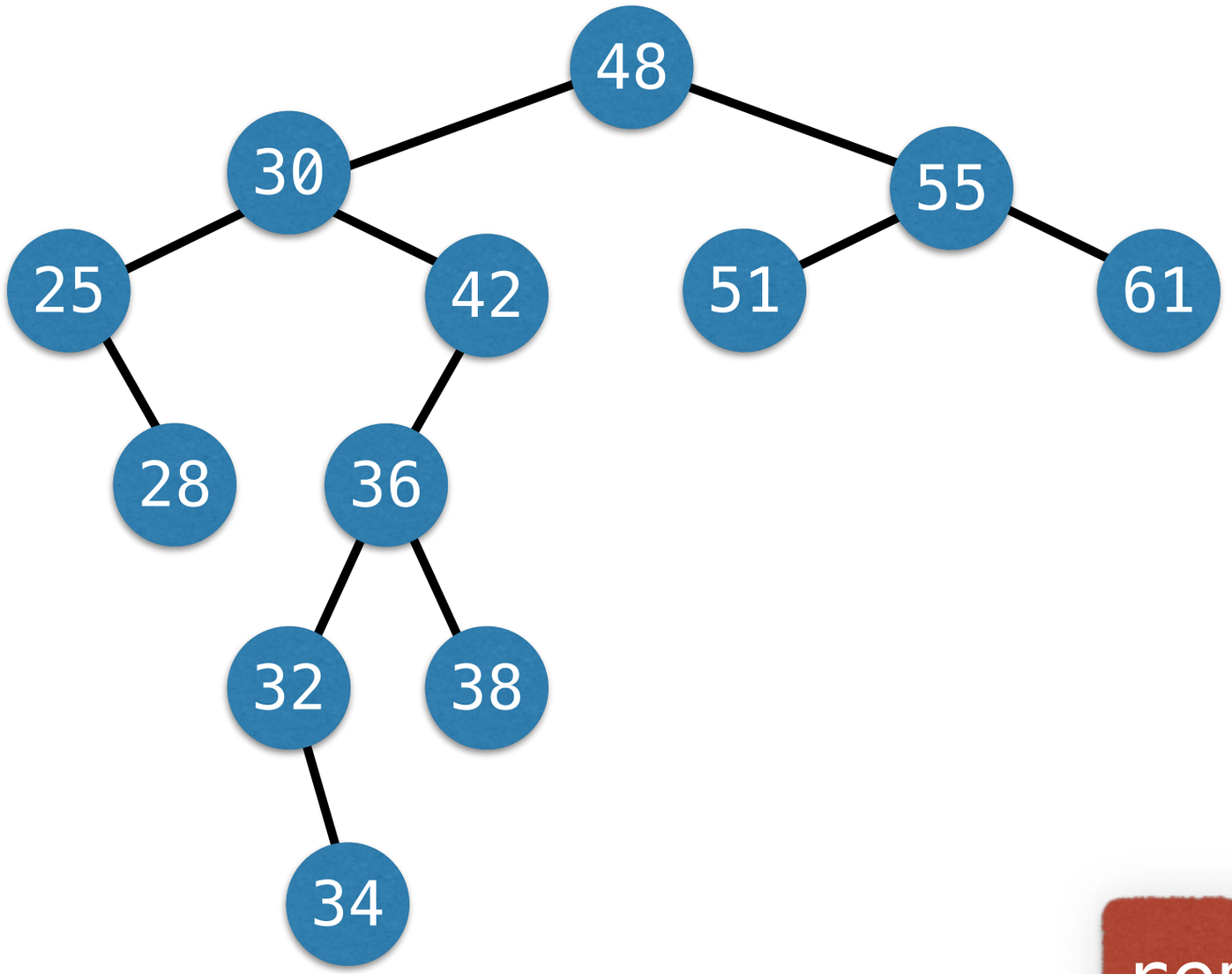


Deleting a Leaf



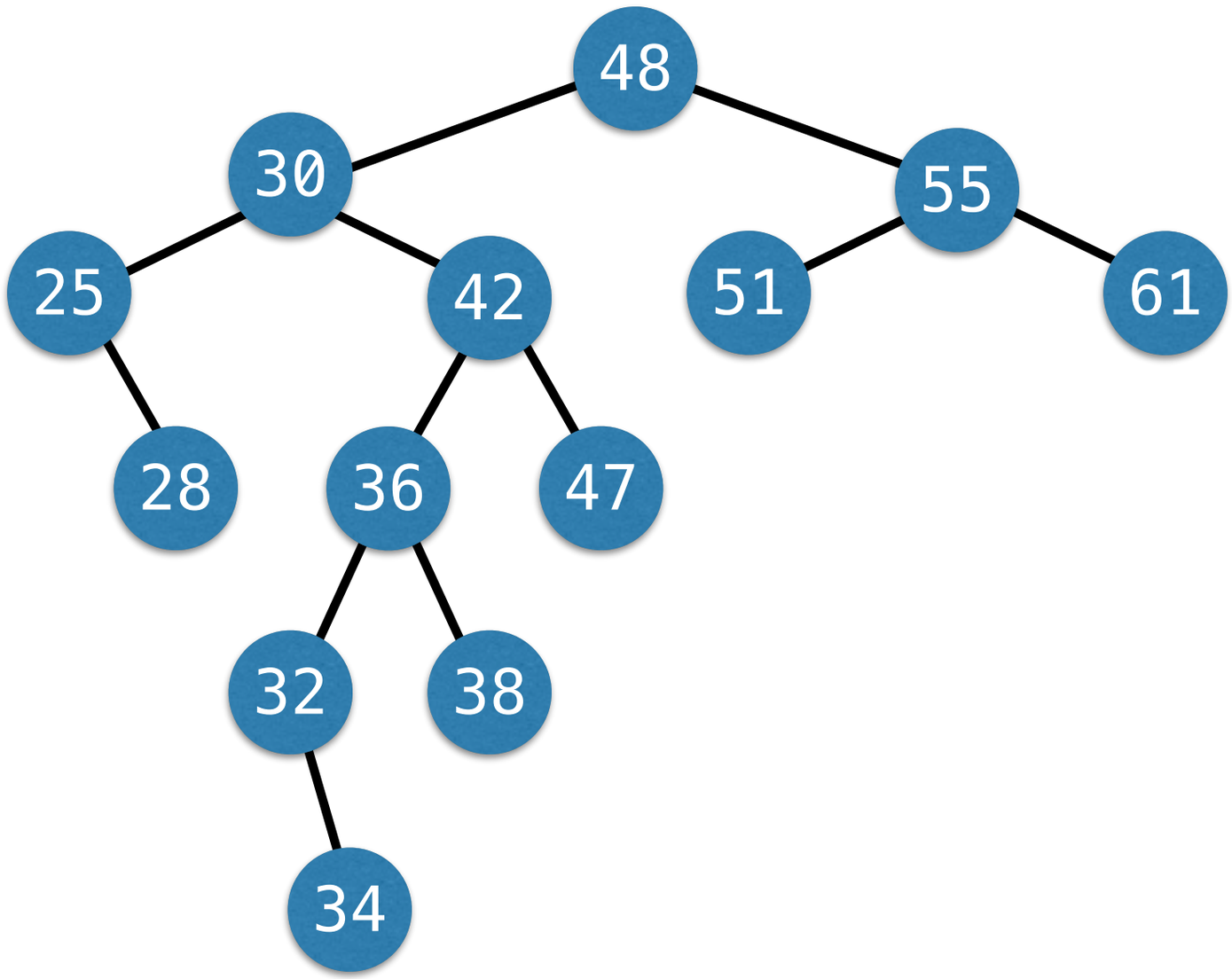

```
remove(47)
```

Deleting a Leaf

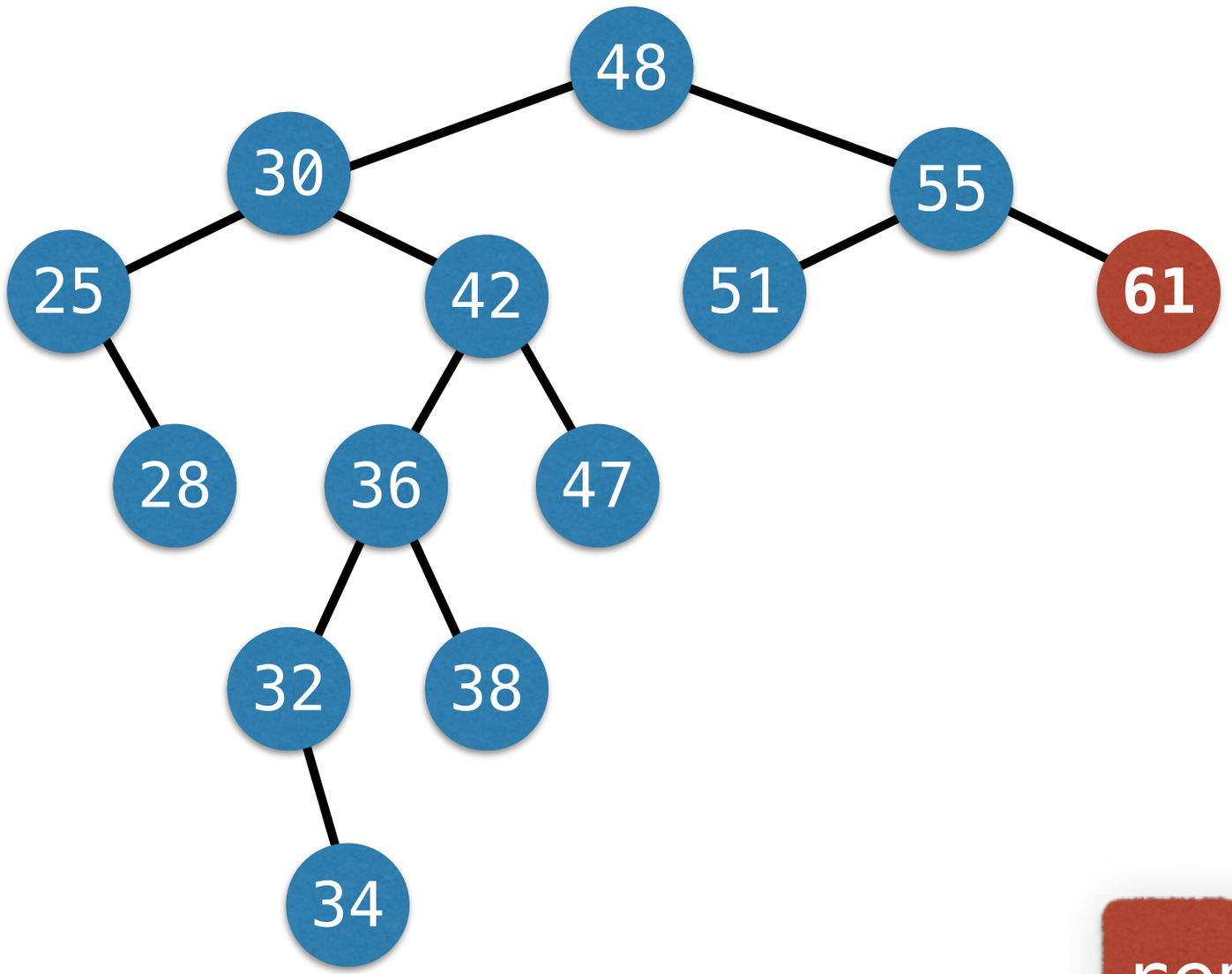


remove(47)

