Trees

Johan Montelius

KTH

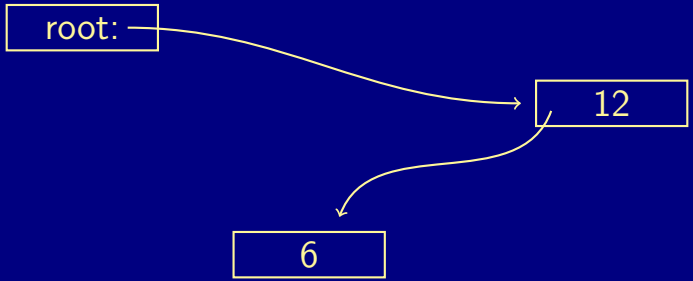
HT22

a tree

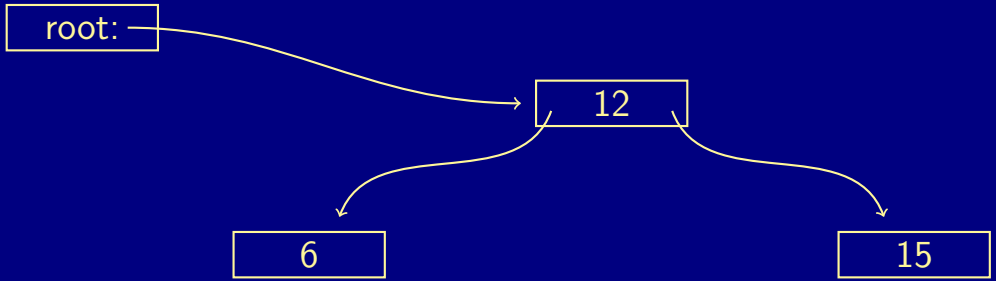
root:

a tree

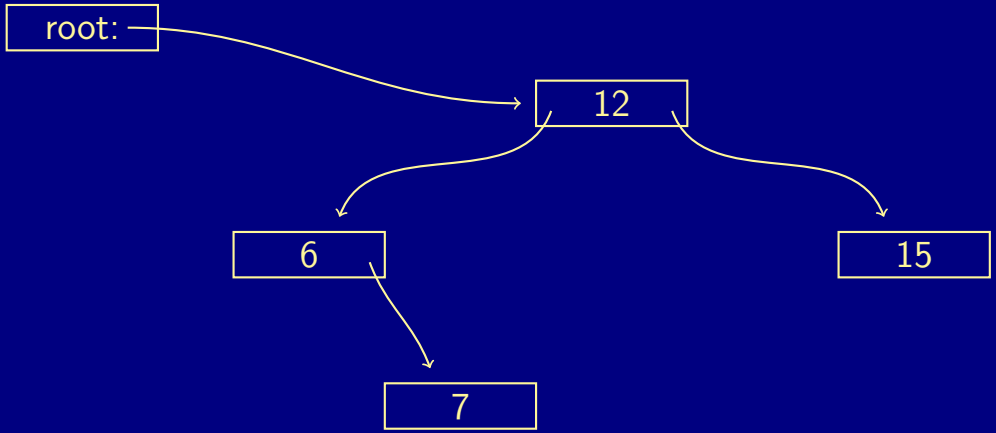
a tree



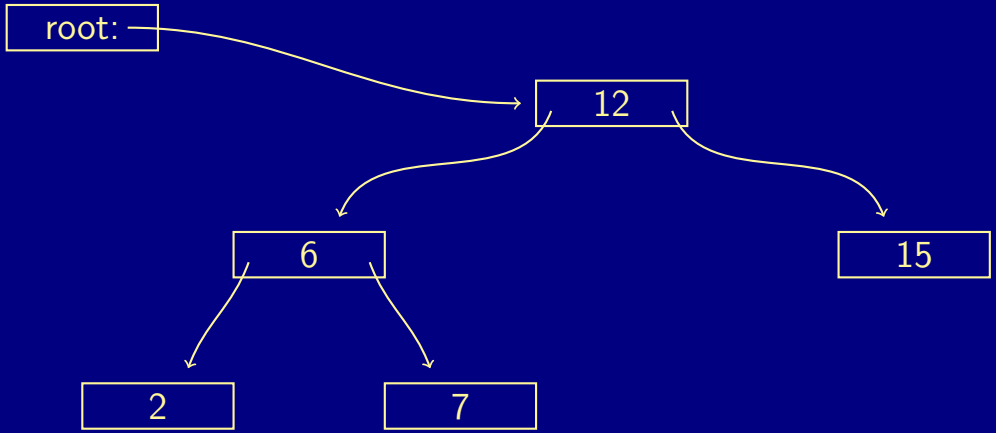
a tree



a tree



a tree



the node

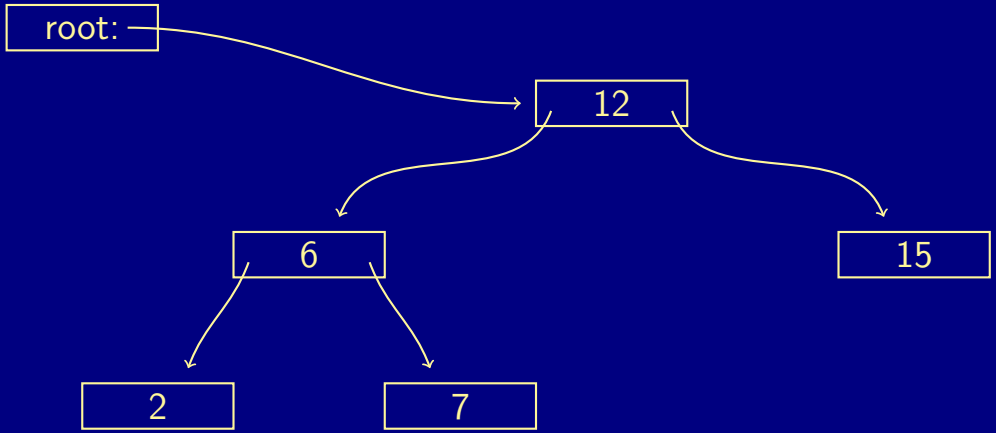
```
public class BinaryTree {
  public Node root;
  :
  :


}
```

