Binary Search Trees

Lecture 33
Section 19.2

Robb T. Koether

Hampden-Sydney College

Fri, Apr 14, 2017

Outline

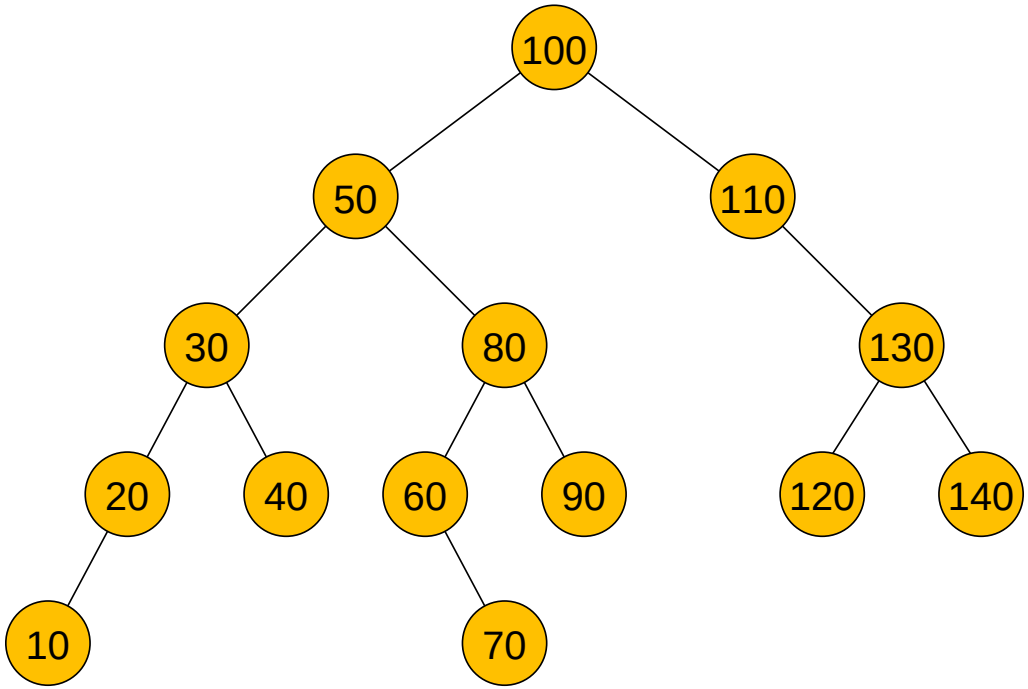
Binary Search Trees

Definition (Binary Search Tree)

A binary search tree is a binary tree with the following properties.

- There is a total order relation on the members in the tree.
- At every node, every member of the left subtree is less than or equal to the node value.
- At every node, every member of the right subtree is greater than or equal to the node value.

A Binary Search Tree



A binary search tree

Binary Search Tree Interface

Mutators

```
void insert(const T& value);
void remove(const T& value);
```

- `insert()` – Insert a new node containing the value into the binary search tree.
- `remove()` – Remove the node containing the value from the binary search tree.

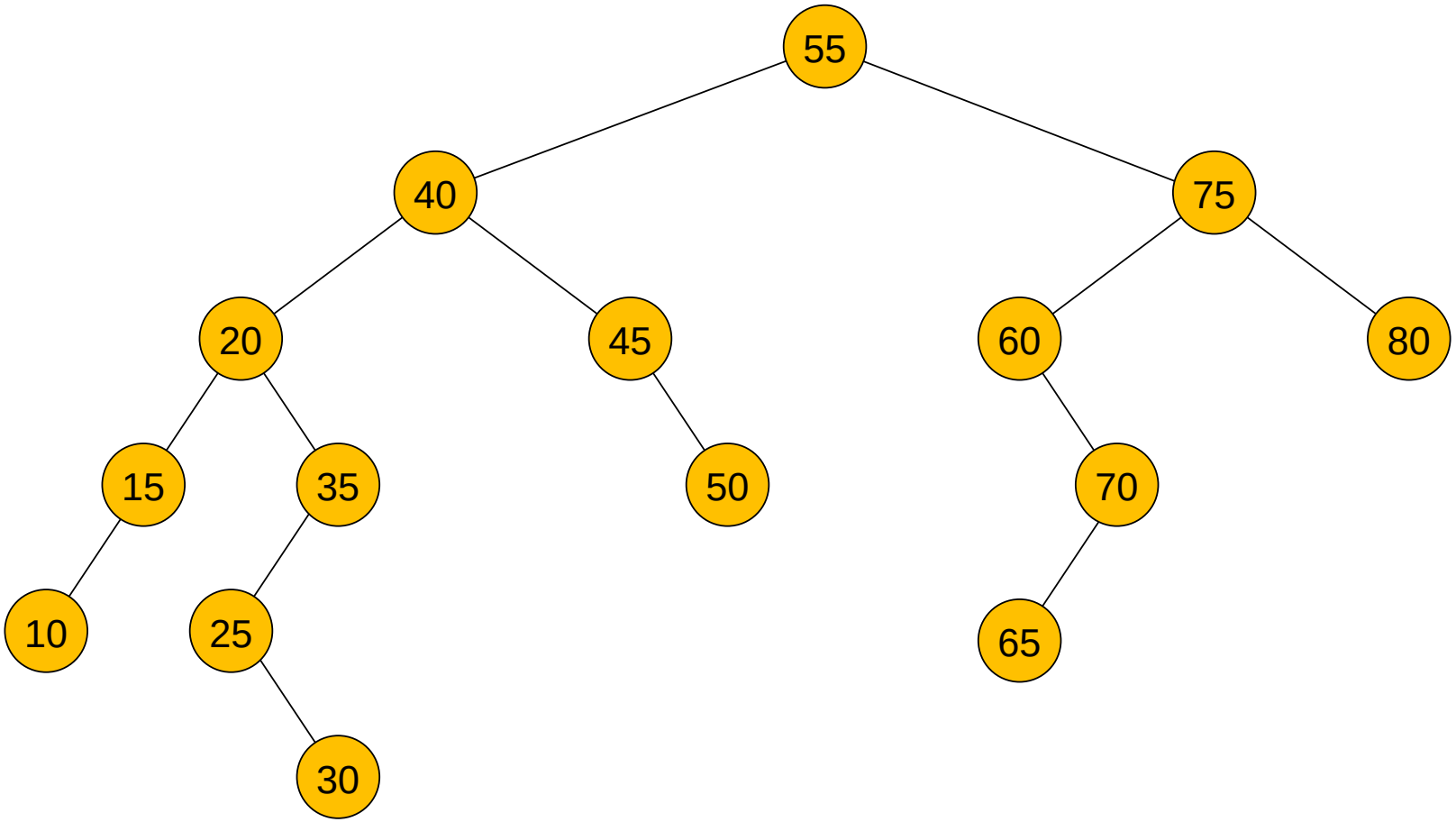
Binary Search Tree Interface

Other Member Functions

```
T* search(const T& value) const;
void countBalance();
```