Deleting a Leaf

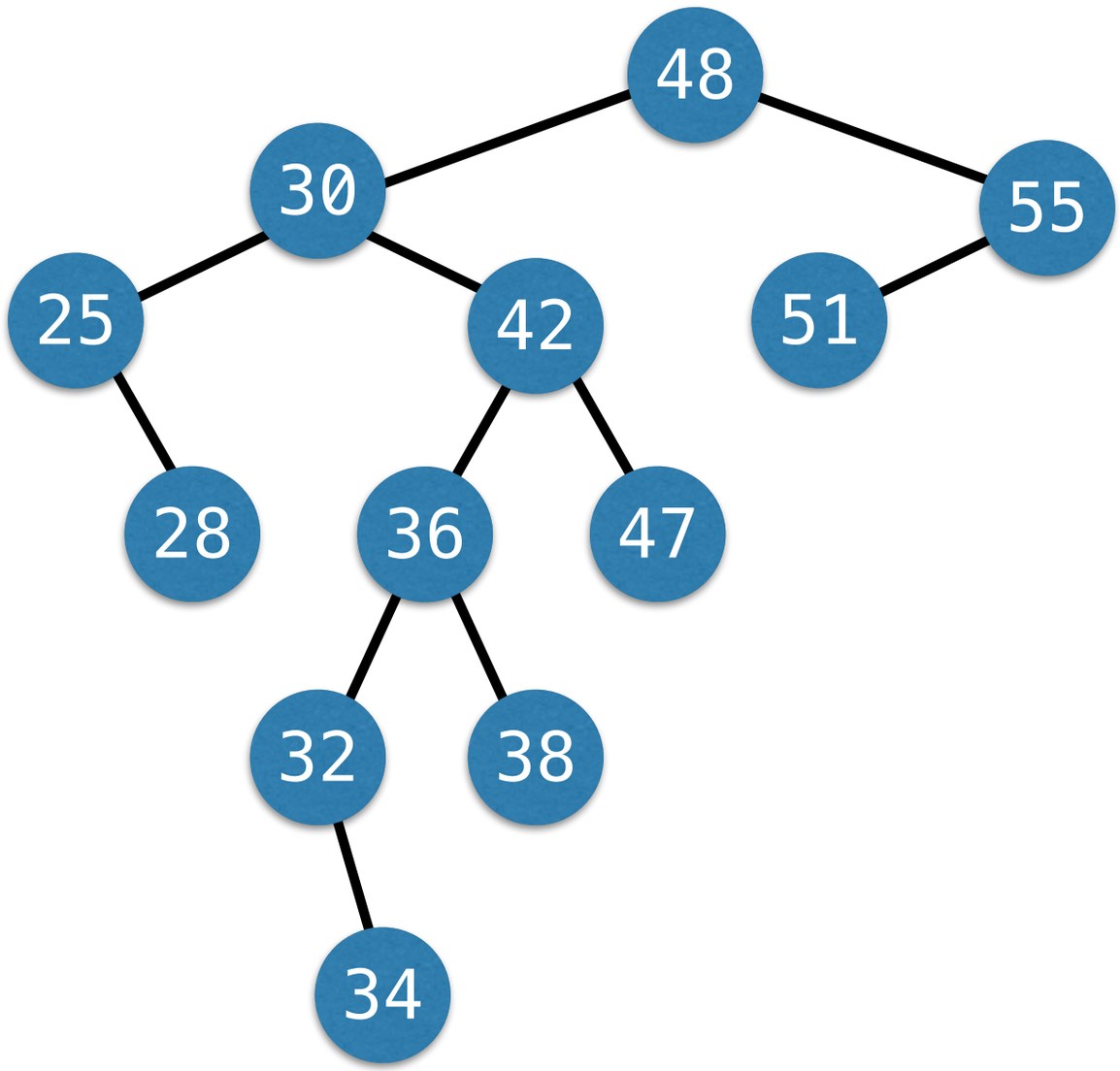


Deleting a Leaf



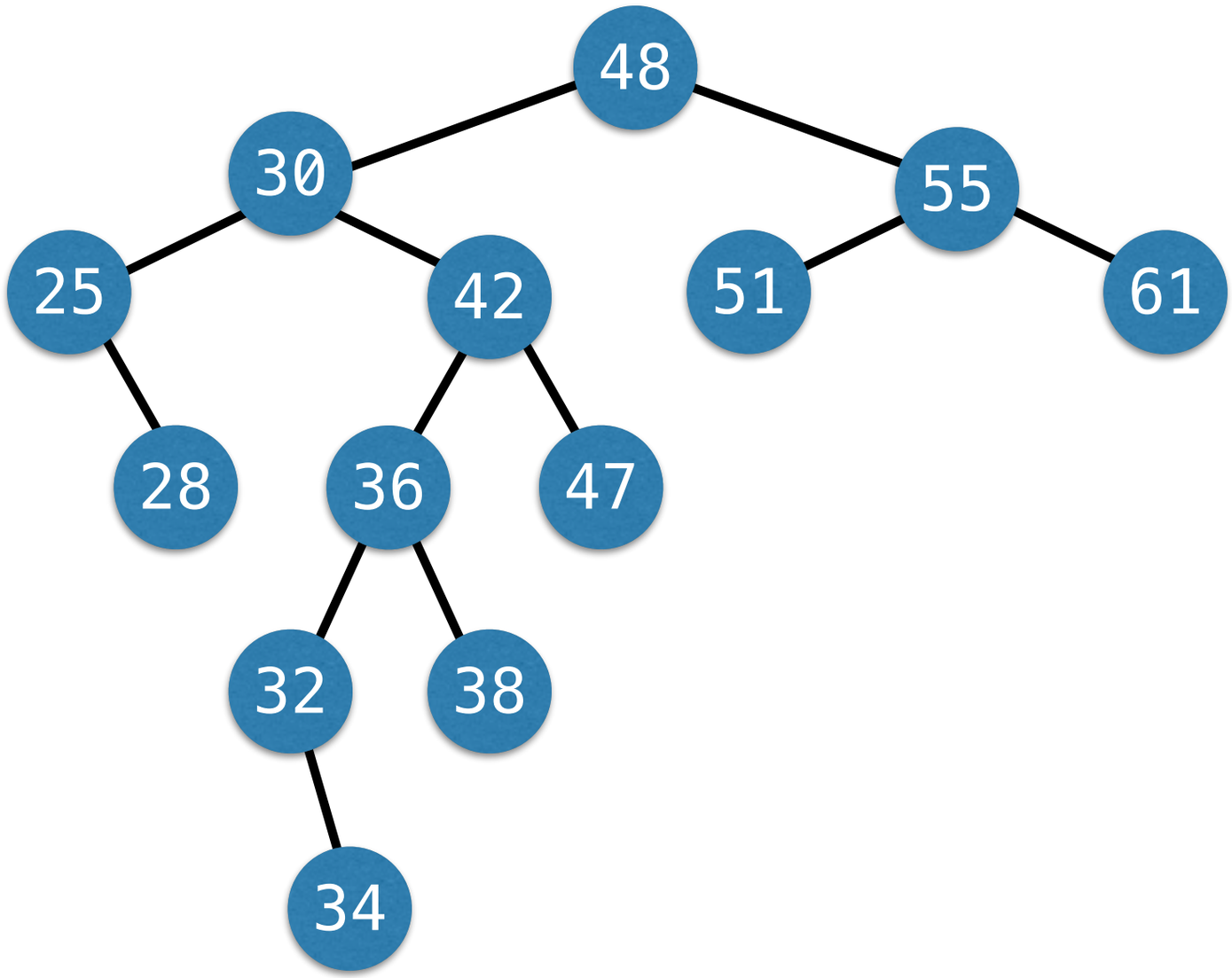

```
remove(61)
```

Deleting a Leaf

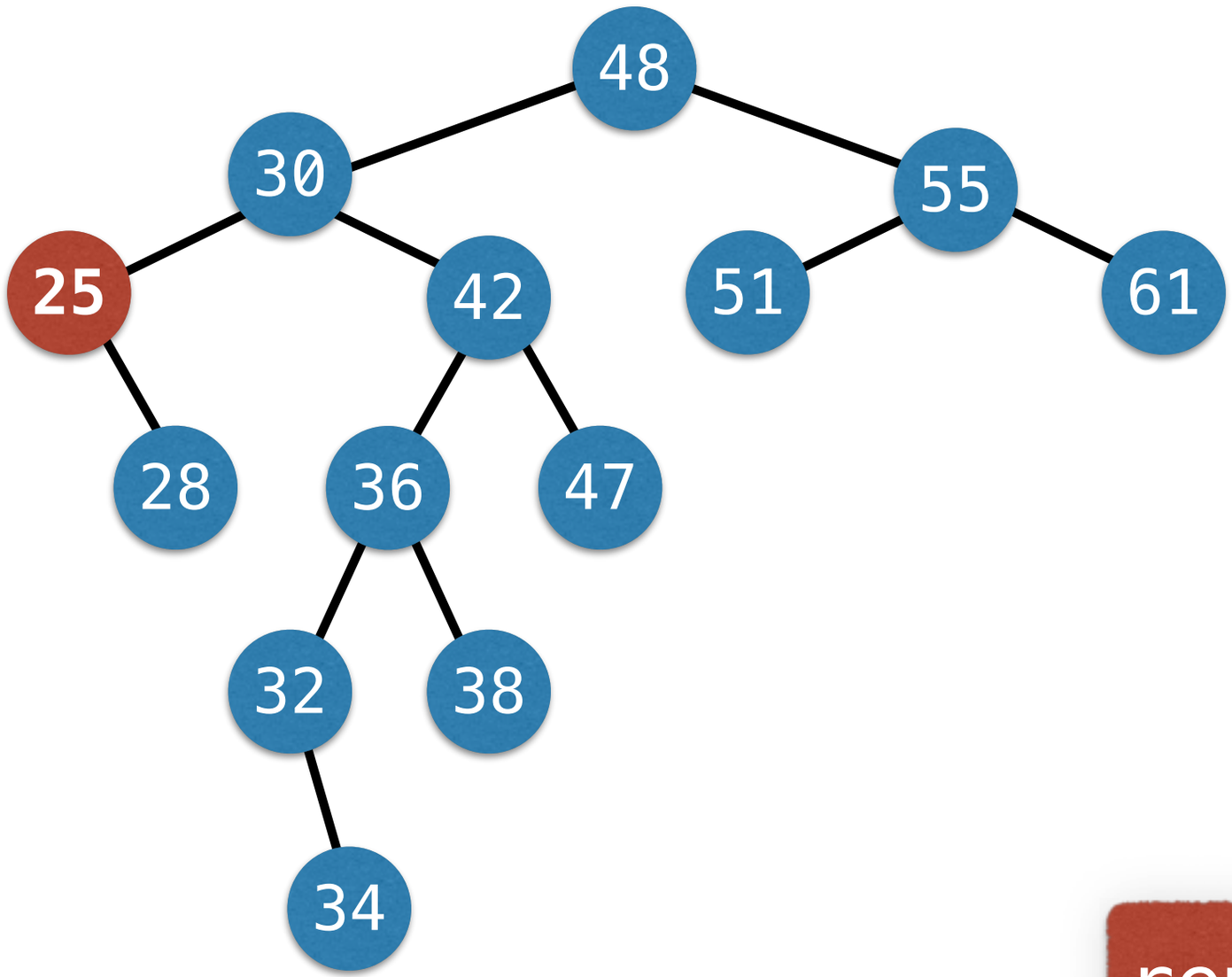



```
remove(61)
```

Deleting a Node with One Child

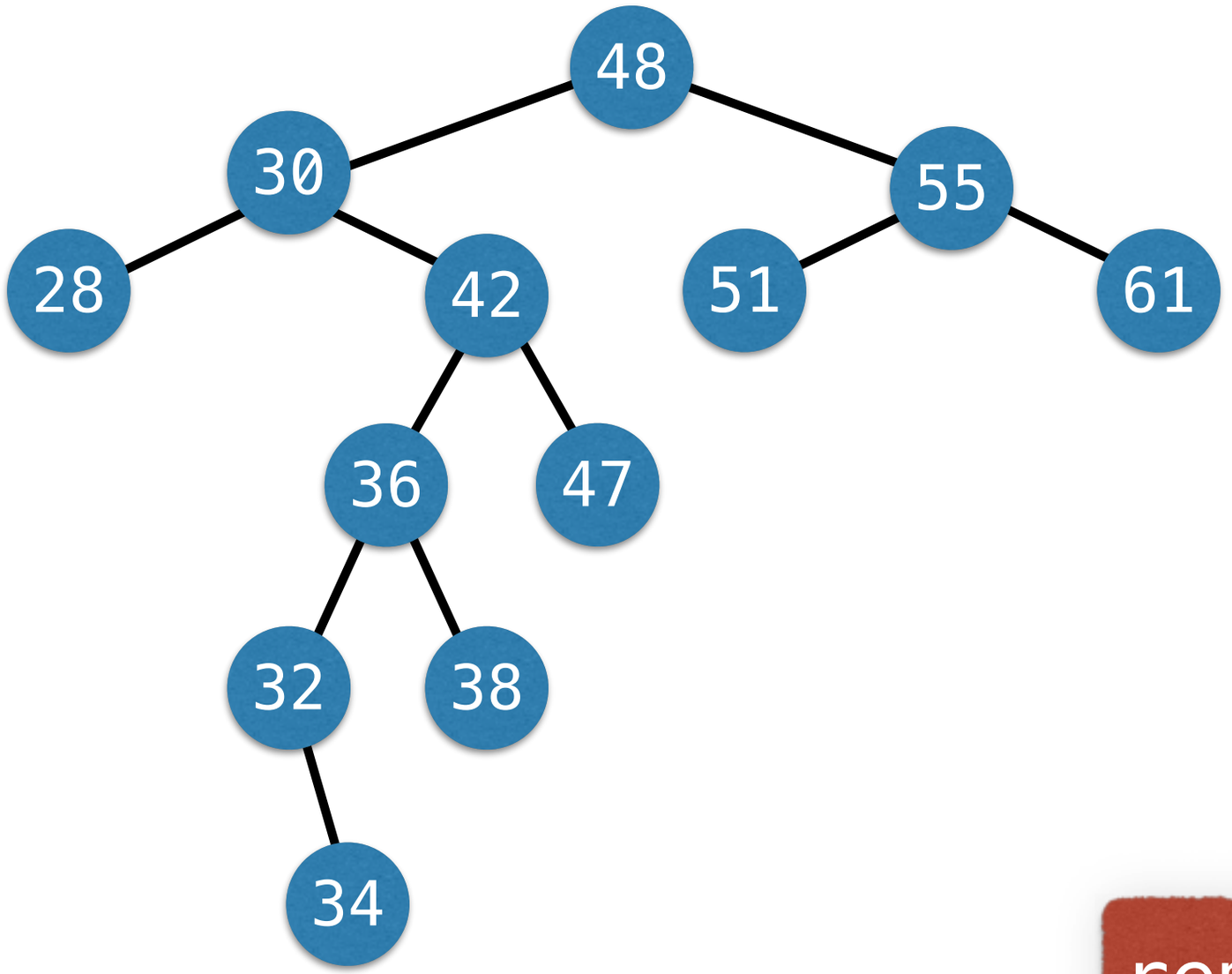


