Lecture 3

Instructor: Karteek Sreenivasaiah

10th August 2018

Binary Search Trees

Recall that a Binary Search Tree (BST) has the following crucial property:
For every node X in the BST, we have:

- Every node in the left subtree of X contains a value smaller than that of X.
- Every node in the right subtree of X contains a value larger than that of X.

Binary Search Tree

A BST supports the following functions:

- INSERT(*node*, *val*) – Inserts *val* into the BST rooted at *node*.
- SEARCH(*node*, *val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- SUCC(*val*) – Returns the smallest element greater than *val* in the BST.
- PRED(*val*) – Returns the largest element lesser than *val* in the BST.
- DELETE(*val*) – Deletes *val* from the BST.

Successor

The SUCC(VAL) procedure is as follows:

- Find the node which stores *val*. Refer to this node as "*node*".
- Two cases:

Case 1: *node* has a right child.
Case 2: *node* does not have a right child.

Successor - Case 1

Case 1

(*node* has a right child)

- Go to the right child.
- Keep descending to the left as long as possible.
- Return the value in the node where we end.

Successor - Case 2

Case 2

(*node* does not have a right child)

Find the nearest ancestor *anc* such that:

- *node* is in the left subtree of *anc*.

Case 2

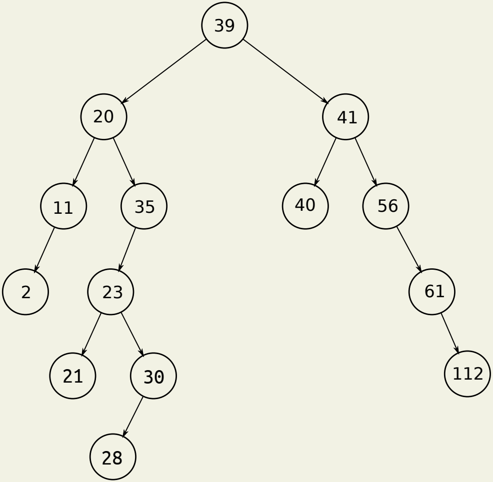
(*node* does not have a right child)

Find the nearest ancestor *anc* such that:

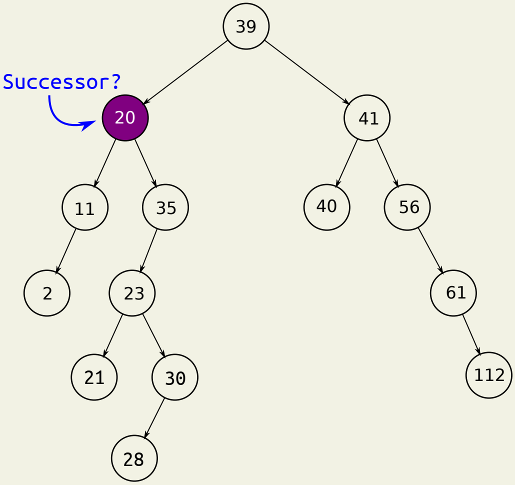
- *node* is in the left subtree of *anc*.

i.e., keep going upwards using the parent pointer till you arrive at such an ancestor.