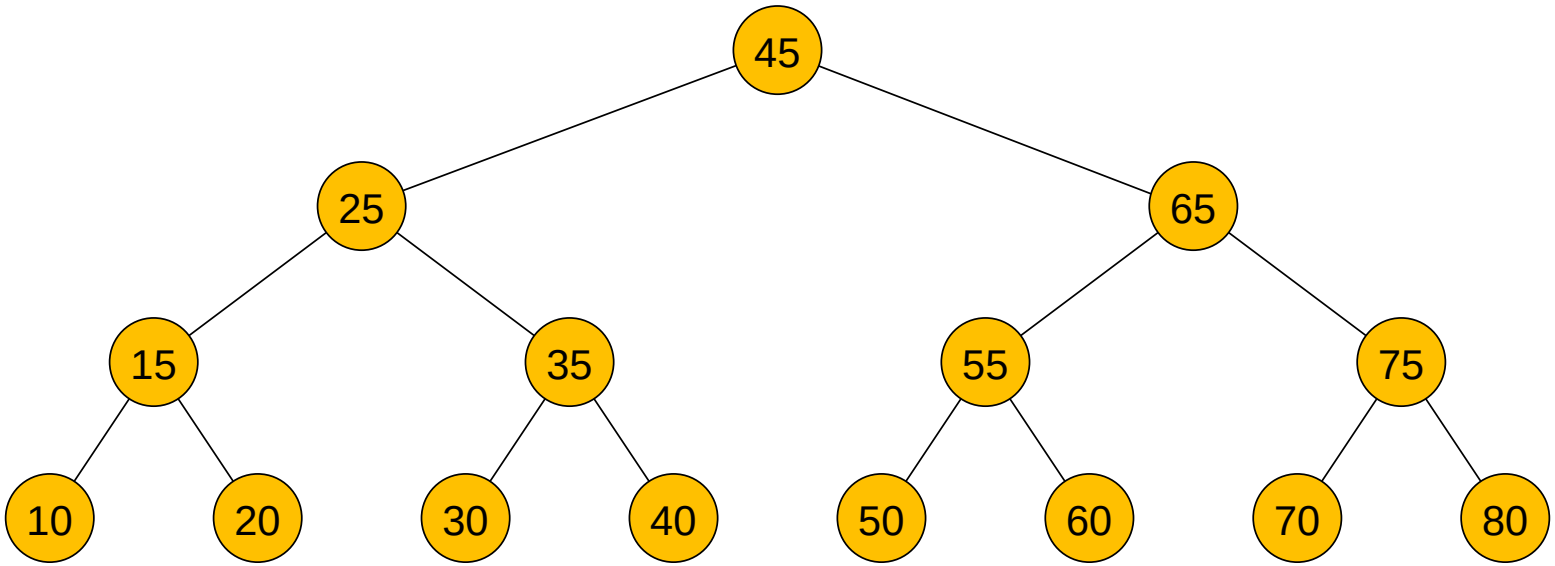
- `search()` – Search the binary search tree for the value. Return a pointer to the node where the value is found. Return `NULL` if the value is not found.
- `countBalance()` – Count-balance the binary search tree.

Balanced vs. Unbalanced Trees



For the unbalanced tree, what is the average number of comparison?

Balanced vs. Unbalanced Trees



For the balanced tree, what is the average number of comparison?

Outline

Searching a BinarySearchTree

Searching a Binary Search Tree

- Beginning at the root node, apply the following steps recursively.
  - Compare the value to the node data.
  - If it is equal, you are done.
  - If it is less, search the left subtree.
  - If it is greater, search the right subtree.
  - If the subtree is empty, the value is not in the tree.

Outline

Inserting a Value into a BinarySearchTree

Inserting into a Binary Search Tree

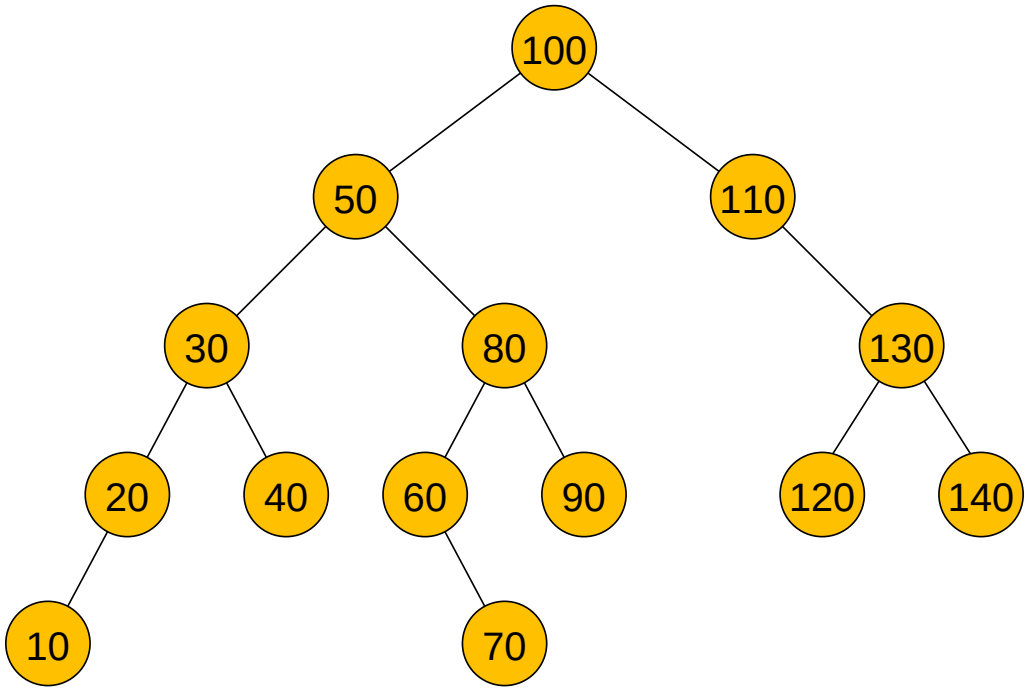
- Beginning at the root node, apply the following steps recursively.
  - Compare the value to the node data.
  - If it is less (or equal), continue recursively with the left subtree.
  - If is is greater, continue recursively with the right subtree.
  - When the subtree is empty, attach the node as a subtree.