Deleting a Node with One Child



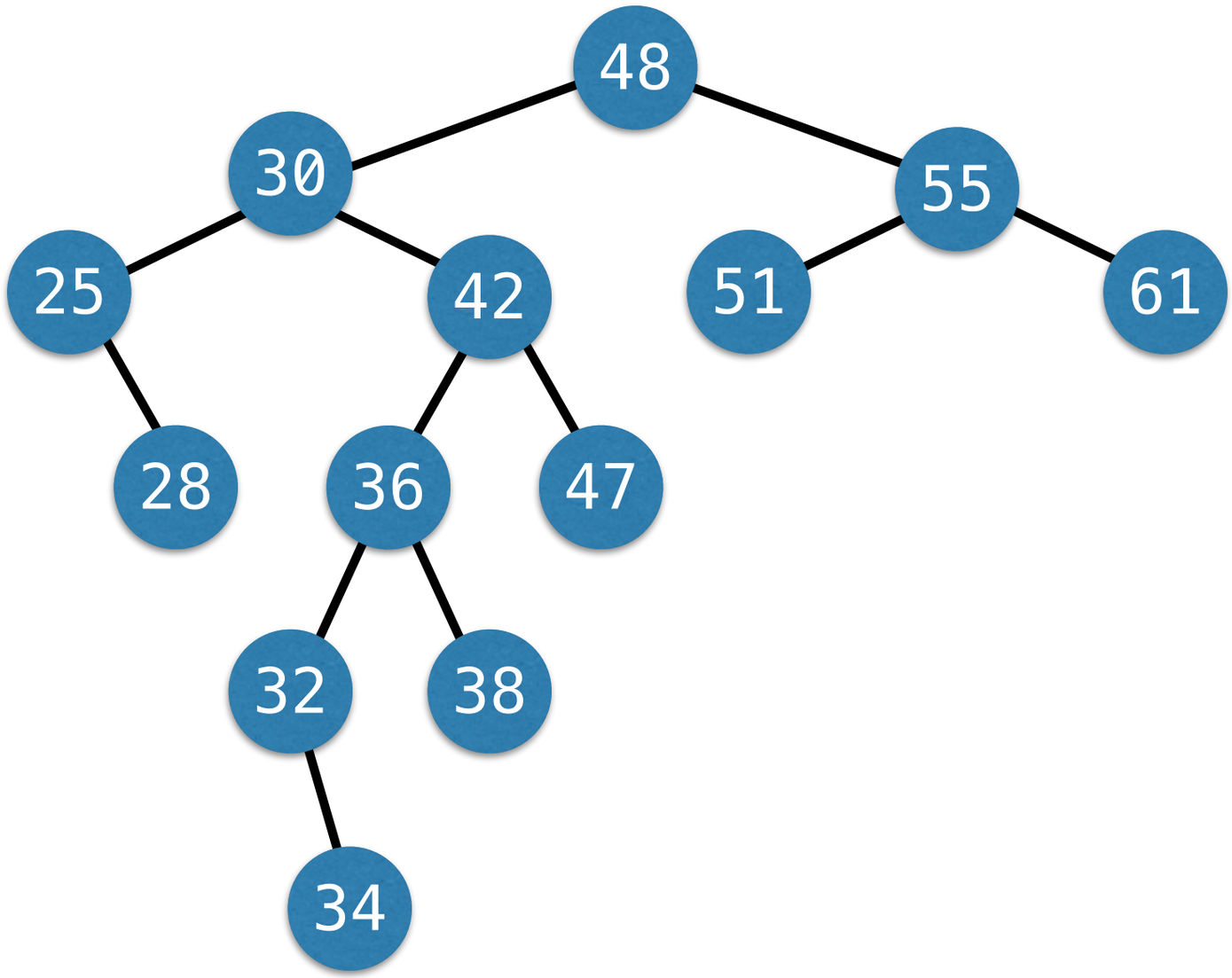

```
remove(25)
```

Deleting a Node with One Child

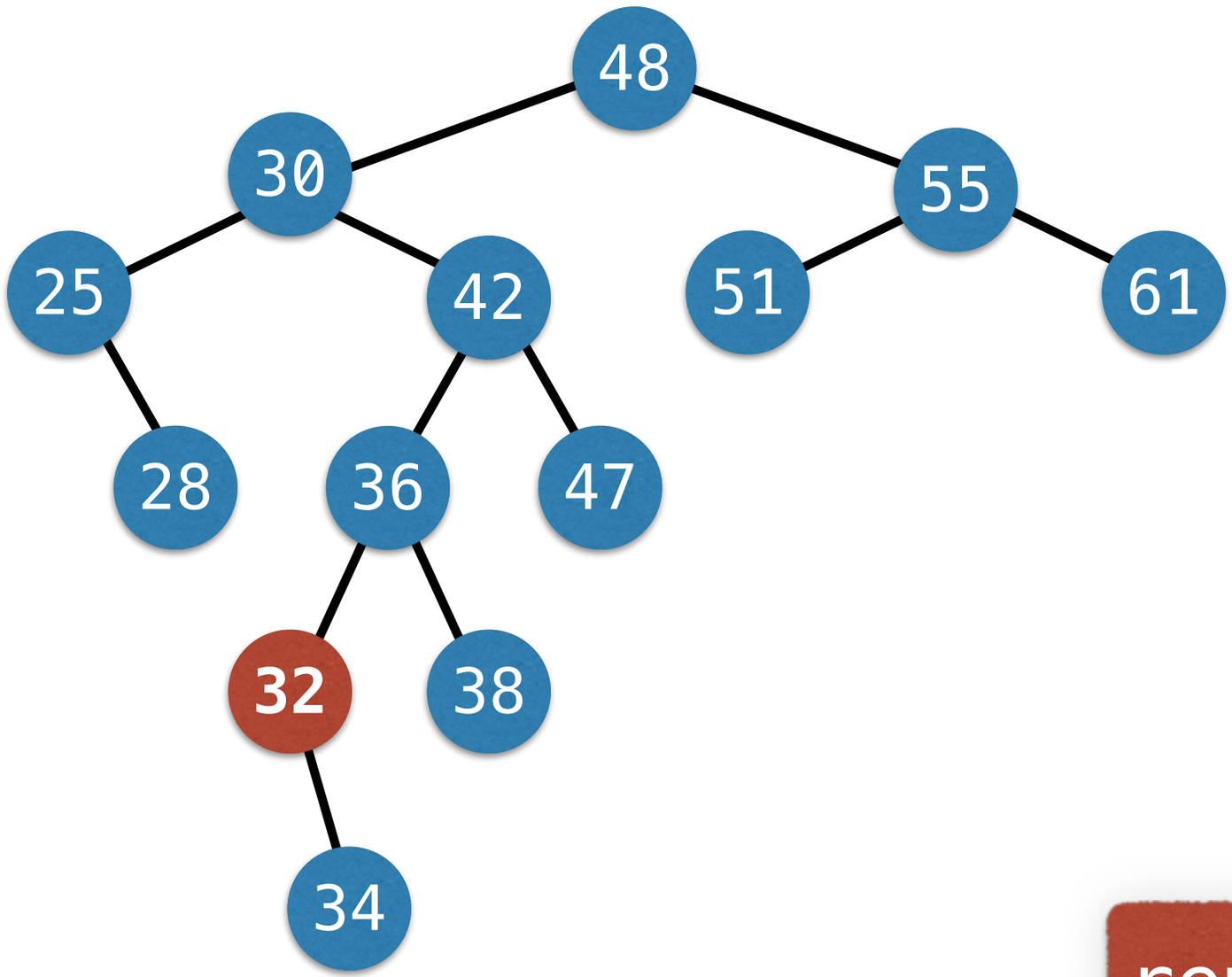



```
remove(25)
```

Deleting a Node with One Child

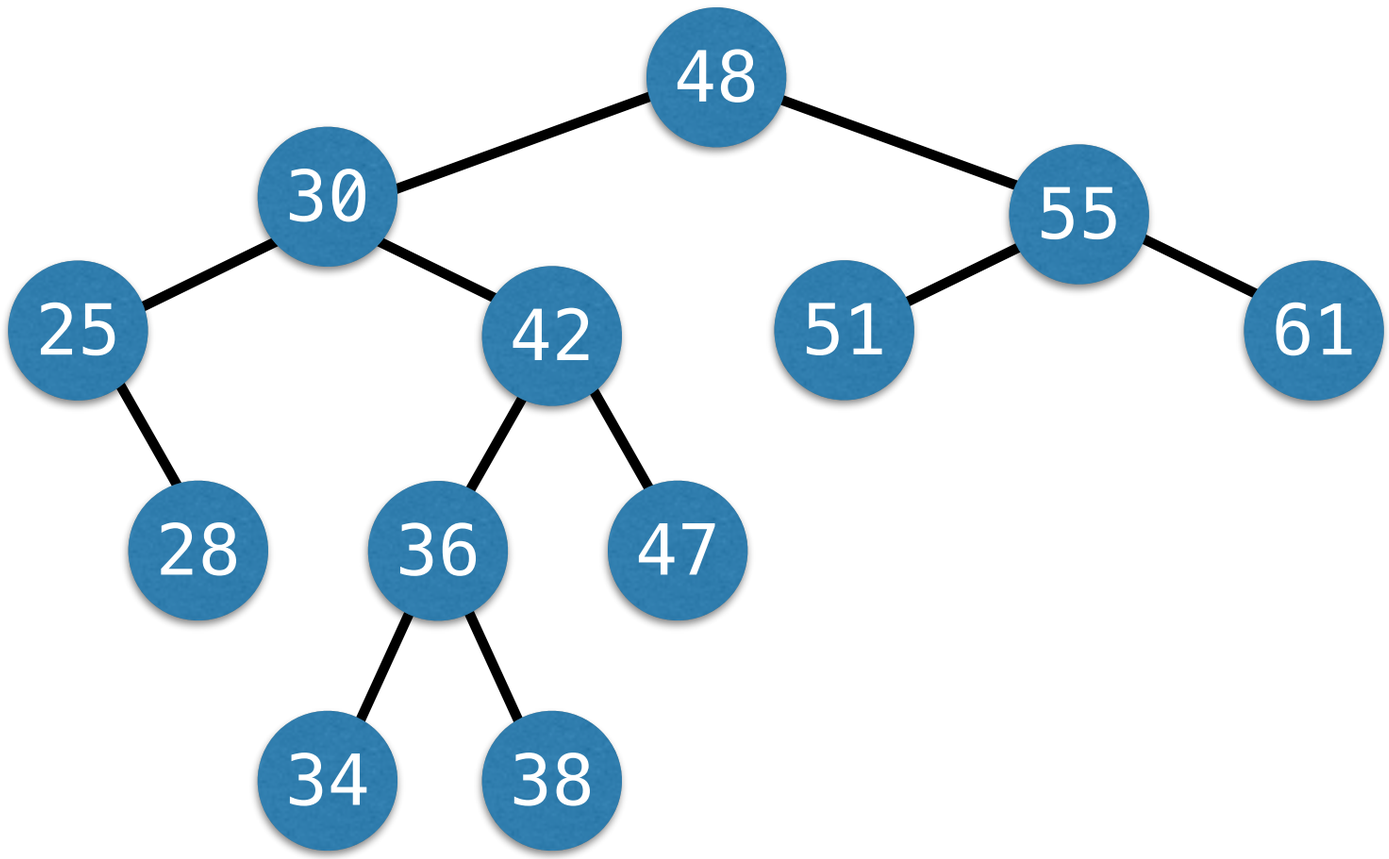