the node

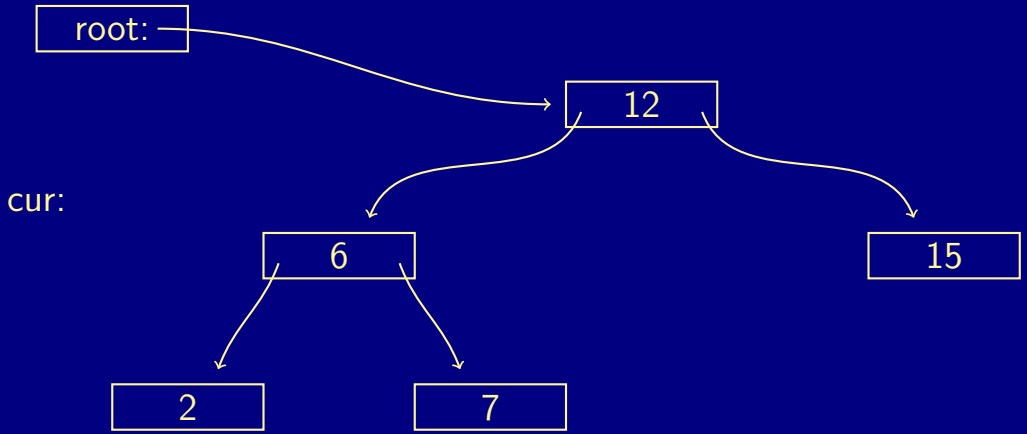
```
public class BinaryTree {
  public Node root;
  :
  :


}
```

```
private Node {
  Integer key;
  Integer value;
  Node left;
  Node right;
}
```