Outline

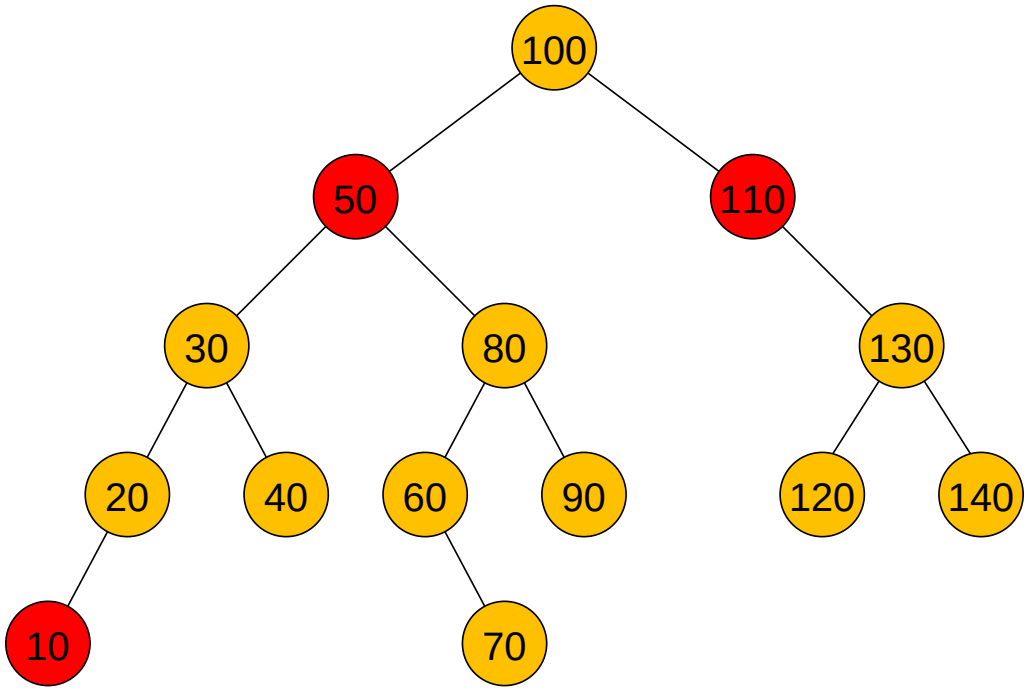
Deleting a Value from a BinarySearchTree

- Perform a search to locate the value.
- This node will have
  - Two children, or
  - One child, or
  - No child.