Deleting a Node with One Child



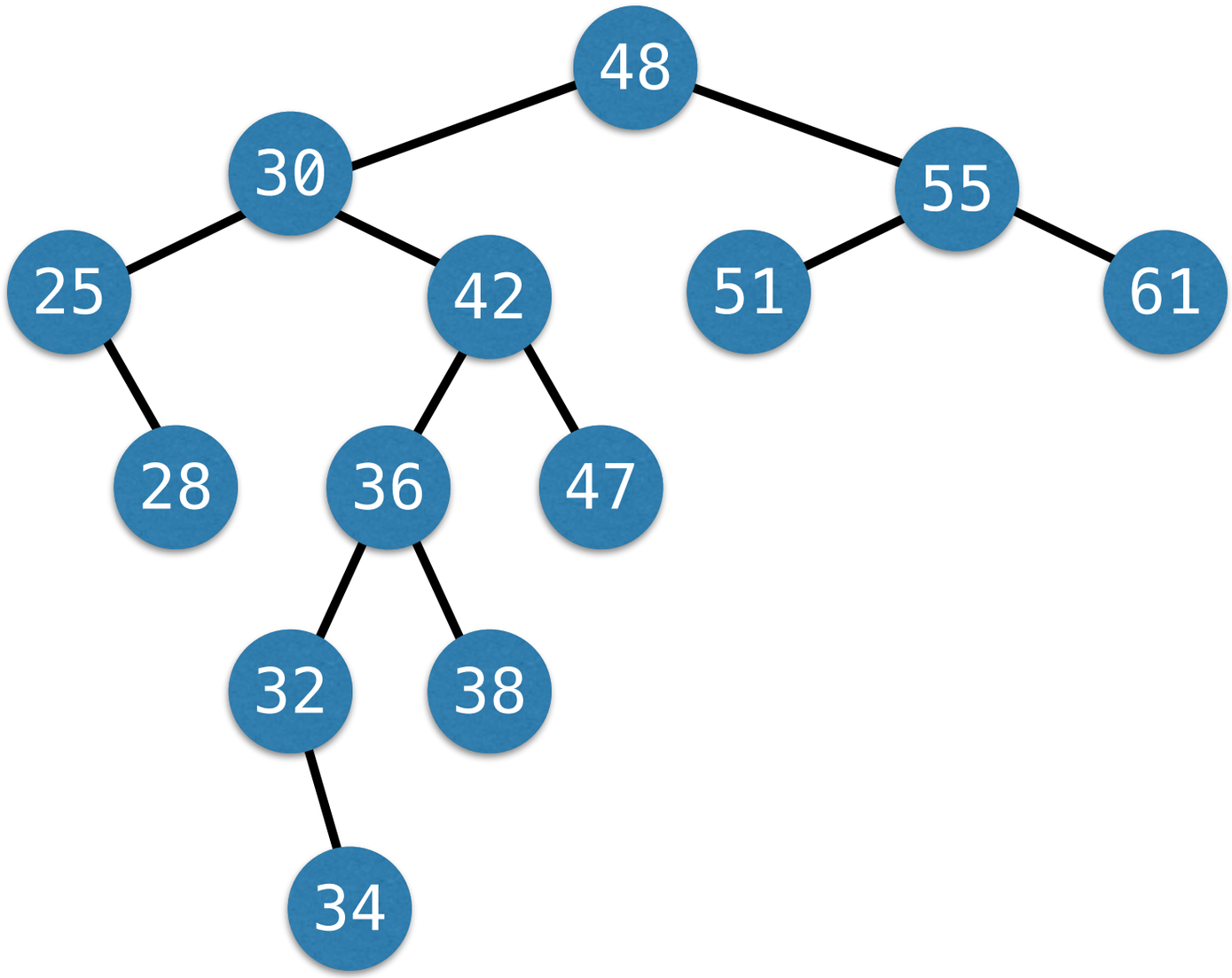

```
remove(32)
```

Deleting a Node with One Child

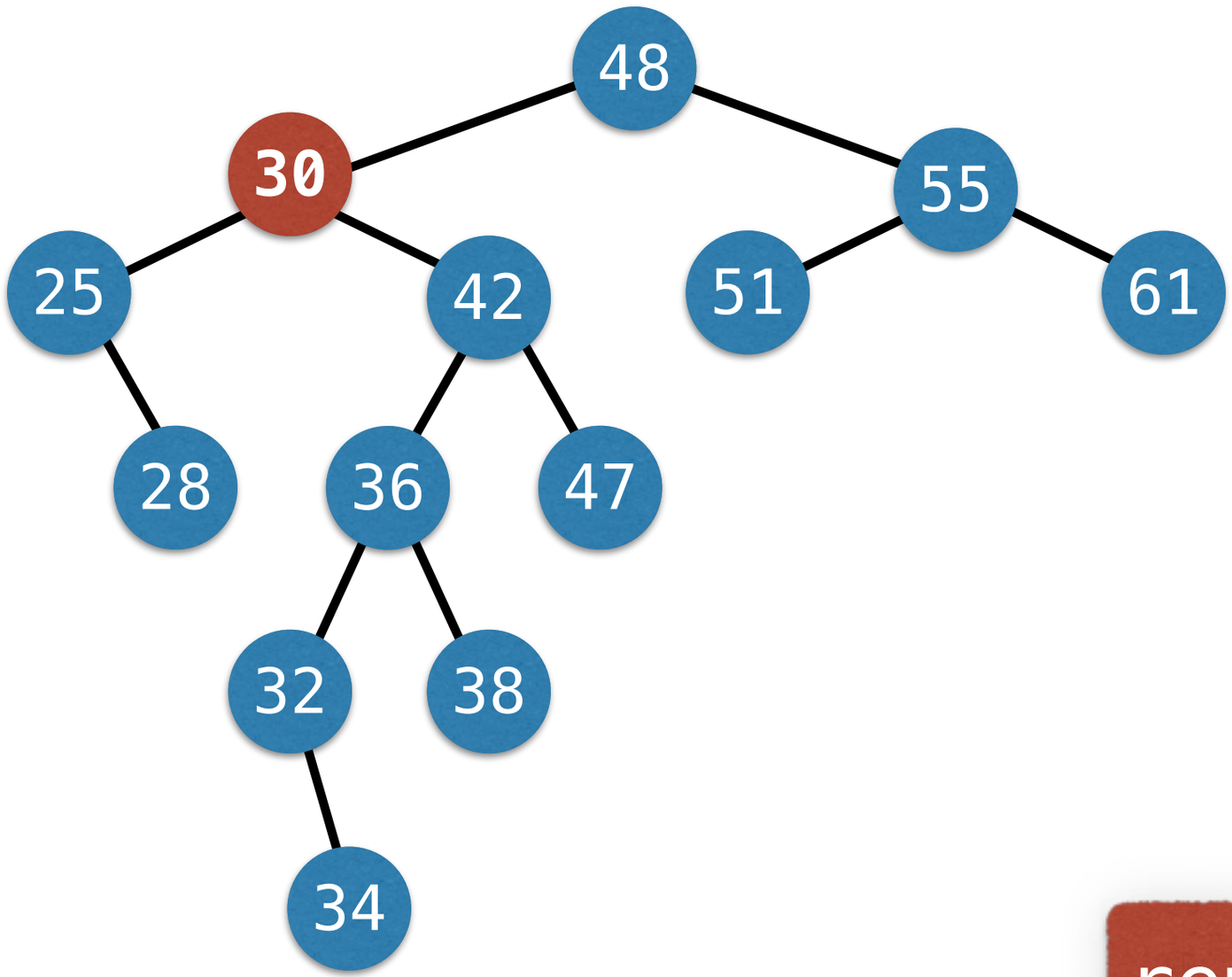



```
remove(32)
```

Deleting a Node with Two Children

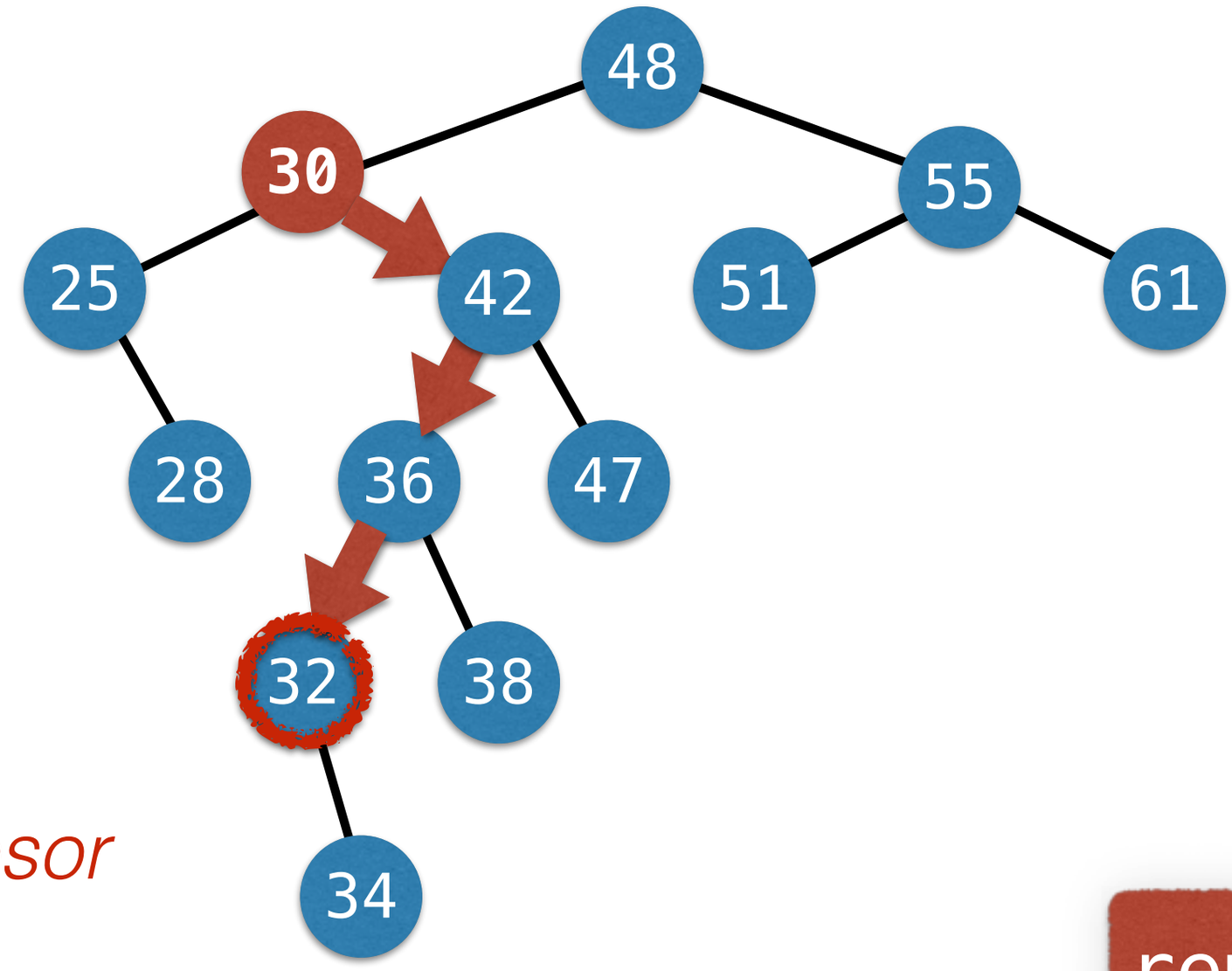