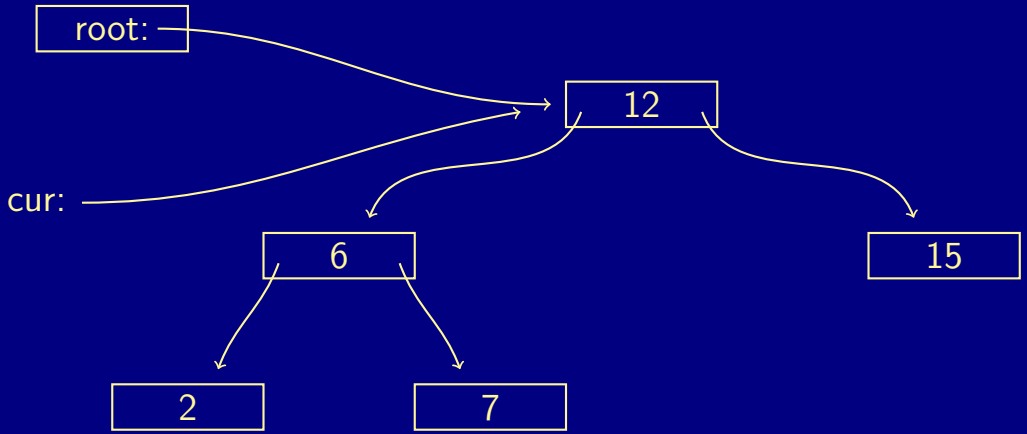
add a new key-value pair



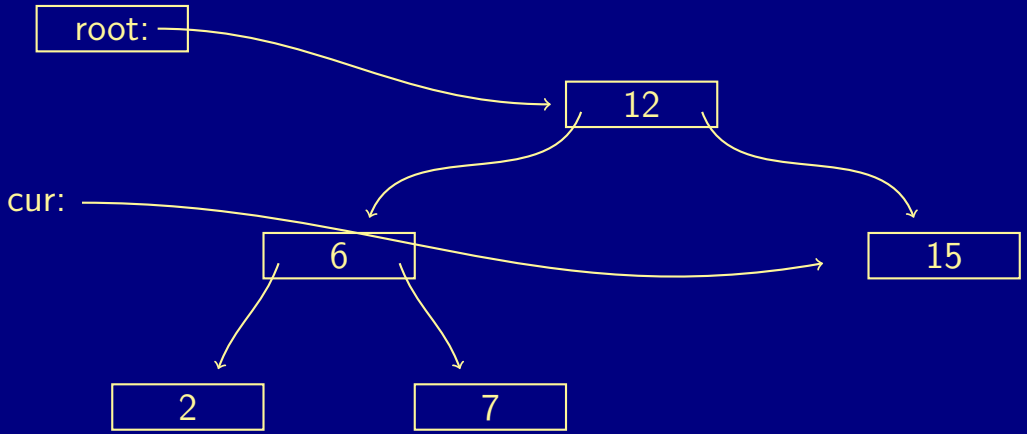
add a new key-value pair



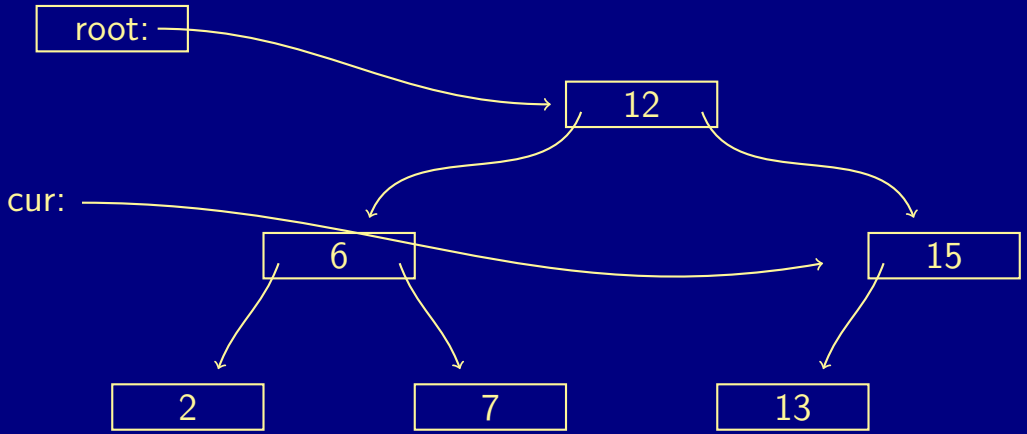
add a new key-value pair



add a new key-value pair



add a new key-value pair



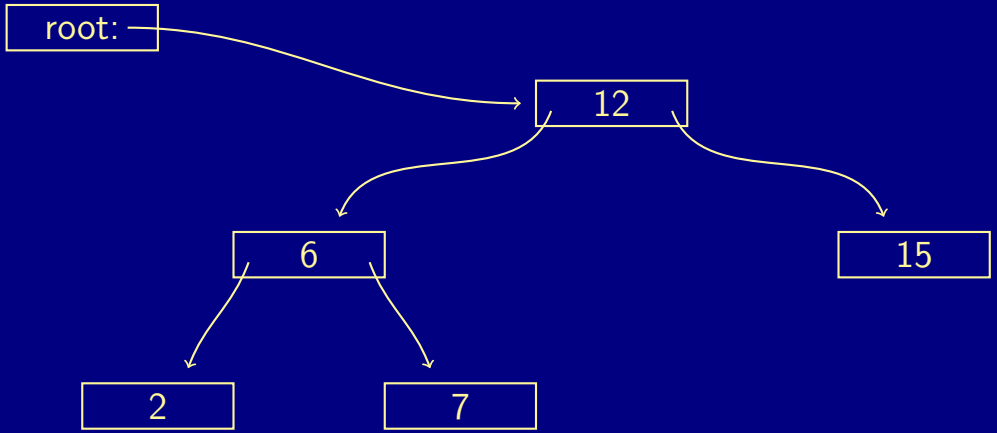
BinaryTree add(Integer key, Integer value)

```
public void add(Integer key, Integer value) {
    if (root == null)
        root = new Node(key, value);
    else
        root.add(key, value);
}
```

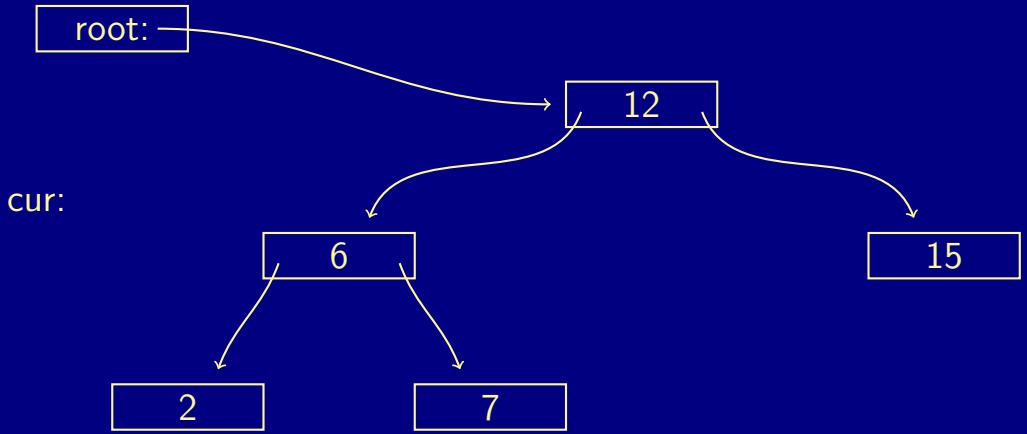
Node add(Integer key, Integer value)

```
private void add(Integer key, Integer value) {

    if (this.key  == key) {
        this.value = value;
        return;
    }
    if (this.key > key)
        if (this.left != null)
            this.left.add(key, value);
        else
            this.left = new Node(key,value);
    else
        :
 }
```