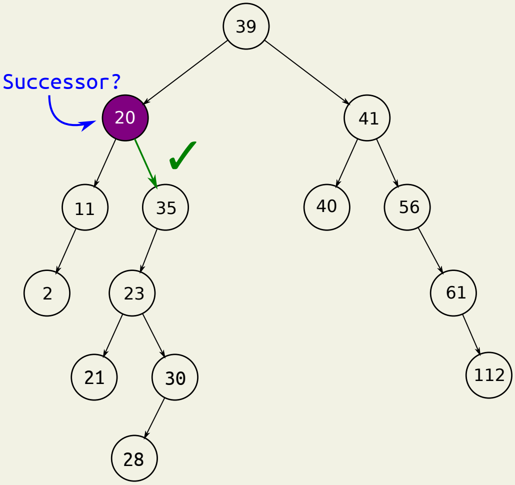
Example



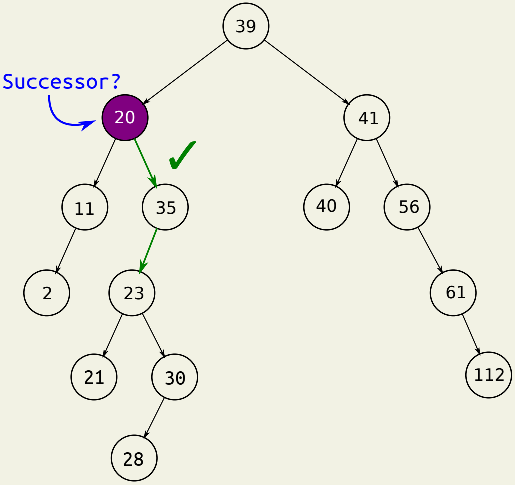
Example



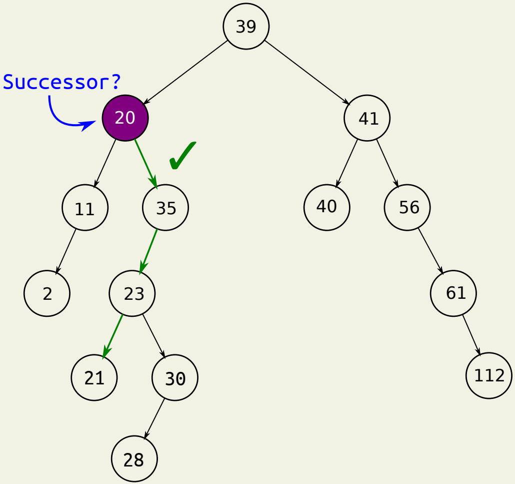
Example



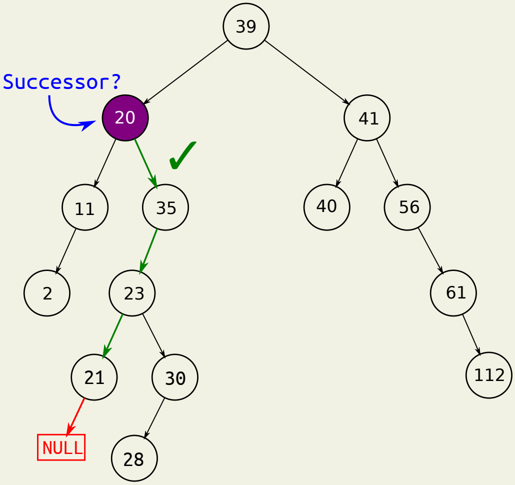
Example



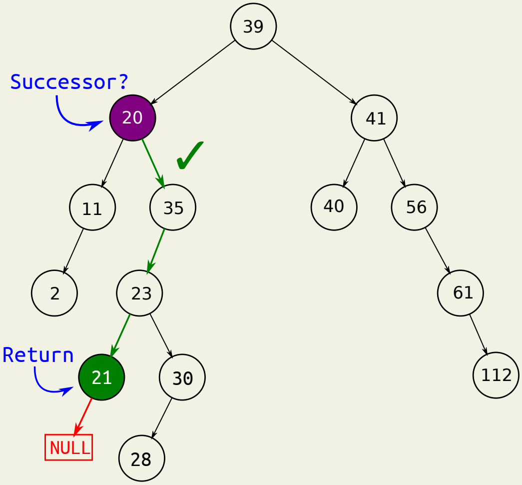
Example



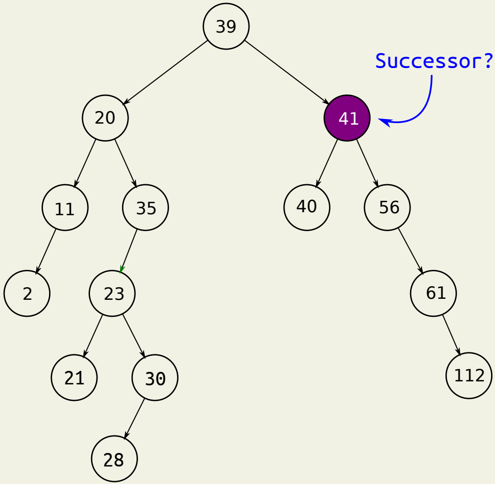
Example



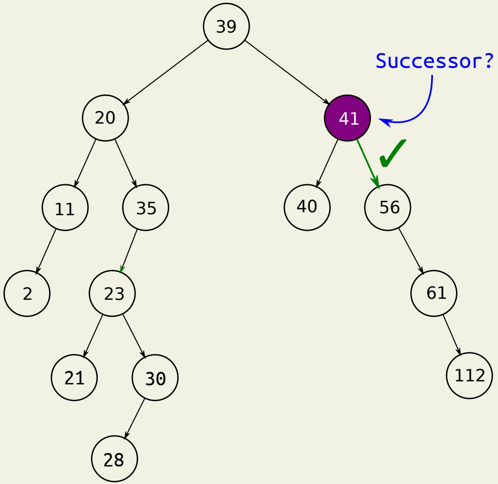
Example