A Binary Search Tree



A Binary Search Tree

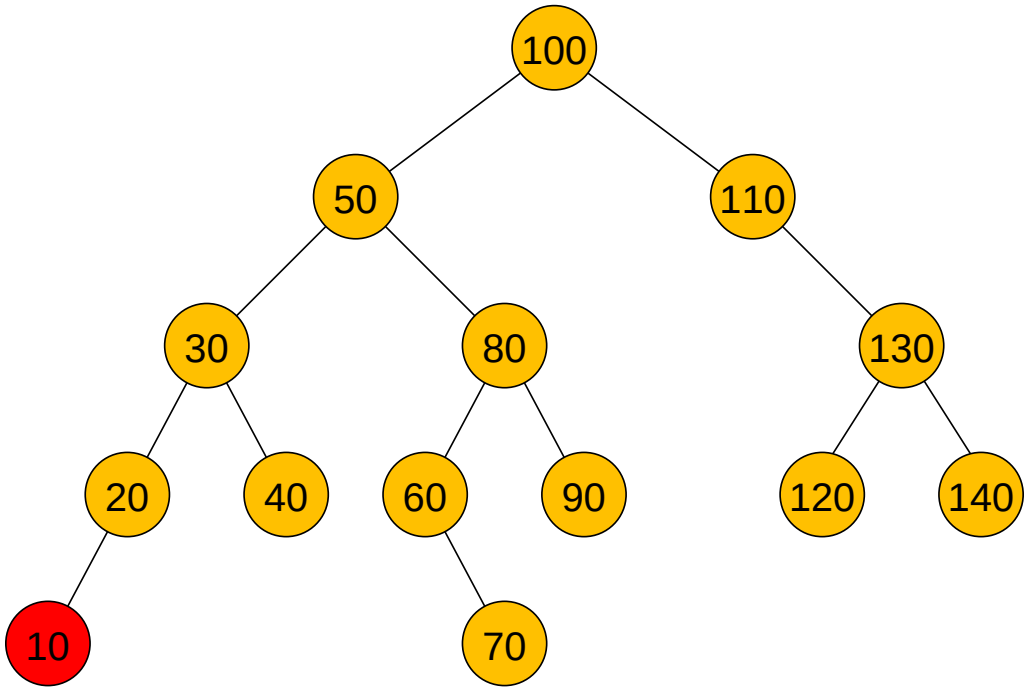


Deleting a Value from a `BinarySearchTree`

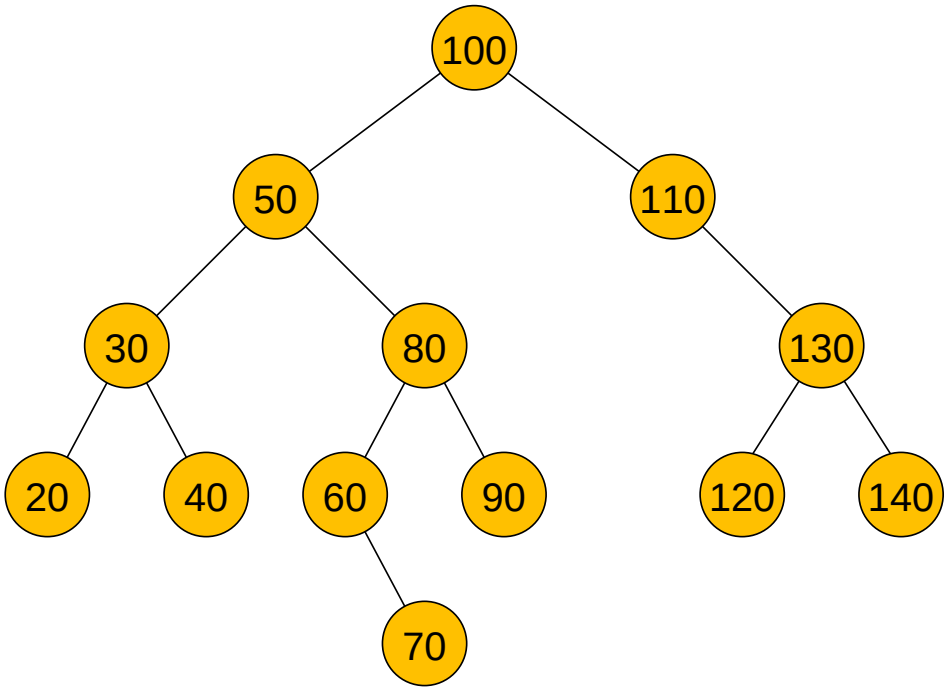
Case 1: No Child

- Delete the node.

Delete a Node with No Child



Delete a Node with No Child

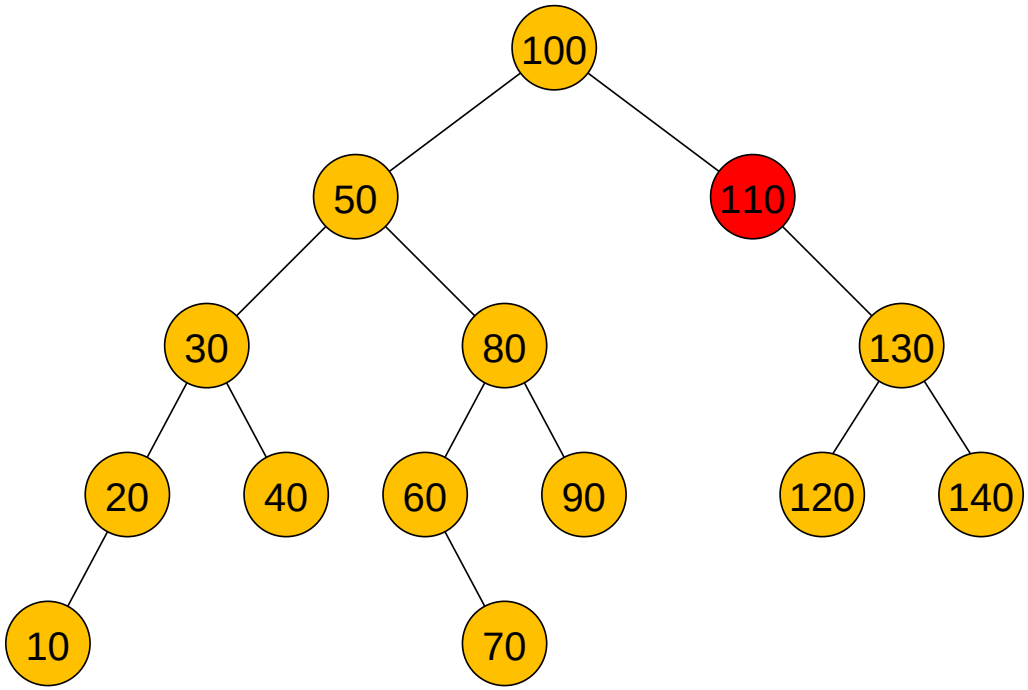


Deleting a Value from a `BinarySearchTree`

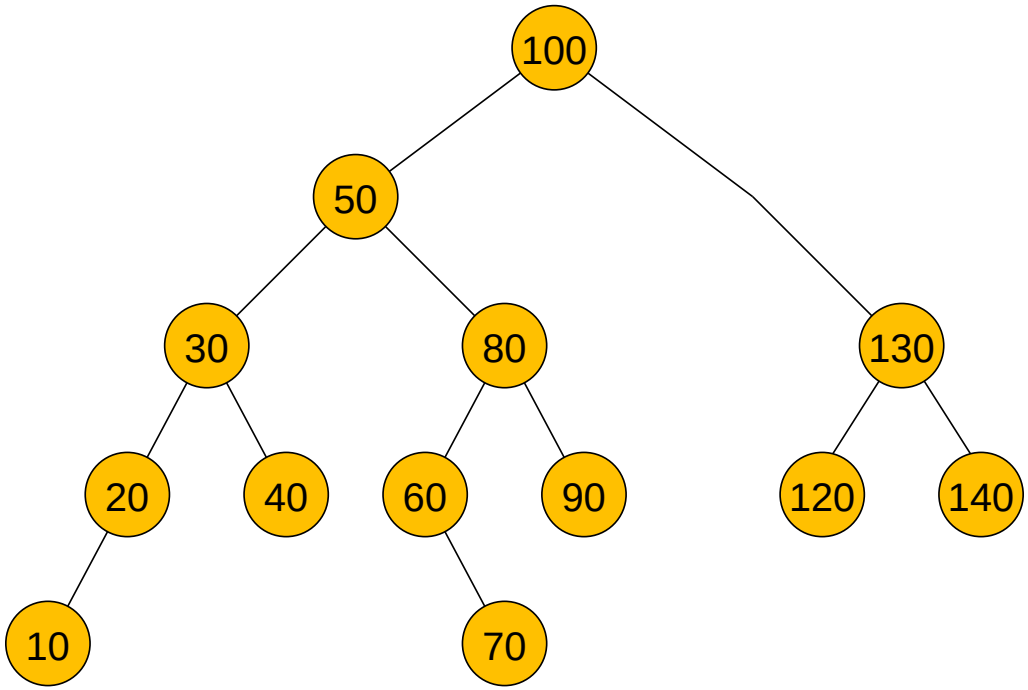
Case 2: One Child

- Replace the node with the subtree of which the child is the root.