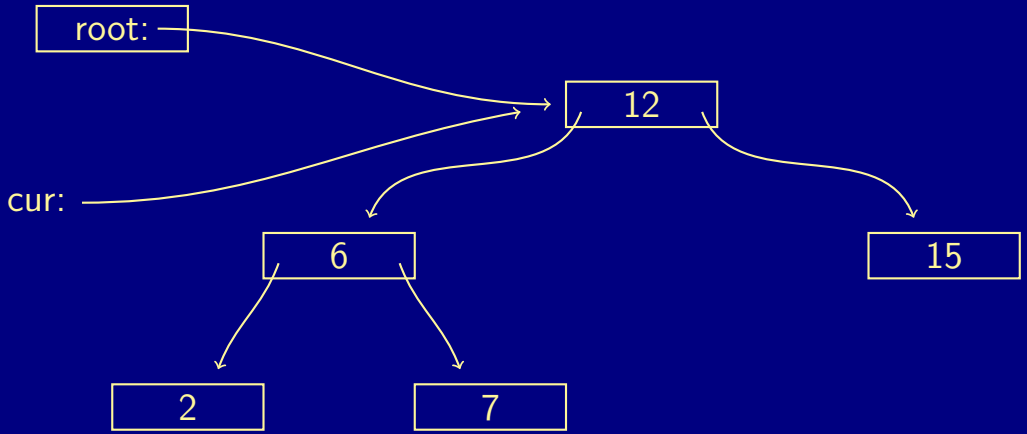
lookup - find the value of a key



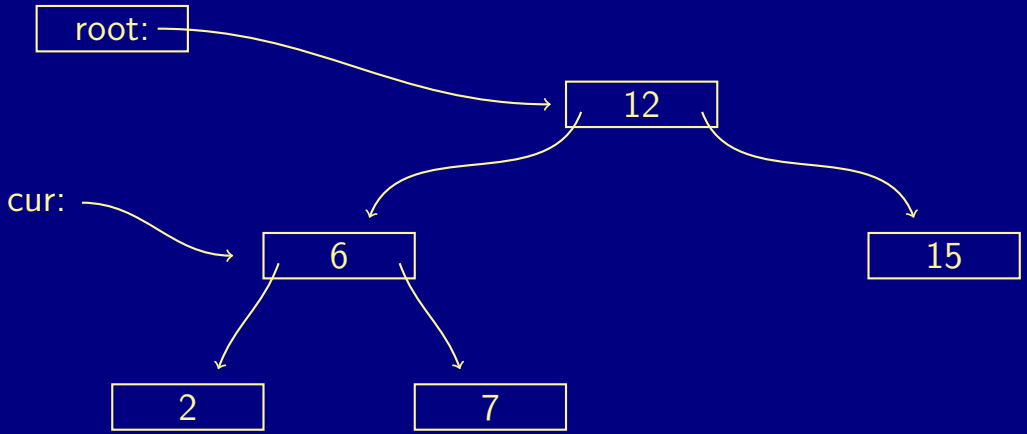
lookup - find the value of a key



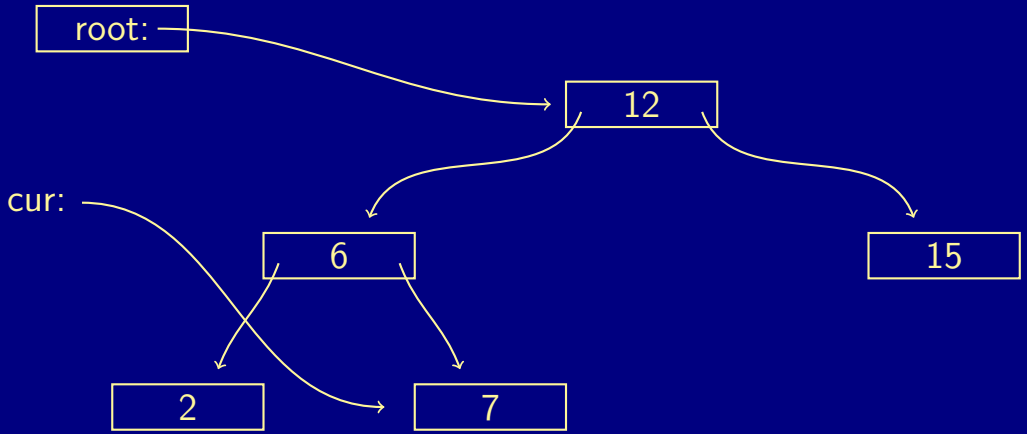
lookup - find the value of a key



lookup - find the value of a key



lookup - find the value of a key



lookup(Integer key)

```
public class BinaryTree {
   :
  public int lookup(Integer key) {
    Node cur = this.root;
    while (cur != null) {
      if (cur.key == key)
        return cur.value;
      if (cur.key < key)
        cur = cur.right;
      else
        cur = cur.left;
    }
    return null;
  }
   :
}
```

asymptotic complexity

- What is the run time complexity of lookup()?

asymptotic complexity

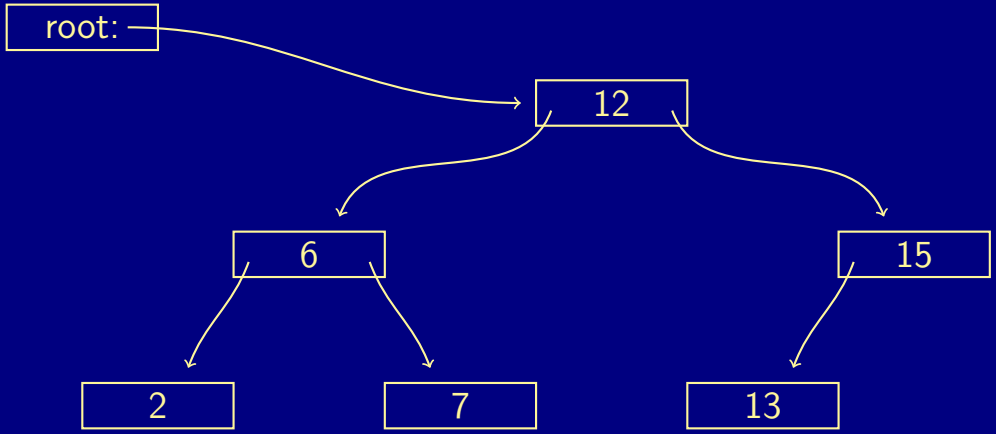
- What is the run time complexity of lookup()?
- Depending on what?