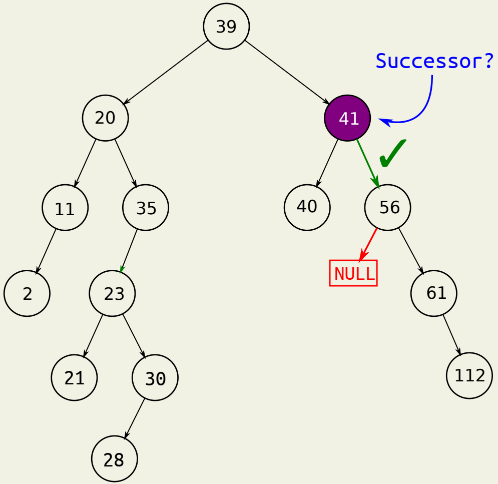
Example



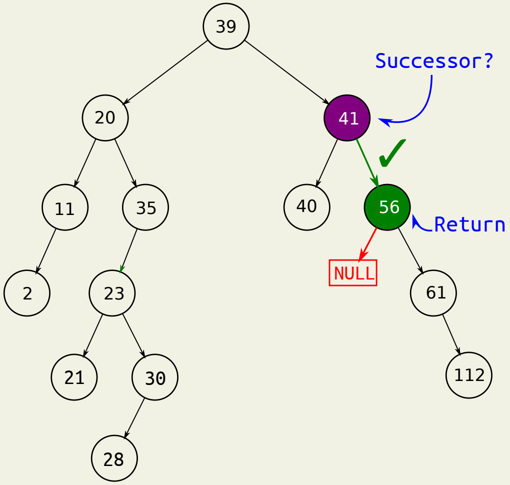
Example



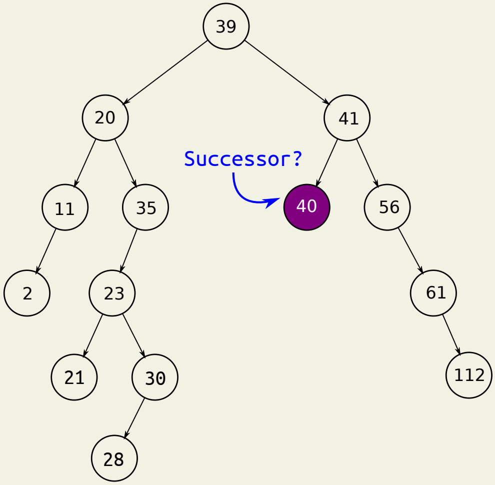
Example



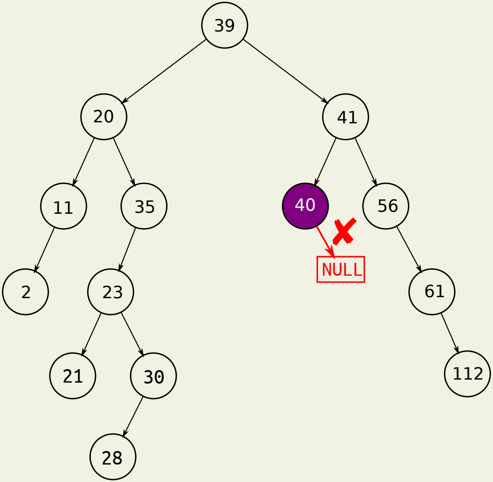
Example



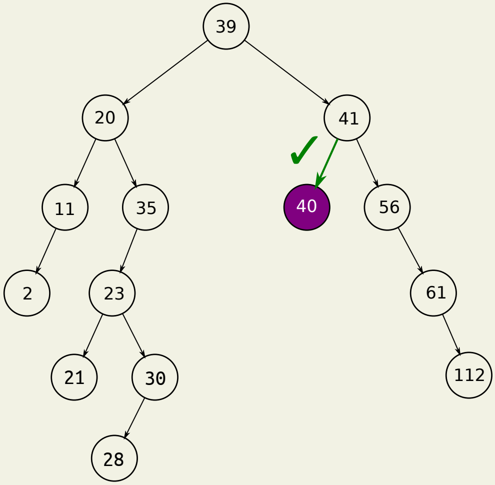
Example



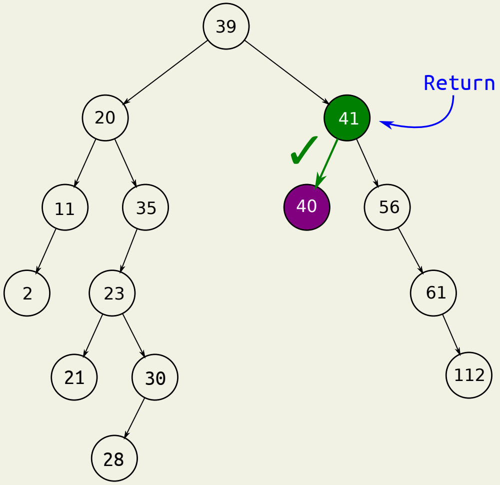
Example



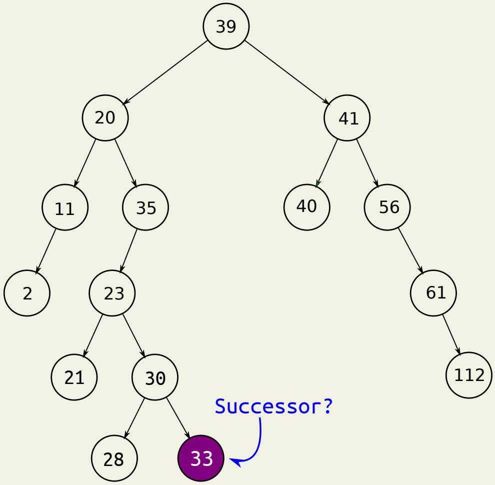
Example