Deleting a Node with Two Children




```
remove(30)
```

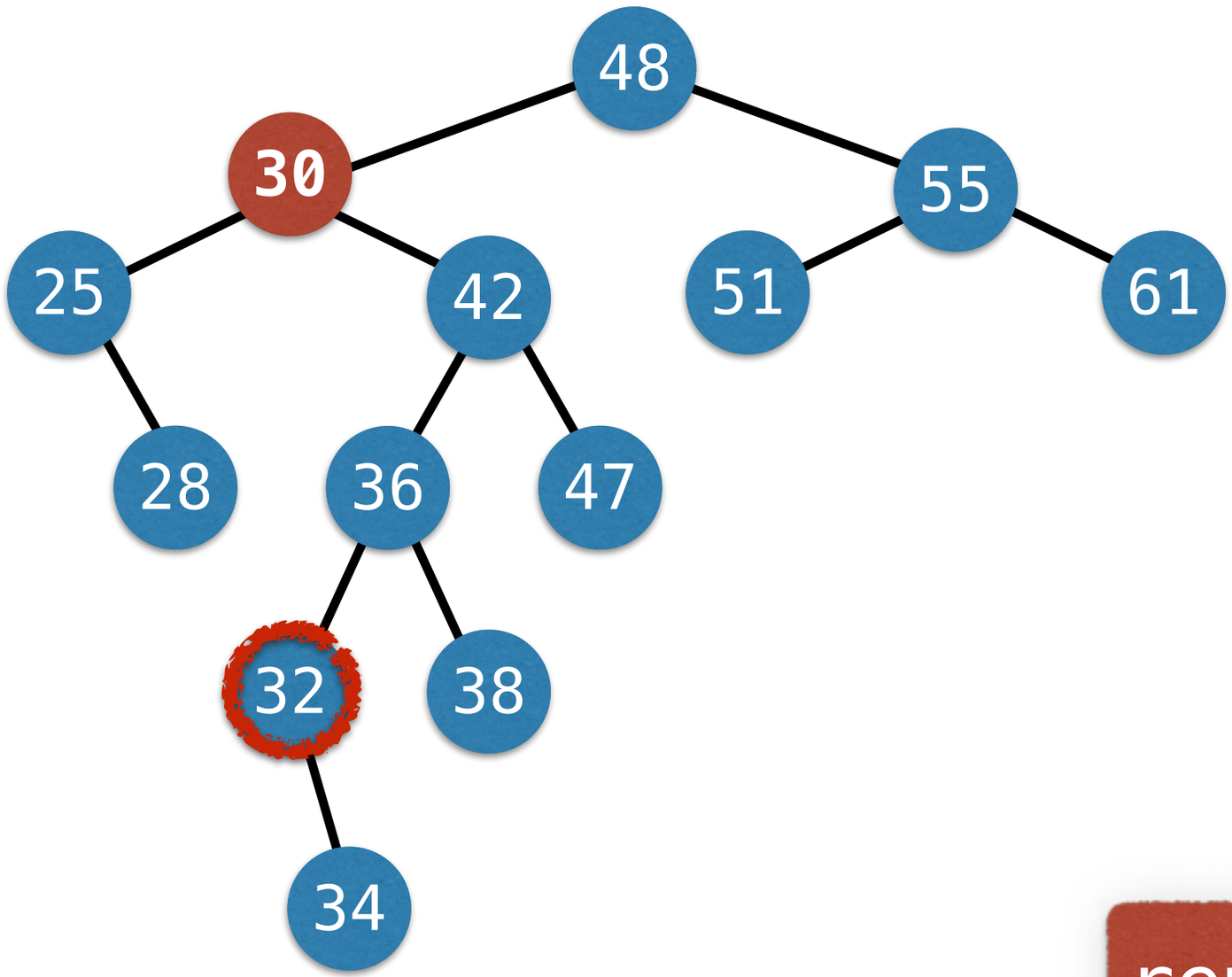
Deleting a Node with Two Children



Find succesor

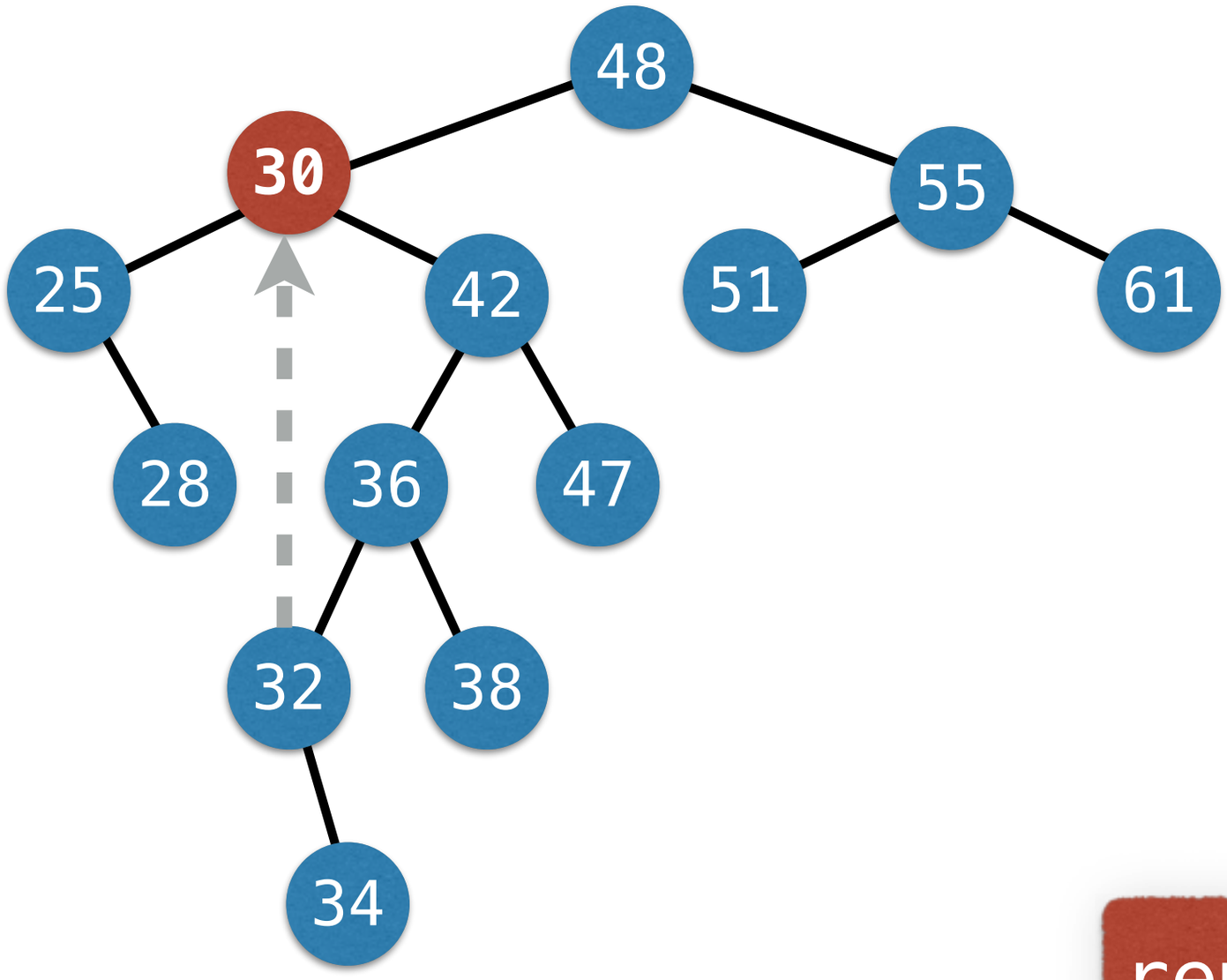
`remove(30)`

Deleting a Node with Two Children



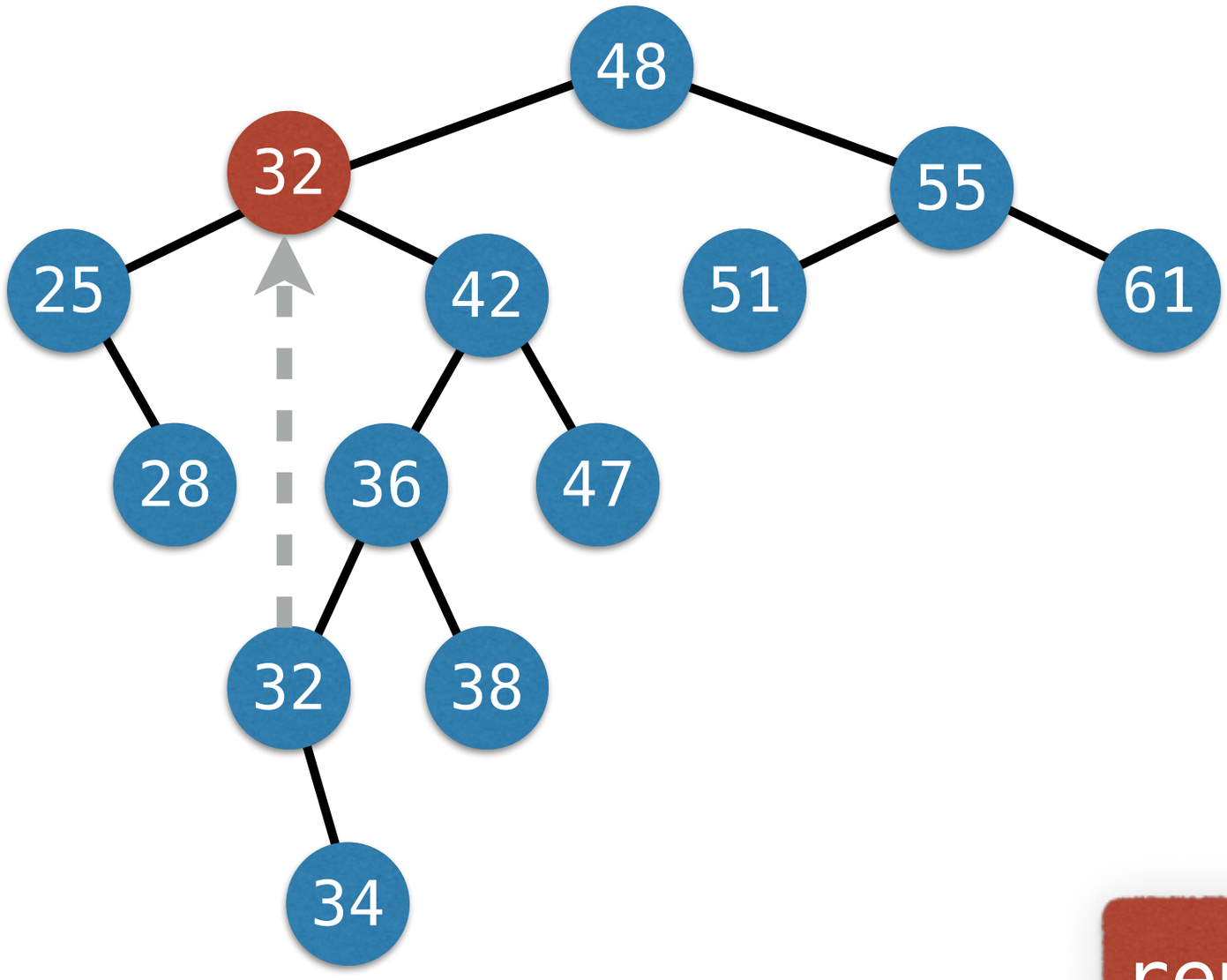
remove(30)