asymptotic complexity

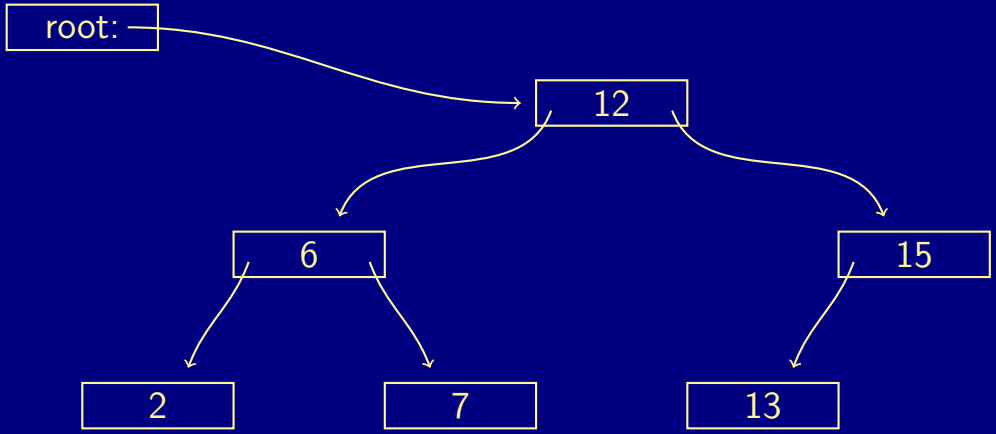
- What is the run time complexity of lookup()?
- Depending on what?

Not all trees are balanced :-(

delete 2, 7 or 13



delete 2, 7 or 13



Deleting a leaf is simple.

BinaryTree delete(Integer key)

```
public void delete(Integer key) {

    if (root == null)
        return;

    root = root.delete(key);
}
```

BinaryTree delete(Integer key)

```
public void delete(Integer key) {

    if (root == null)
        return;

    root = root.delete(key);
}
```

Node delete(key) should return a new root where the key-value node is deleted.

Node delete(Integer key)

```
private Node delete(Integer k) {
    if (this.key == k) {
      // what do we do?
    }
    if (this.key < k && this.right != null)  {
        Node deleted = this.right.delete(k);
        this.right = deleted;
        return this;
    }
    if (this.key > k && this.left != null)  {
        :
    }
    return this;
}
```

Node delete(Integer key)

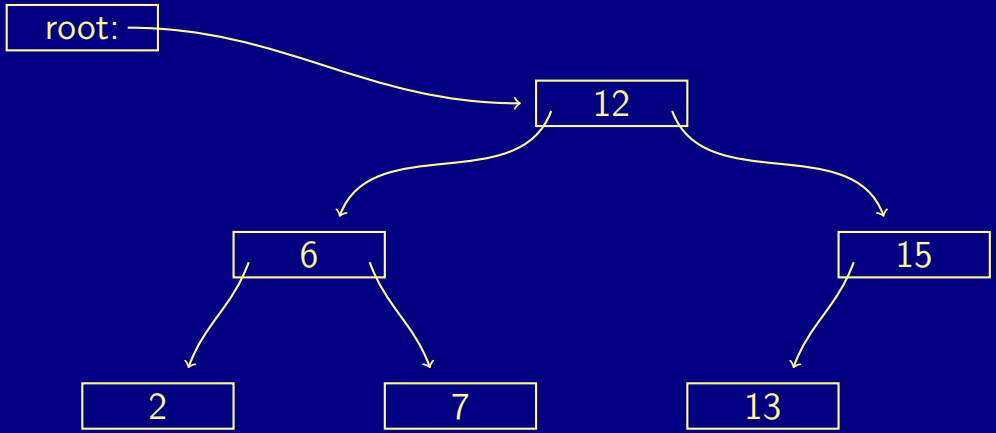
```
if (this.key == k) {
  if (this.left == null)
     return this.right;
  if (this.right == null)
     return this.left;
  :
  :
}
```

Node delete(Integer key)

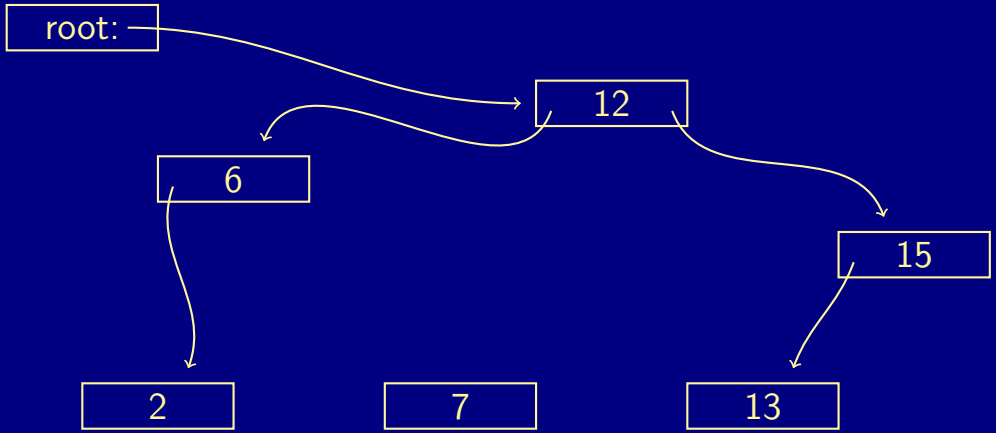
```
if (this.key == k) {
  if (this.left == null)
     return this.right;
  if (this.right == null)
     return this.left;
  :
  :
}
```

This takes care of all the simple cases.