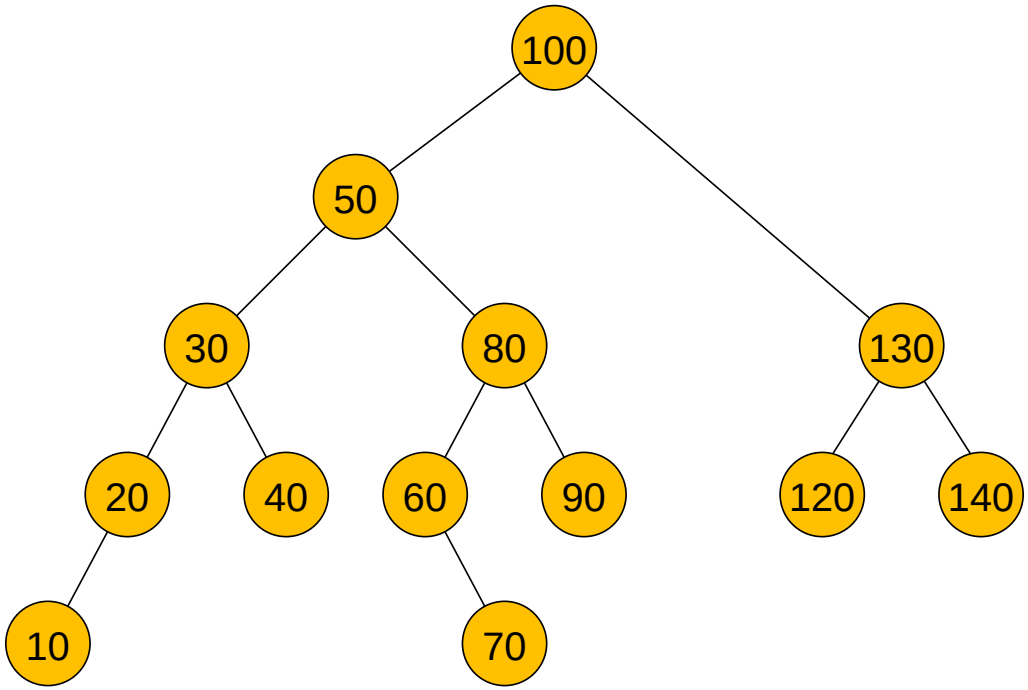
Delete a Node with Two Children



Delete a Node with Two Children



Delete a Node with Two Children

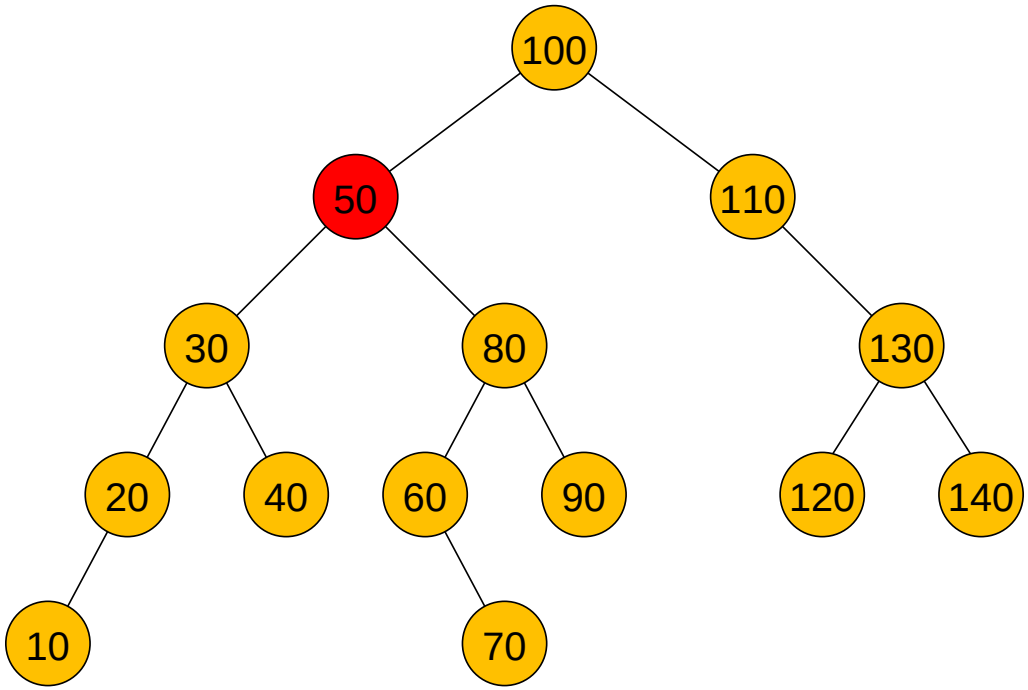


Deleting a Value from a BinarySearchTree

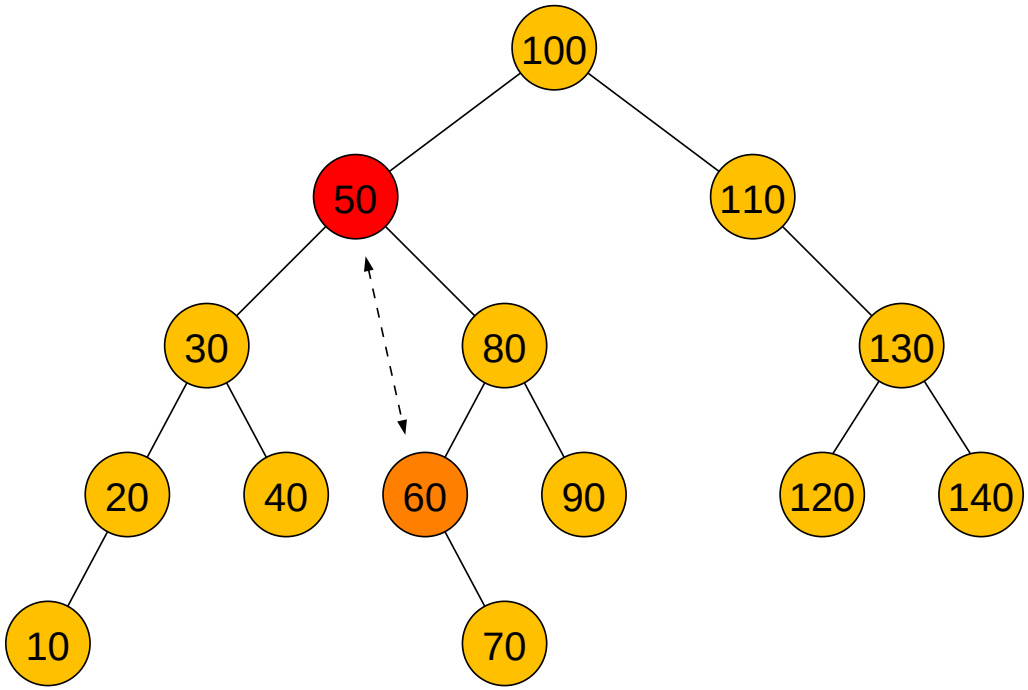
Case 3: Two Children

- Locate the next smaller value in the tree. This value is the rightmost value of the left subtree.
  - Move left one step.
  - Move right as far as possible.
- Swap this value with the value to be deleted.
- The node to be deleted now has at most one child.