Deleting a Node with Two Children



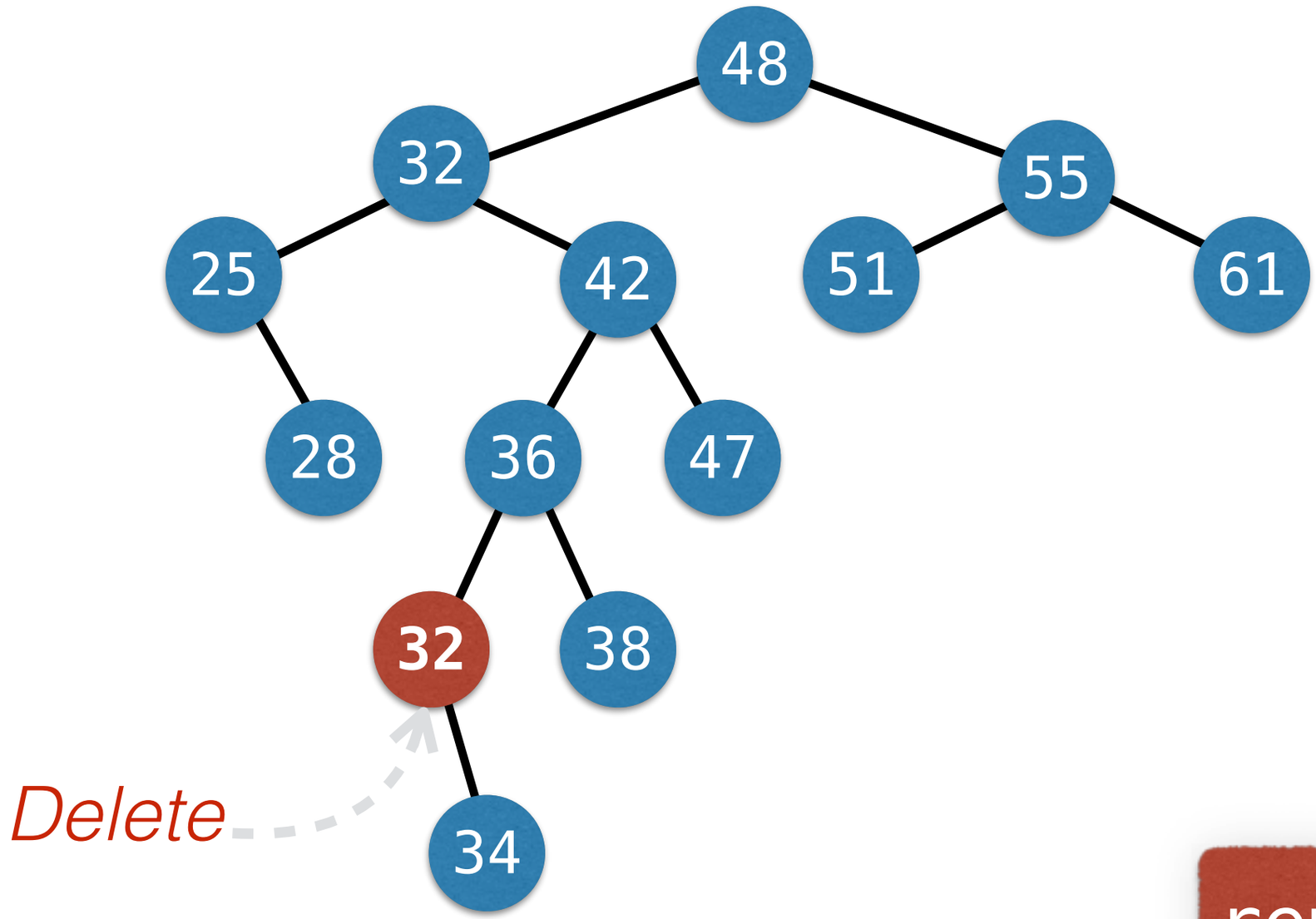

```
remove(30)
```

Deleting a Node with Two Children



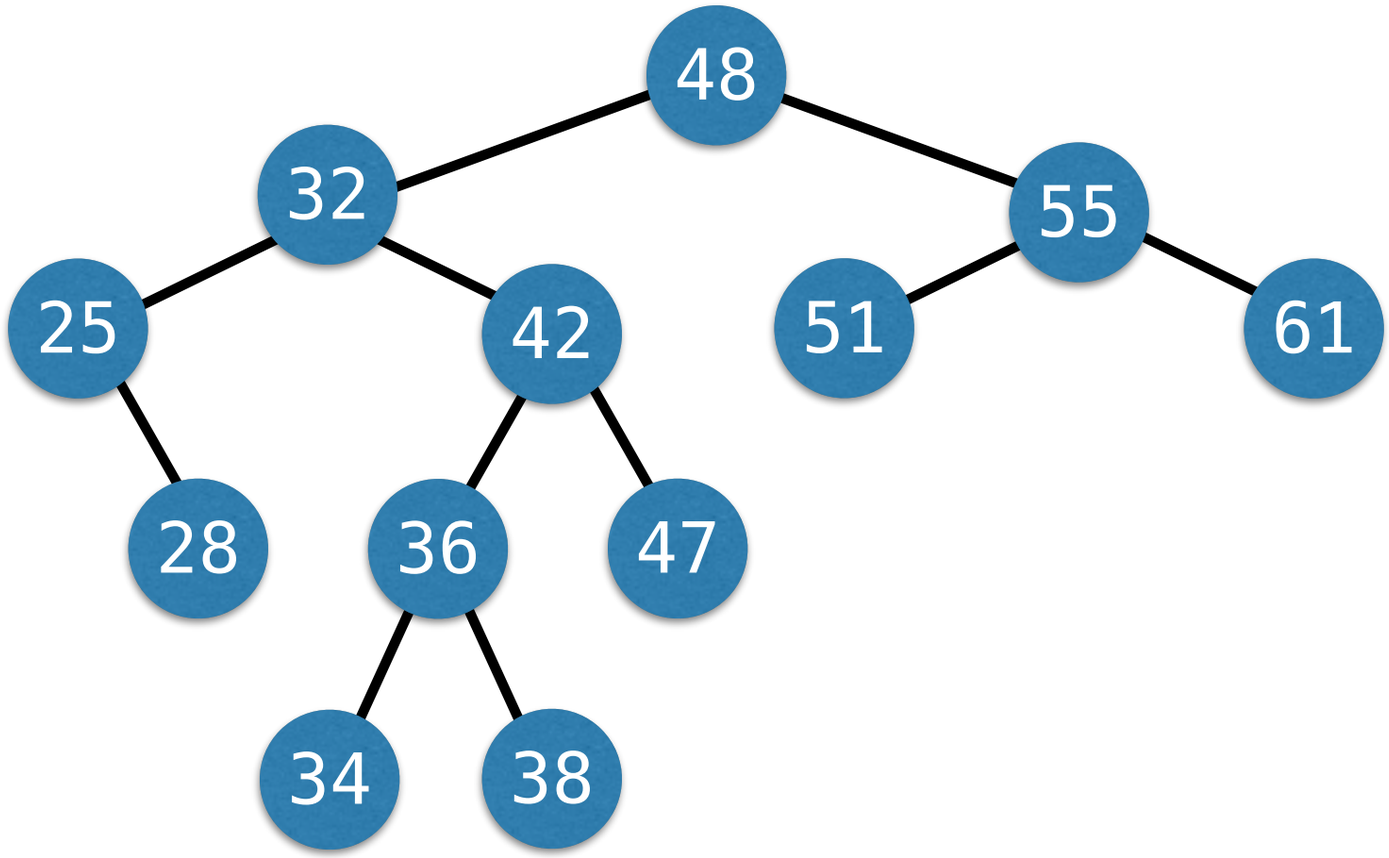

```
remove(30)
```

Deleting a Node with Two Children



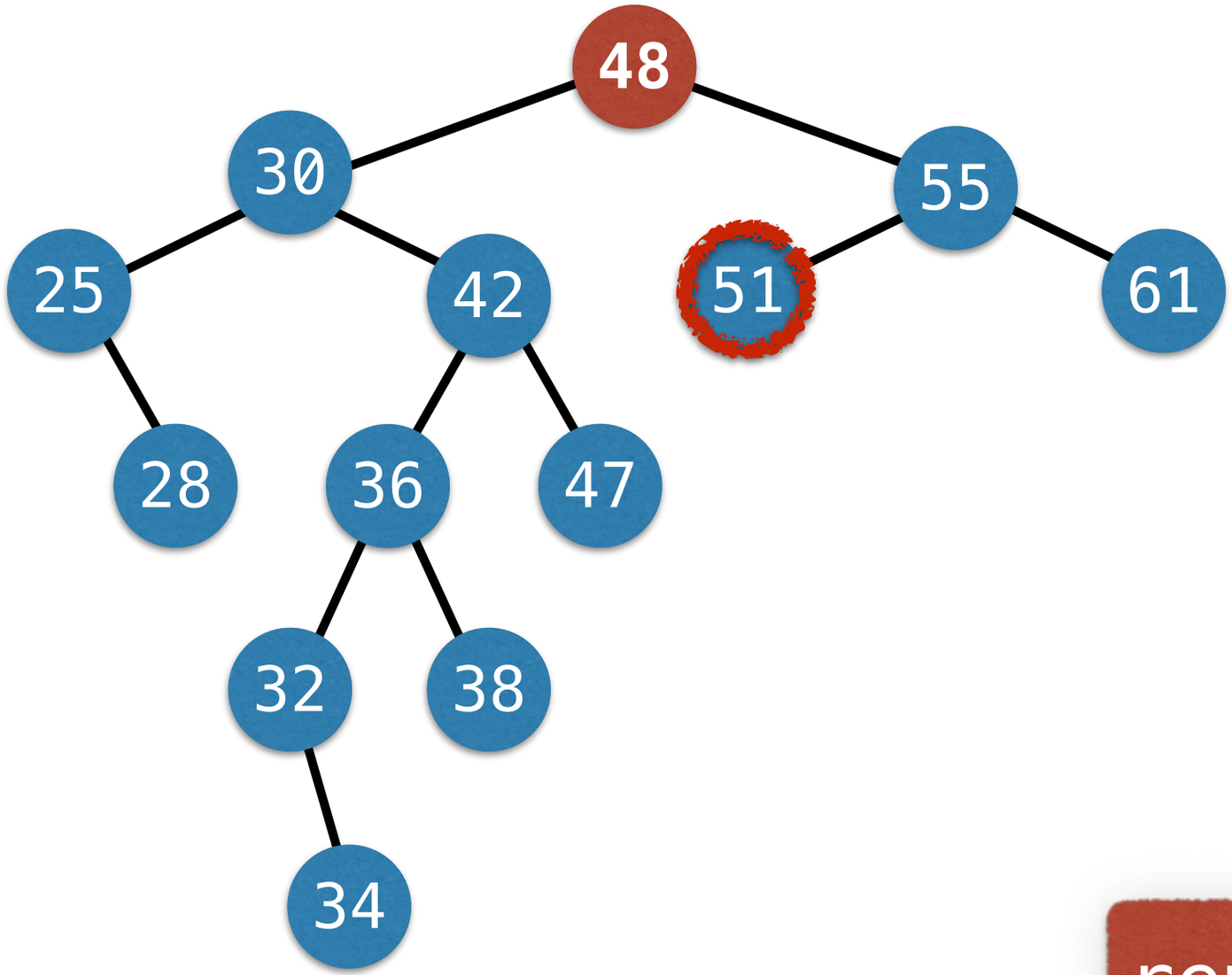

```
remove(30)
```

Deleting a Node with Two Children



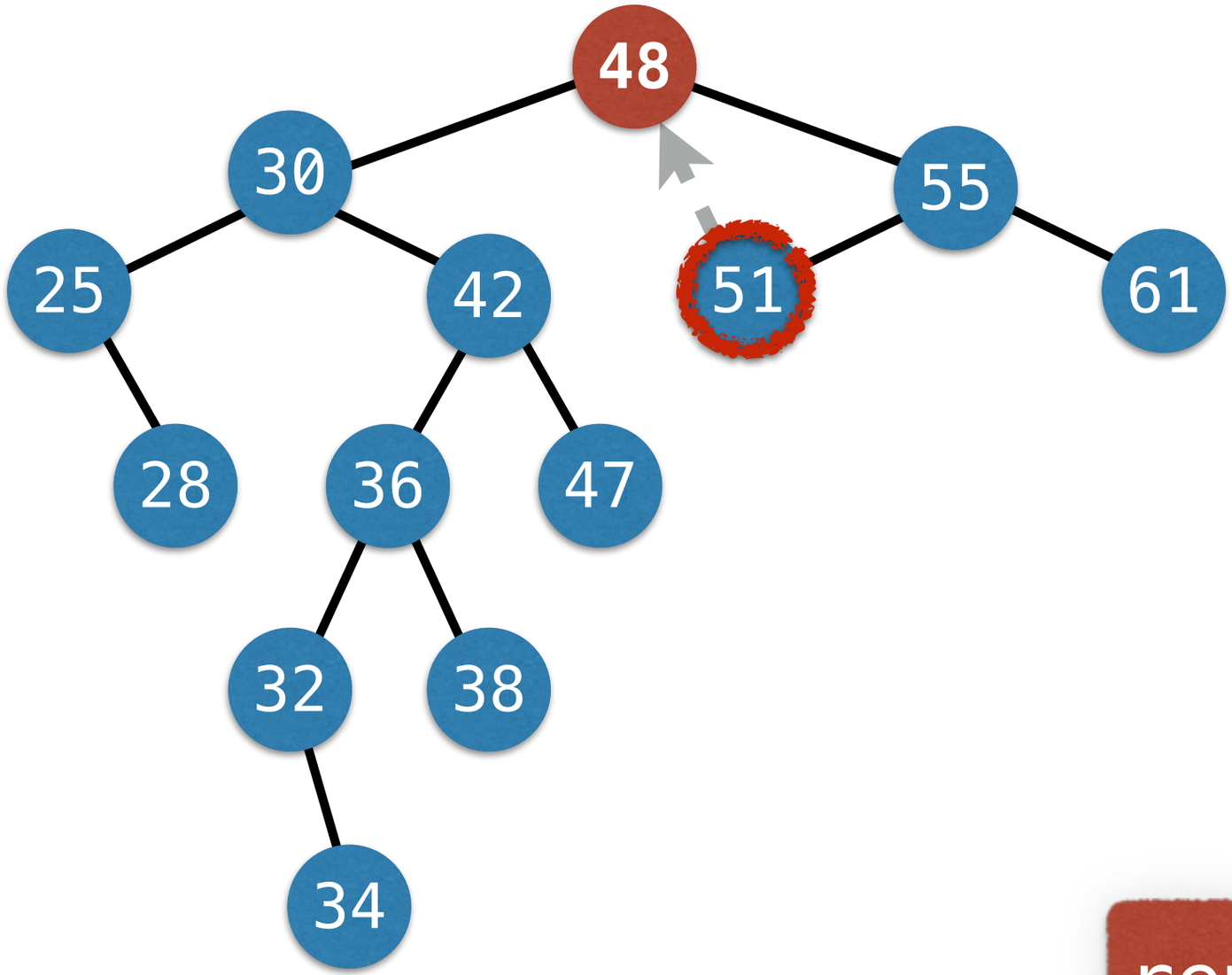

```
remove(30)
```

Deleting a Node with Two Children



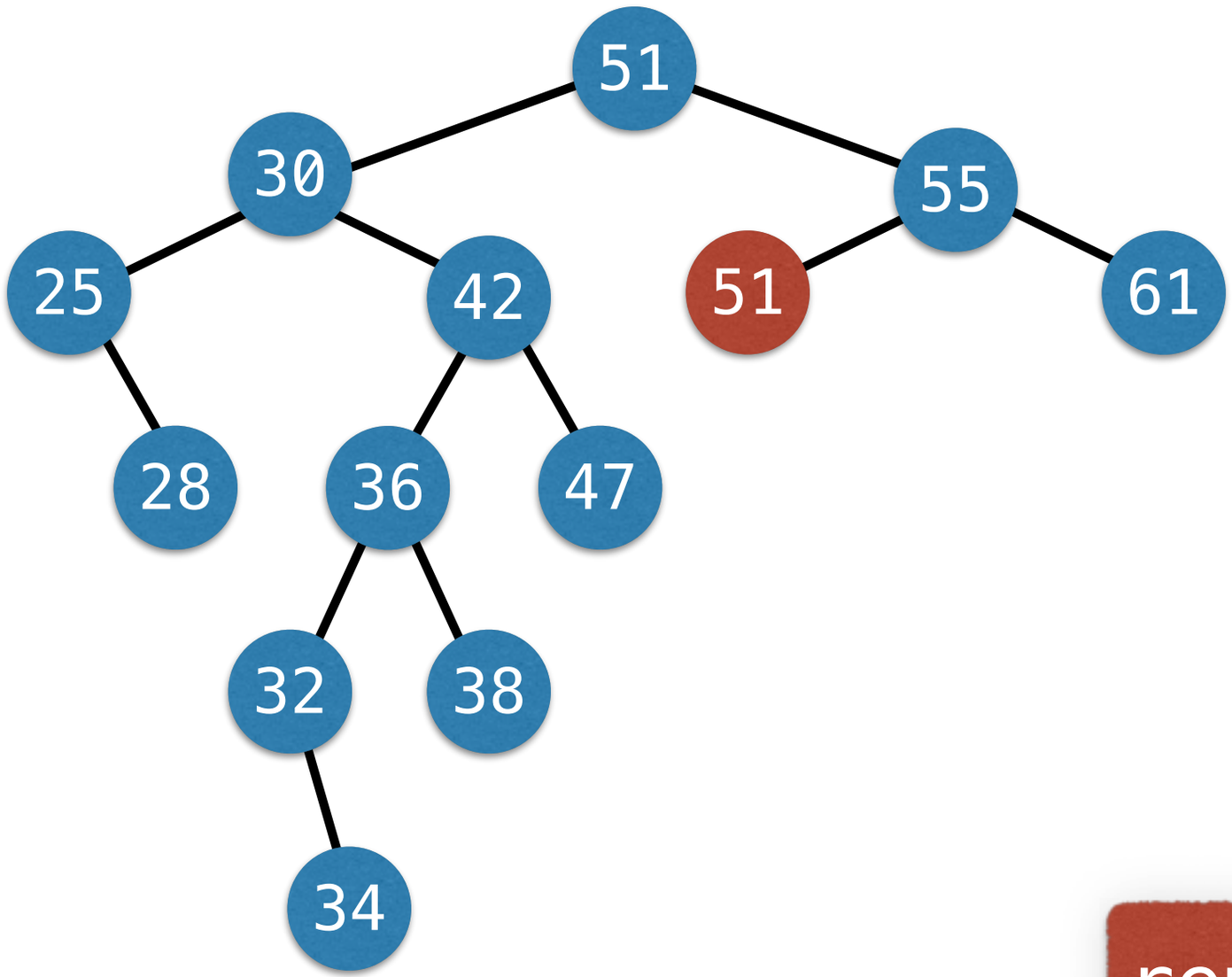

```
remove(48)
```

Deleting a Node with Two Children



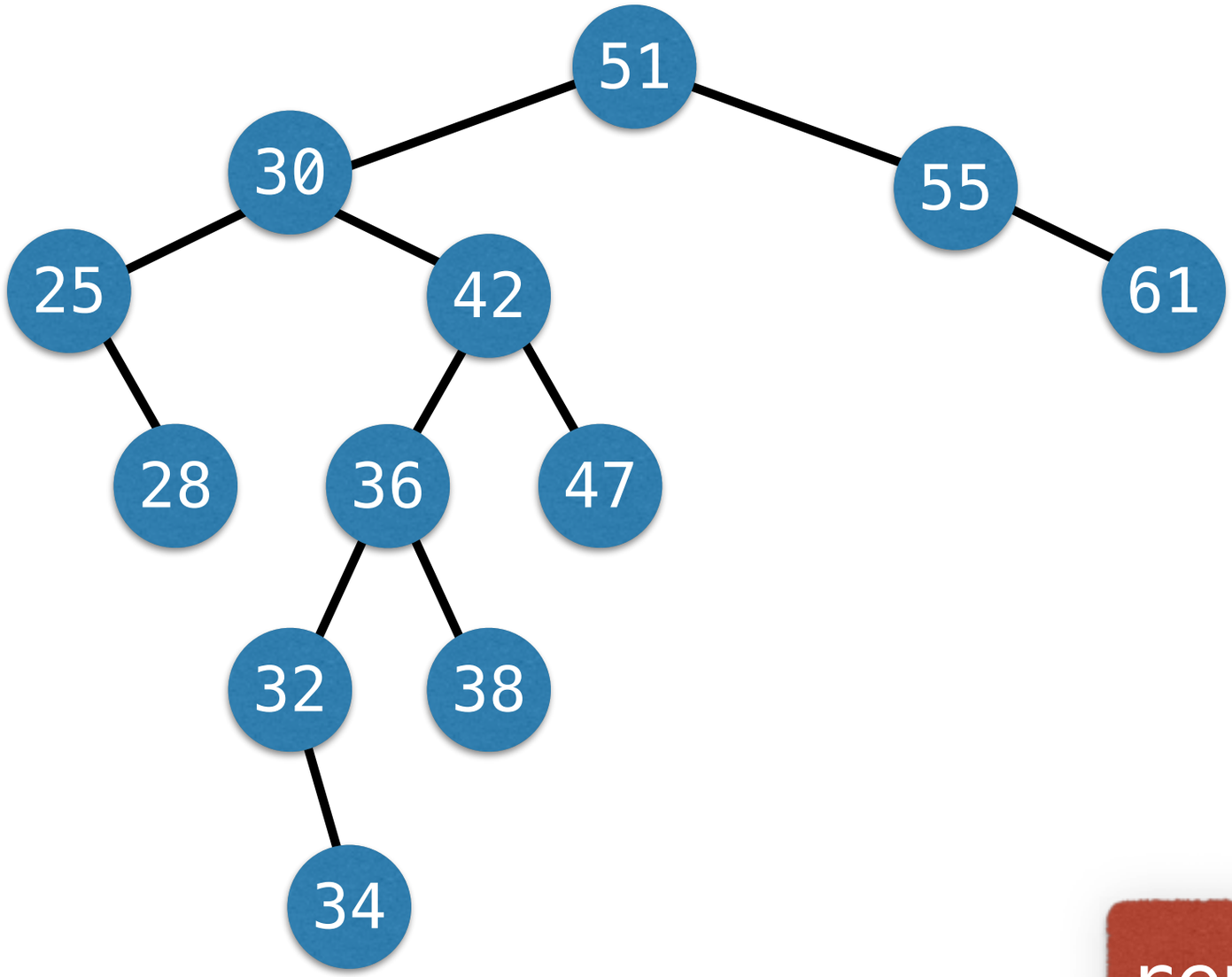

```
remove(48)
```

Deleting a Node with Two Children



remove(48)

Deleting a Node with Two Children




```
remove(48)
```

```
unlinkNode(n):

    if n has two children
        s = successor(n)
        n.data = s.data
        n = s

    replacement = null