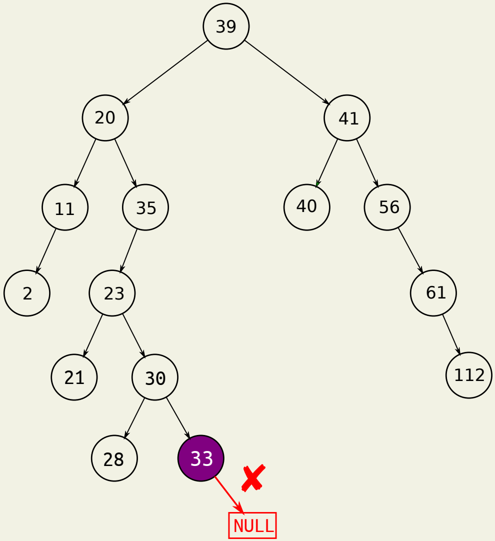
Example



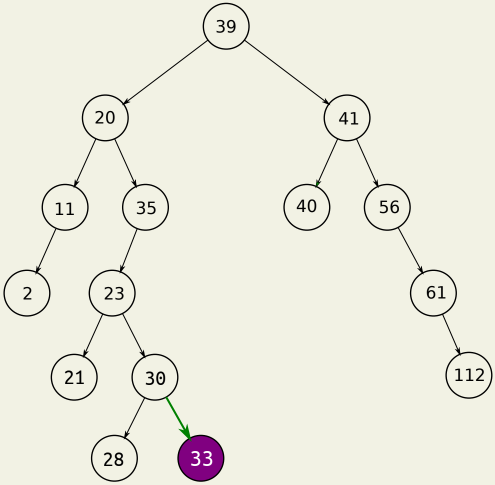
Example



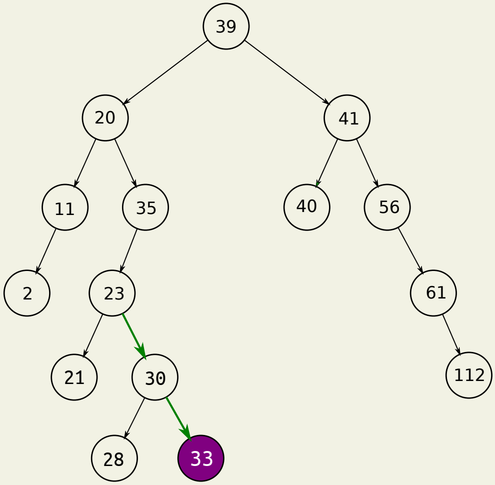
Example



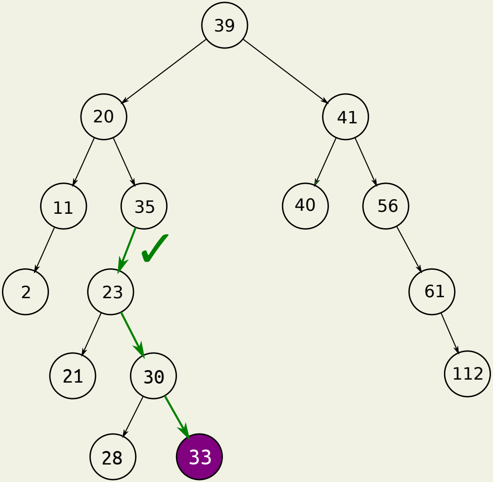
Example



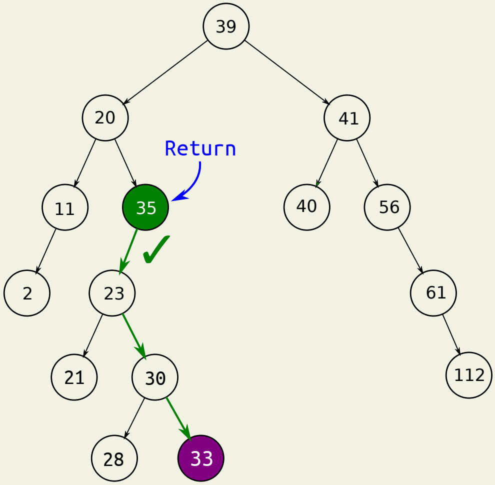
Example



Example



Example



Delete

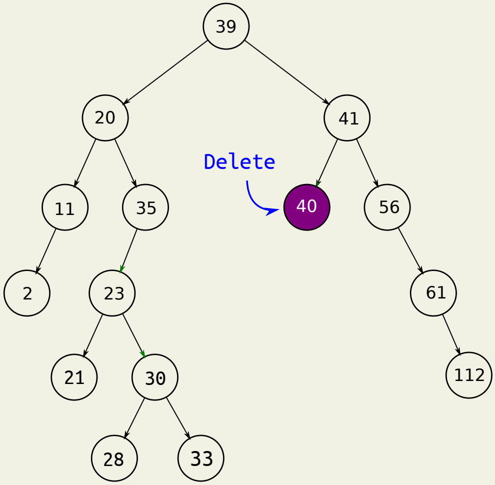
The Delete(val) procedure is as follows:
Find the node that has value *val*.

Delete

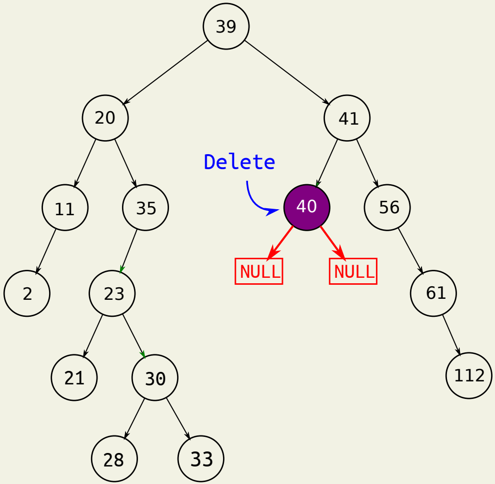
The Delete(val) procedure is as follows:
Find the node that has value *val*.
Three cases:

1. *node* has 0 children. (trivial)
2. *node* has 1 child. (splice)
3. *node* has 2 children:
   - Find successor node X with value x.
   - Splice X out of the tree.
   - Replace *val* with x.
   - Delete node X.