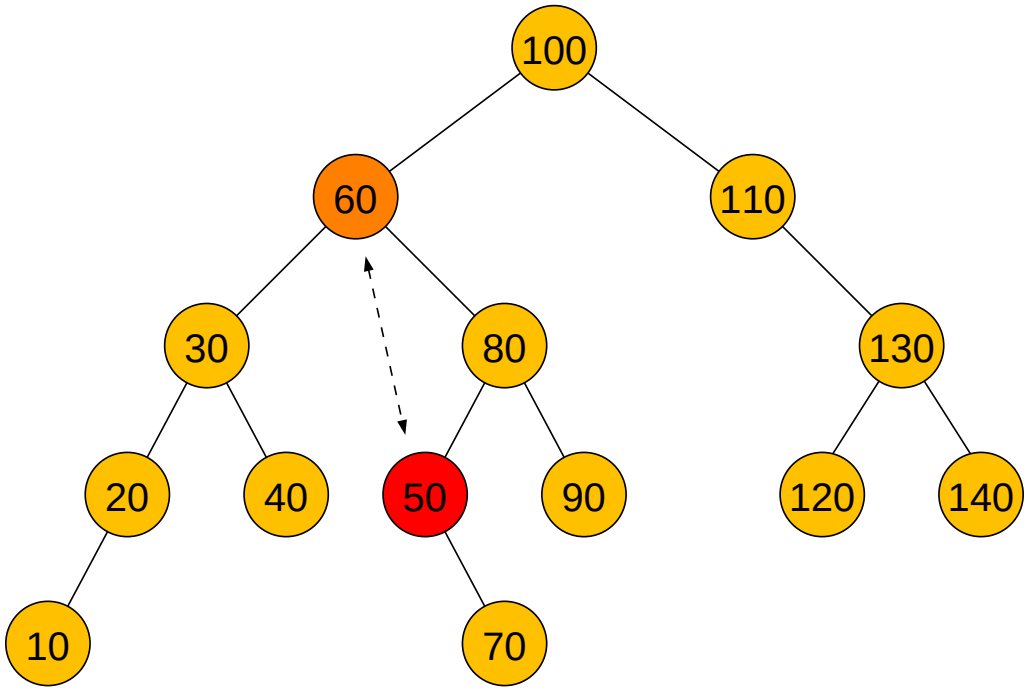
Delete a Node with Three Children



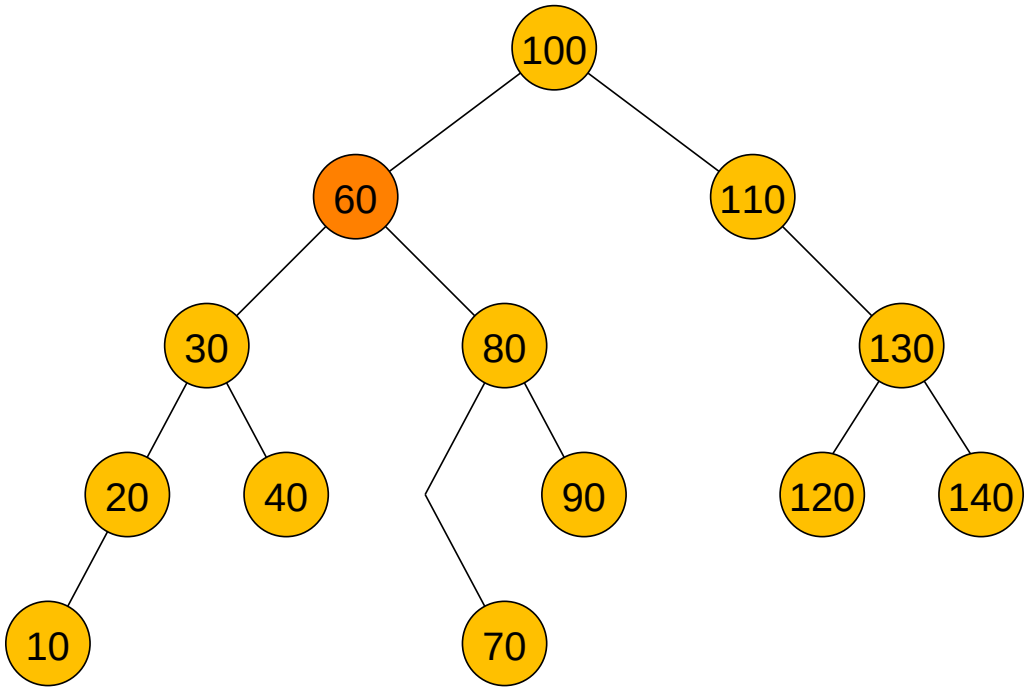
Delete a Node with Three Children



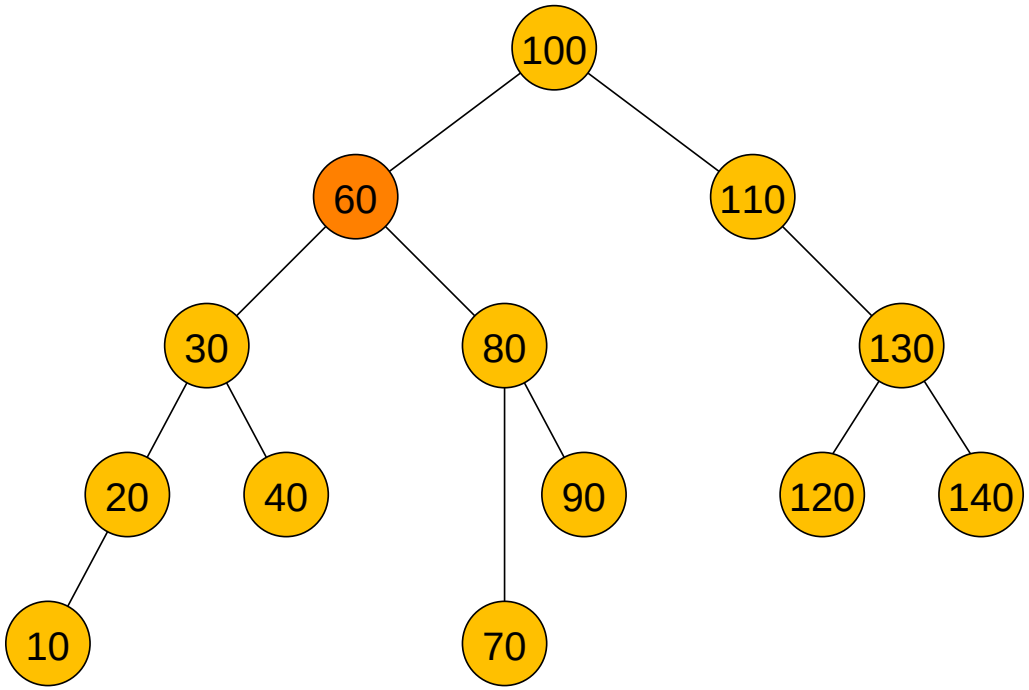
Delete a Node with Three Children



Delete a Node with Three Children



Delete a Node with Three Children



Outline

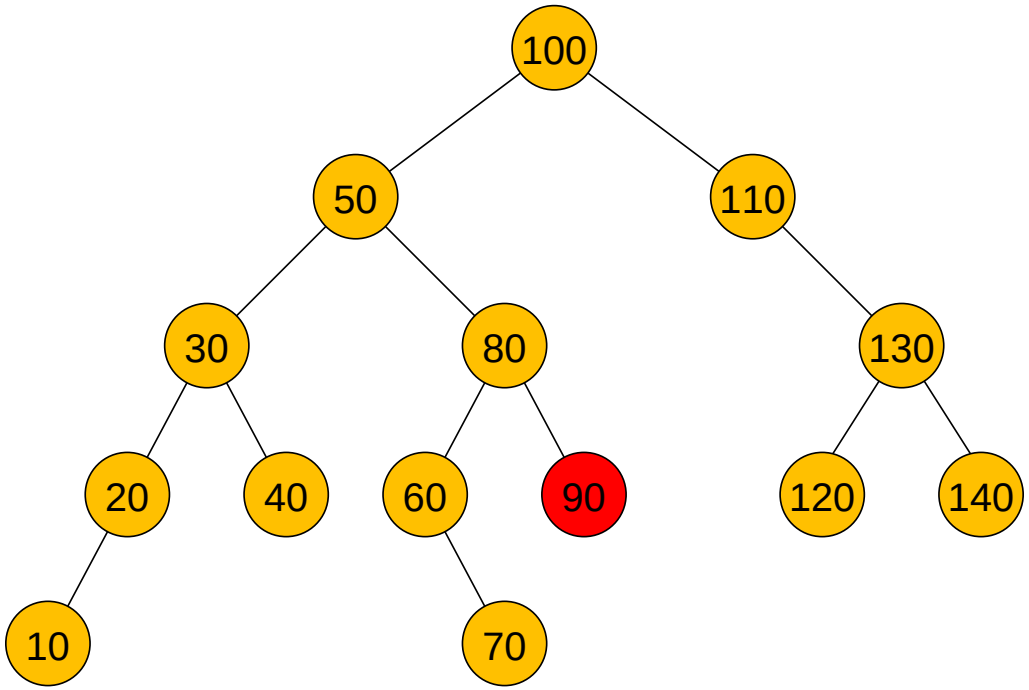