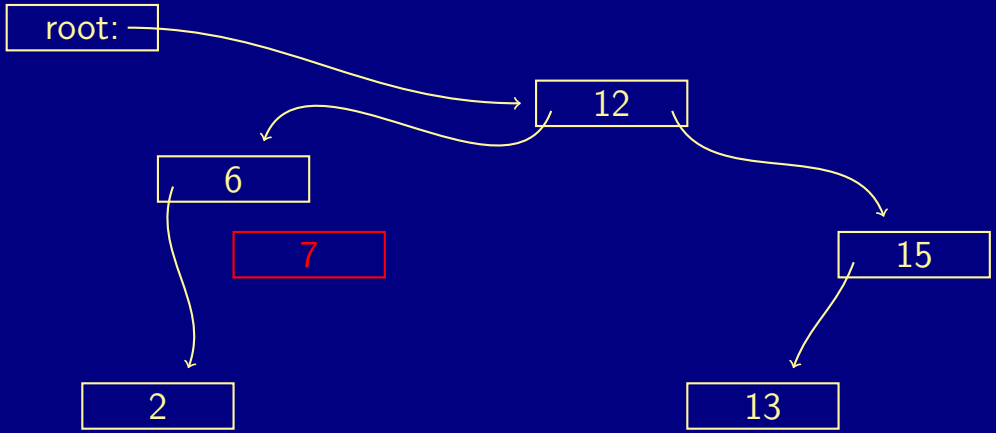
delete 6



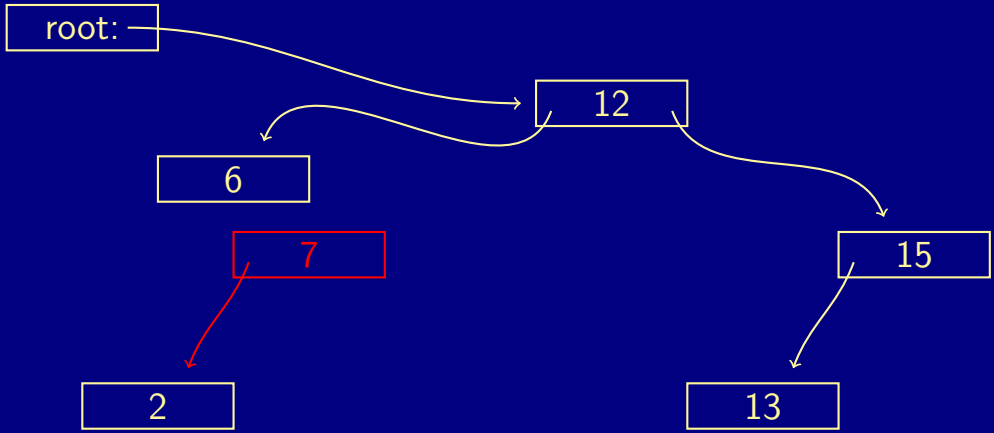
delete 6



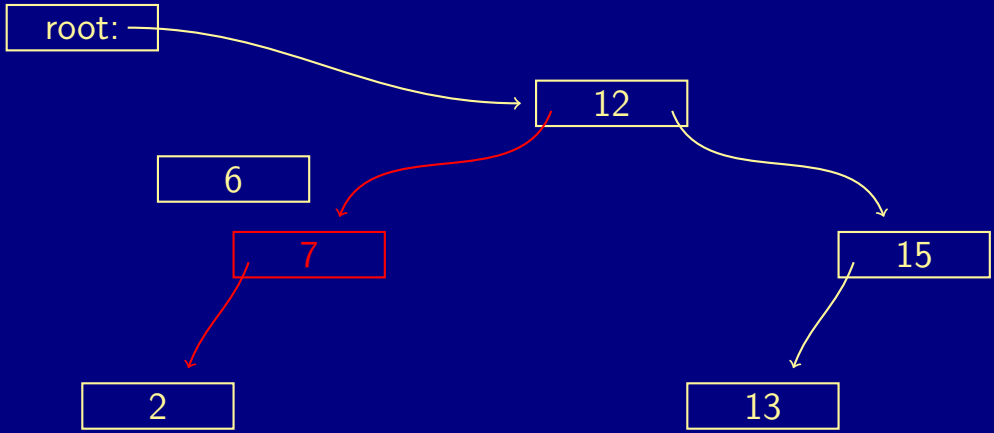
delete 6



delete 6



delete 6



Node delete(Integer key)