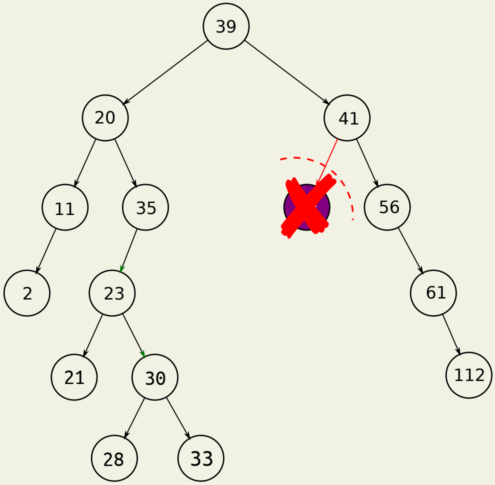
Example



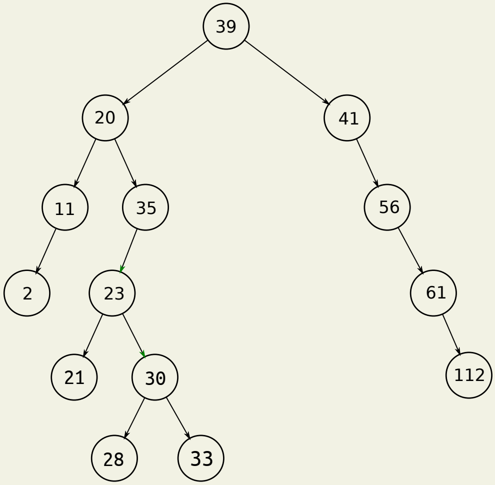
Example



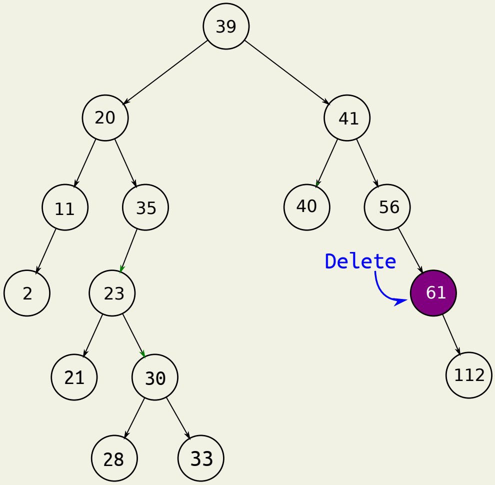
Example



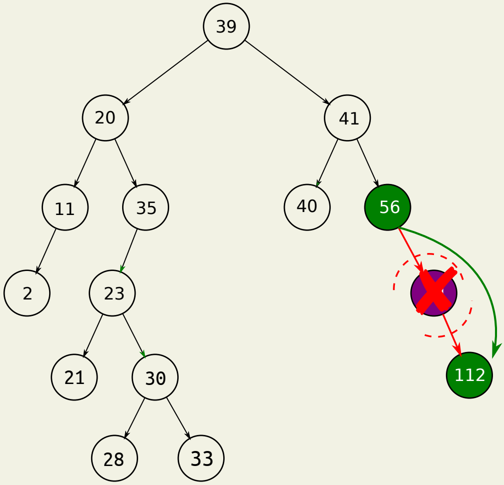
Example



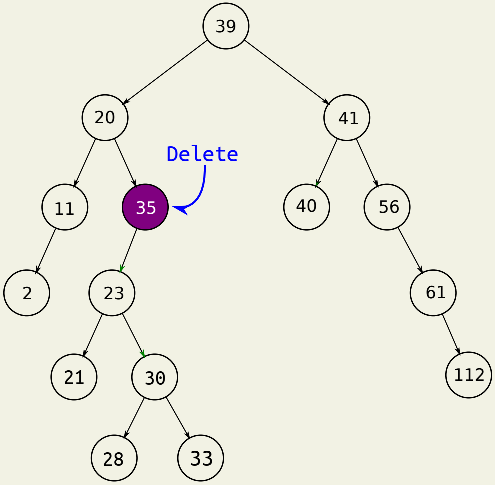
Example



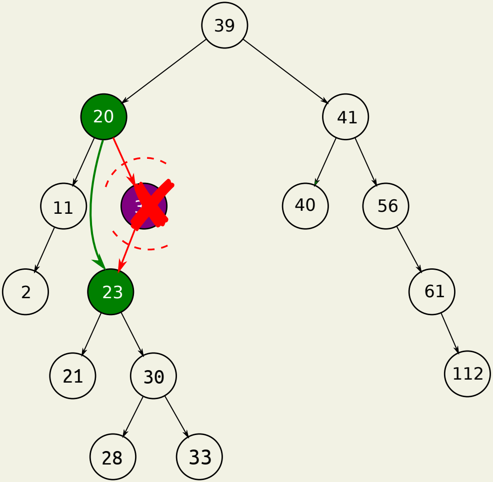
Example



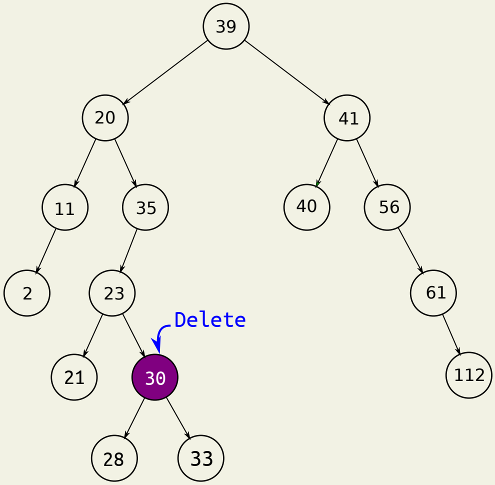
Example



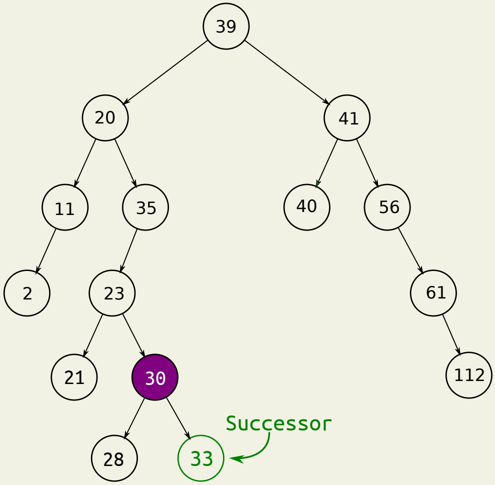
Example



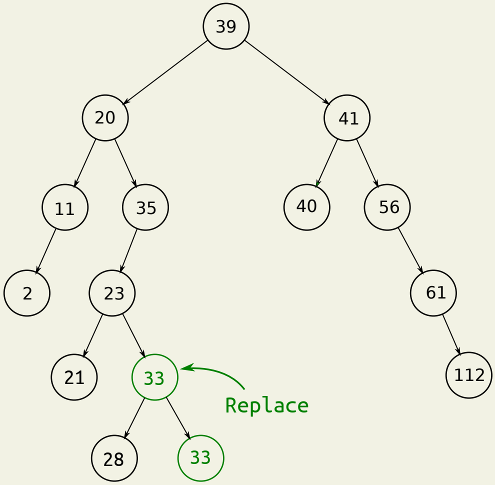
Example



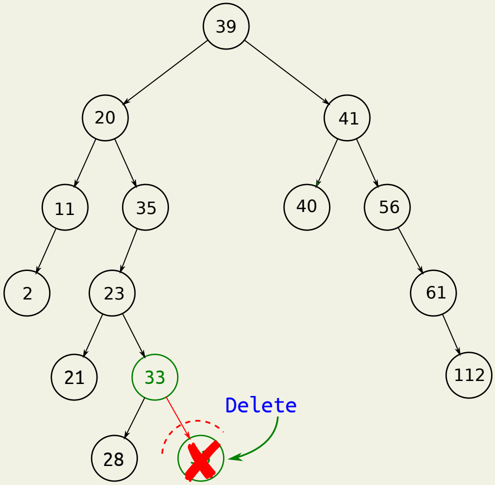
Example



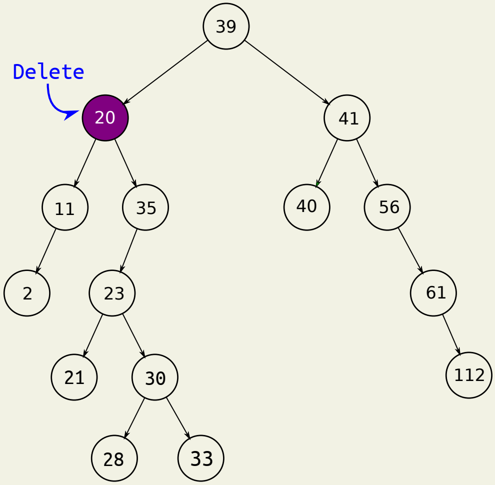
Example



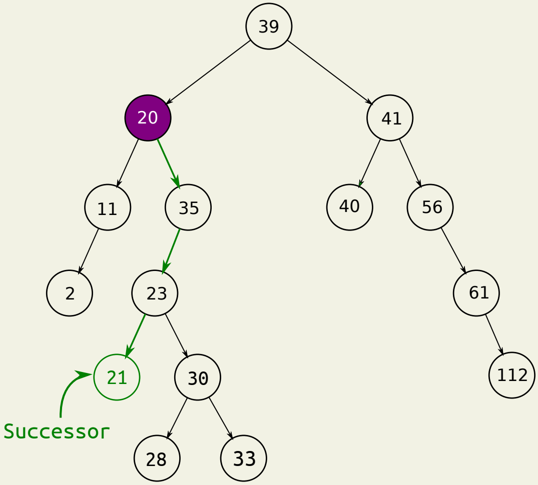
Example