Count-Balancing a `BinarySearchTree`

- Write a function `moveNodeRight()` that will move the largest value of the left subtree to the right subtree.

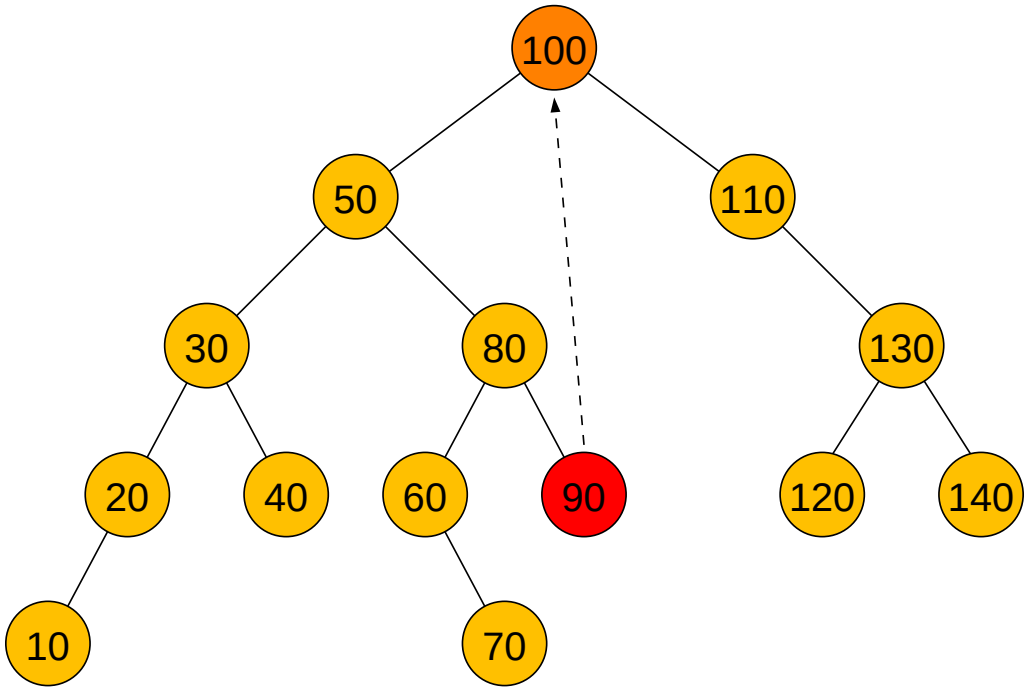
The `moveNodeRight()` Function

- Locate the largest value in the left subtree.
- Delete it (but save the value).
- Place it at the root.
- Insert the old root value into the right subtree.