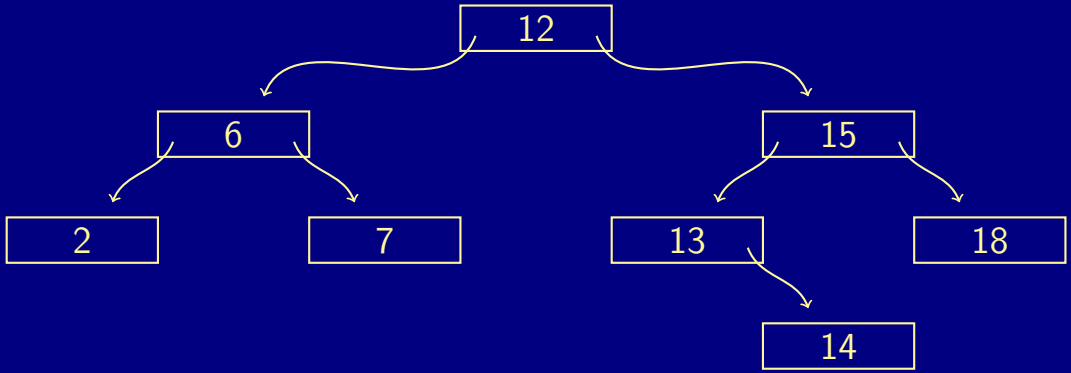
```
if (this.key == k) {
  if (this.left == null)
     return this.right;
  if (this.right == null)
     return this.left;
  Node promoted = this.right.promote();
  promoted.left = this.left;
  return promoted;
}
  :
```

Node delete(Integer key)

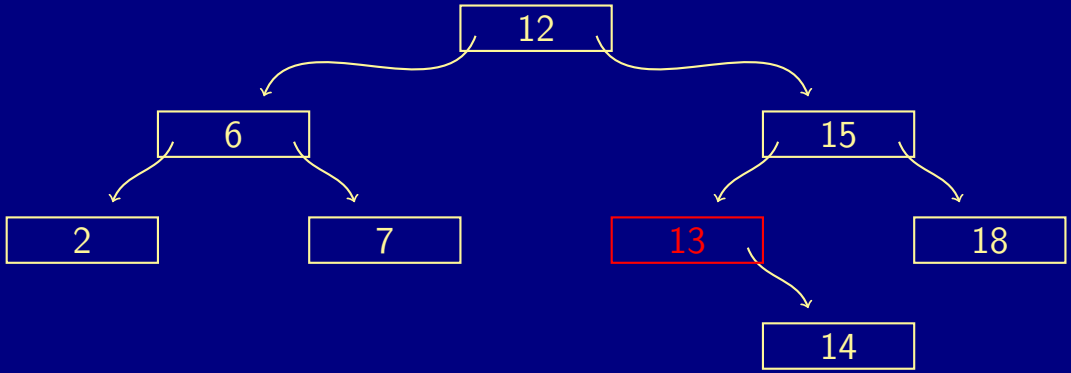
```
if (this.key == k) {
  if (this.left == null)
     return this.right;
  if (this.right == null)
     return this.left;
  Node promoted = this.right.promote();
  promoted.left = this.left;
  return promoted;
}
  :
```

What should promote() do?.