    if n has a left child
        replacement = n.left
    else if n has a right child
        replacement = n.right
```

```
unlinkNode(n):

    if n has two children
        s = successor(n)
        n.data = s.data
        n = s

    replacement = null
    if n has a left child
        replacement = n.left
    else if n has a right child
        replacement = n.right
```

Copy the successor's data into **n** and then arrange to delete the successor.

Now **n** has at most one child. Delete **n** by replacing it with its child.

```
unlinkNode(n):

    if n has two children
        s = successor(n)
        n.data = s.data
        n = s

    replacement = null
    if n has a left child
        replacement = n.left
    else if n has a right child
        replacement = n.right
```

First, determine the replacement node.

```
if n is the root
    root = replacement
else
    if n is a left child of its parent
        n.parent.left = replacement
    else
        n.parent.right = replacement

if replacement != null
    replacement.parent = n.parent

--size
```

Now, link the replacement node to n's parent.

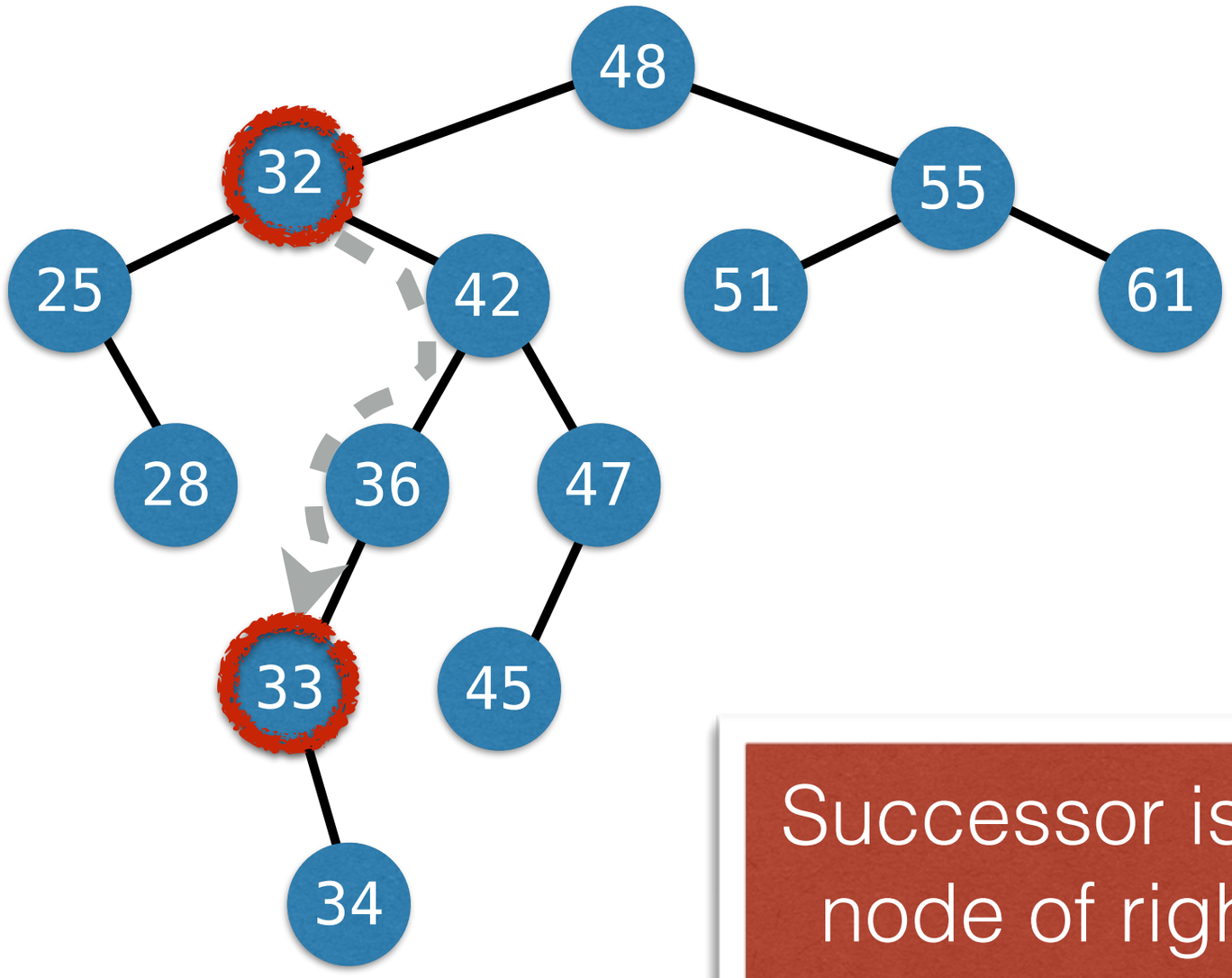
```
if n is the root
    root = replacement
else
    if n is a left child of its parent
        n.parent.left = replacement
    else
        n.parent.right = replacement

if replacement != null
    replacement.parent = n.parent

--size
```

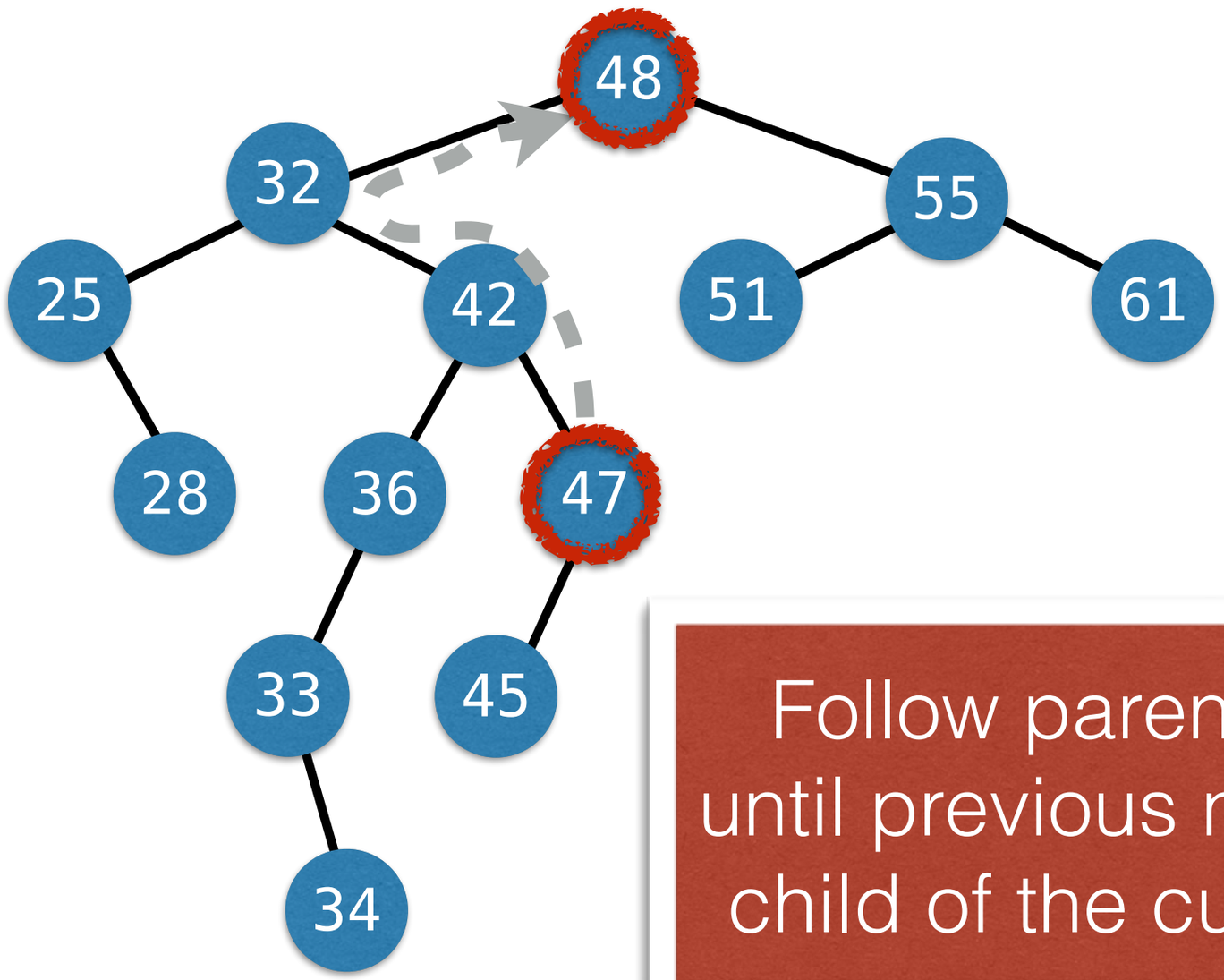
successor()

Case 1: The node has a right subtree.



Successor is in leftmost node of right subtree.

Case 2: The node has no right subtree.



Follow parent pointers, until previous node is a left child of the current node.