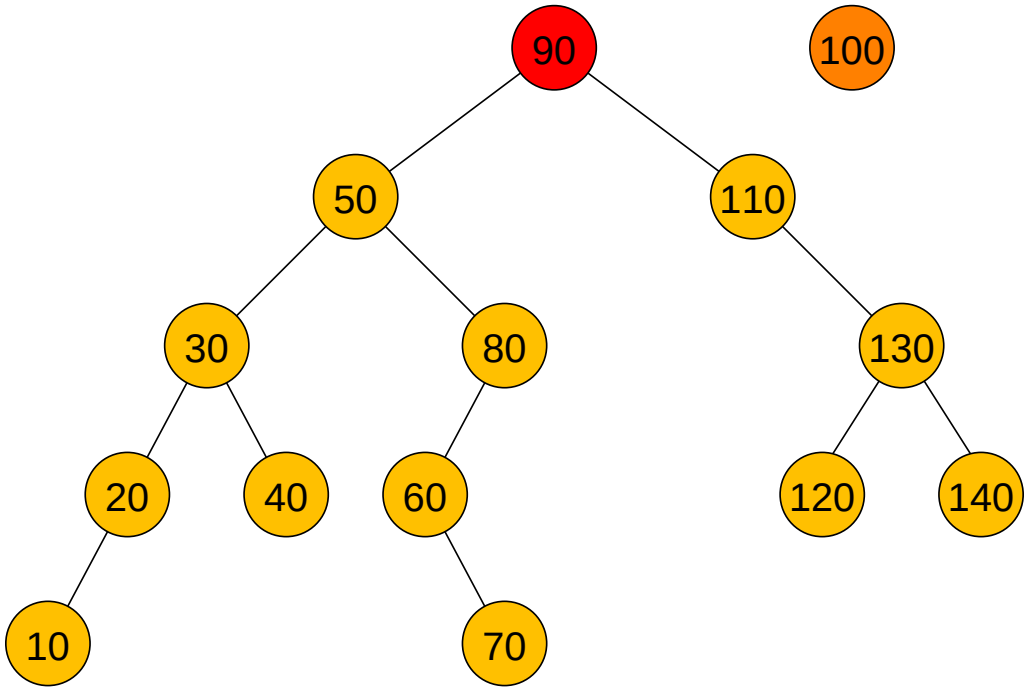
A Binary Search Tree



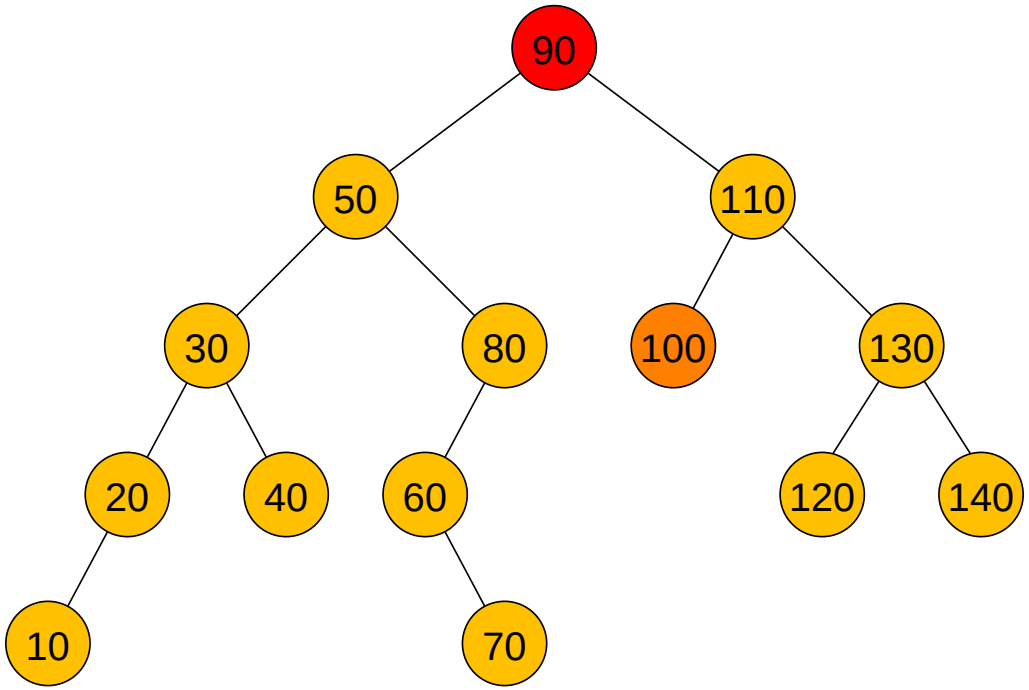
A Binary Search Tree



A Binary Search Tree



A Binary Search Tree



Count-Balancing a `BinarySearchTree`

Count-Balancing a Tree

- Write a similar function `moveNodeLeft()`.
- Apply either `moveNodeRight()` or `moveNodeLeft()` repeatedly at the root node until the tree is balanced at the root.
- Then apply these functions recursively, down to the leaves.

Building a Binary Search Tree

- Suppose we wish to transmit the nodes of a balanced binary search tree to another computer and reconstruct the tree there.
- In what order should the values be transmitted?

Building a Binary Search Tree

- We could use an in-order traversal to transmit them.
- At the receiving end, simply call `insert()` to insert each value into the tree.
- The constructed tree will be identical to the original.
- What do we get if we transmit the values using a pre-order traversal?
- Using a post-order traversal?

Outline

Assignment

Assignment

- Read Section 19.2.