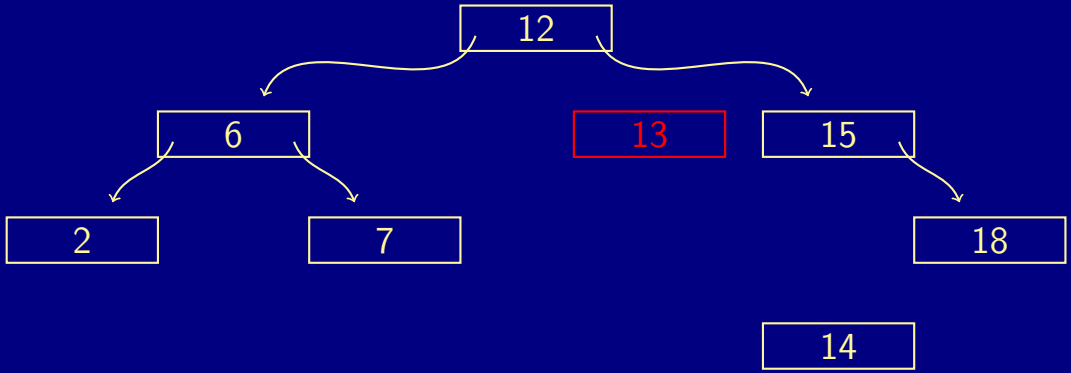
delete 12 by promoting 13



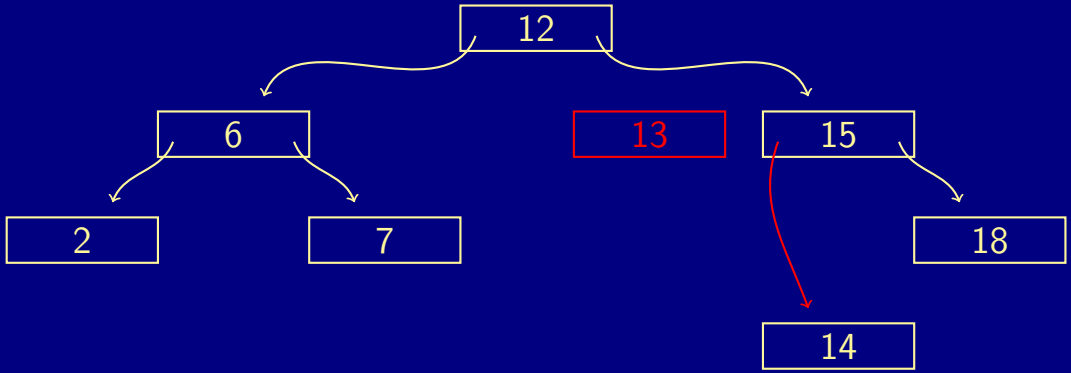
delete 12 by promoting 13



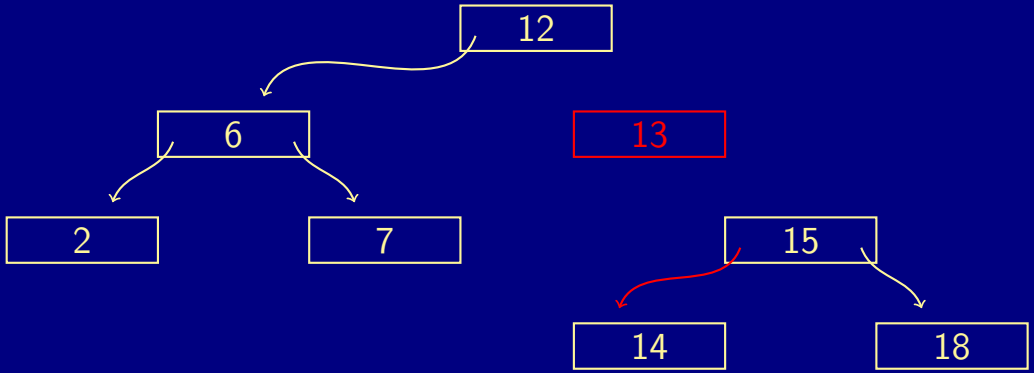
delete 12 by promoting 13



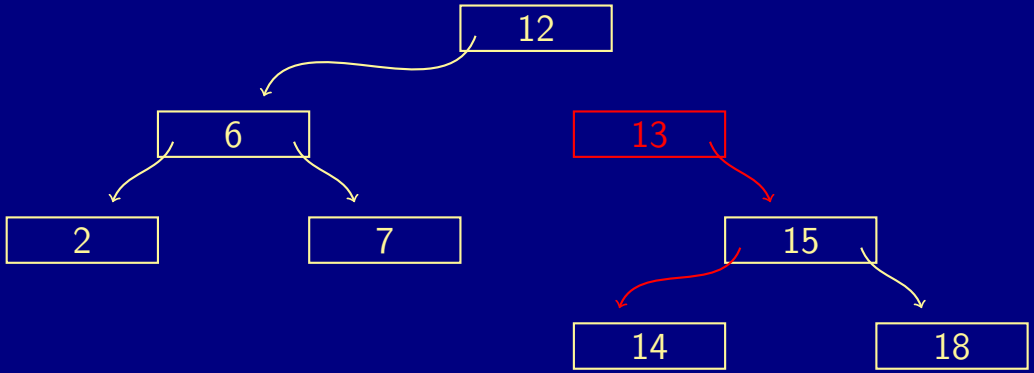
delete 12 by promoting 13



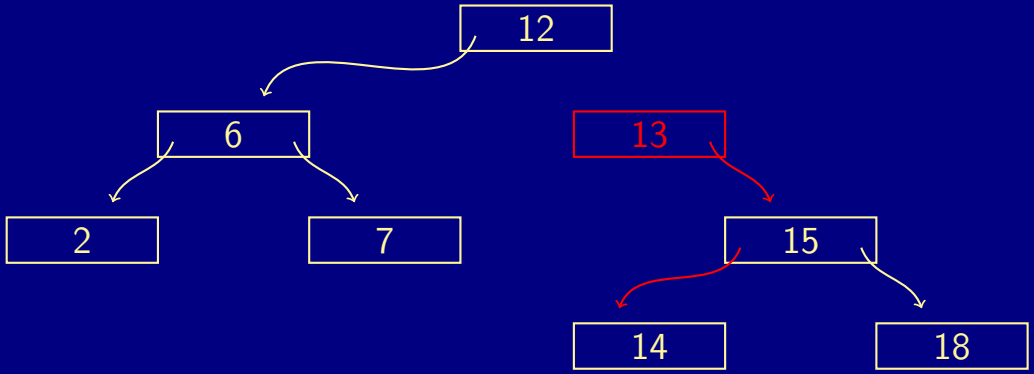
delete 12 by promoting 13



delete 12 by promoting 13

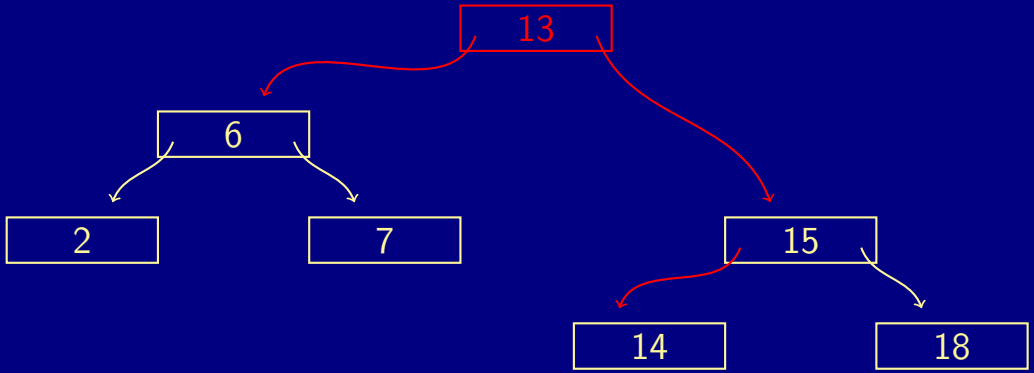


delete 12 by promoting 13



The method promote() should return the root of an ordered tree, with the left branch empty.

delete 12 by promoting 13



Node promote()

```
private Node promote() {
    if (this.left == null)
        return this;
    Node cur = this;
    while ( cur.left.left != null) {
        cur = cur.left;
    }
    Node ret = cur.left;
    cur.left = cur.left.right;
    ret.right = this;
    return ret;
}
```

nota bene

nota bene

The add() and delete() operatins that we have will not keep the tree well balanced.