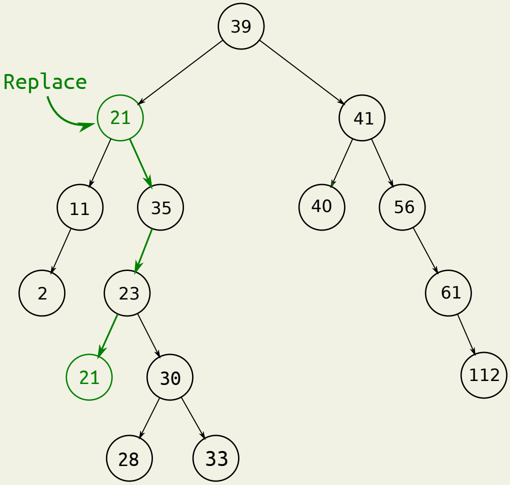
Example



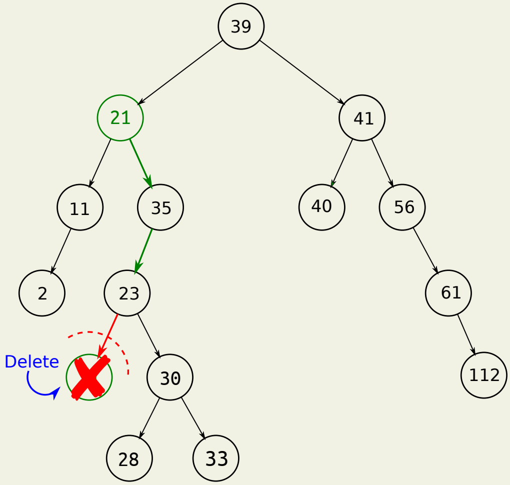
Example



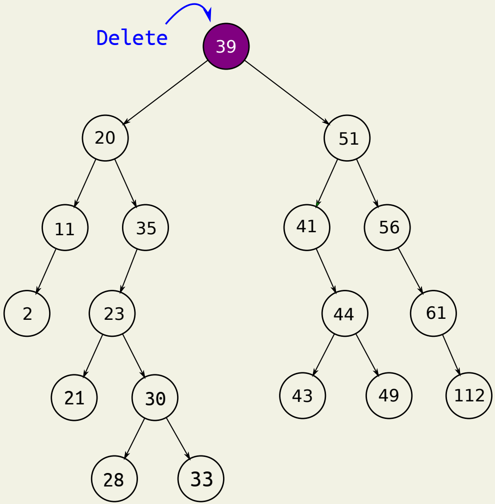
Example



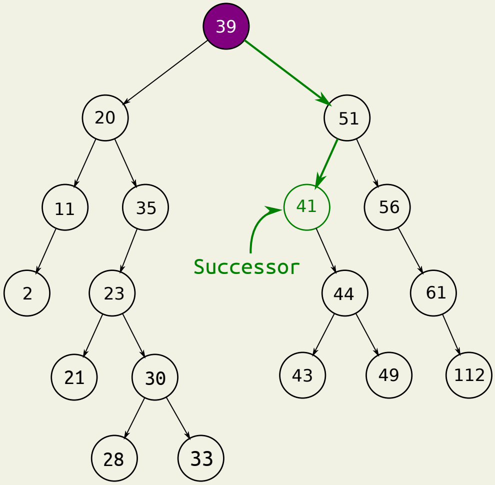
Example



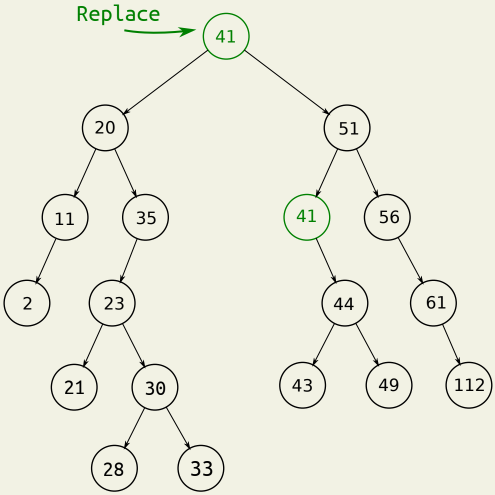
Example



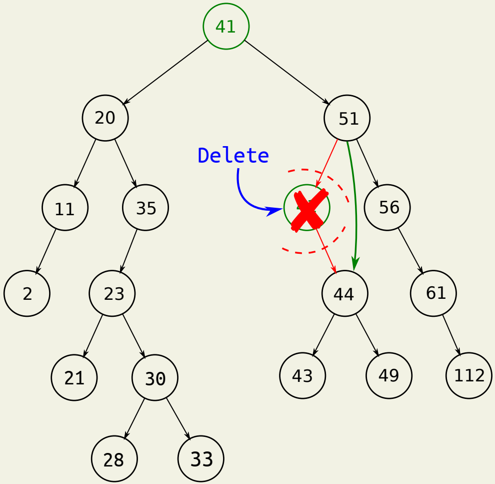
Example



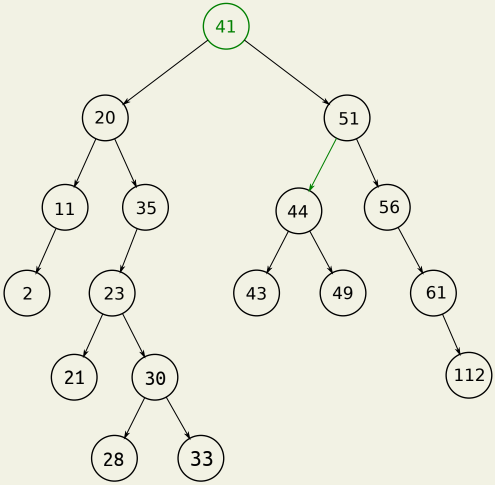
Example



Example



Example



Running Time

Worst case running times for a BST of height h:

- Insert – $O(h)$.
- Succ – $O(h)$.
- Search – $O(h)$.
- Delete – $O(h)$.

The height of a BST depends on the input sequence and can be n after inserting n elements in the worst case.

Balancing a BST

The biggest drawback of BSTs are that they can be quite "unbalanced".
One way to measure if a tree is balanced is to look at the difference between the longest path from root to leaf and the shortest path from root to leaf.

Balancing a BST

The biggest drawback of BSTs are that they can be quite "unbalanced".

One way to measure if a tree is balanced is to look at the difference between the longest path from root to leaf and the shortest path from root to leaf.

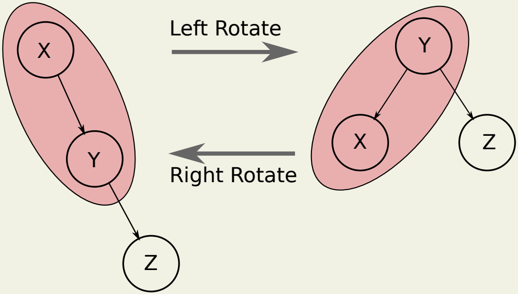
We want to make sure this difference does not get too large.

Rotations

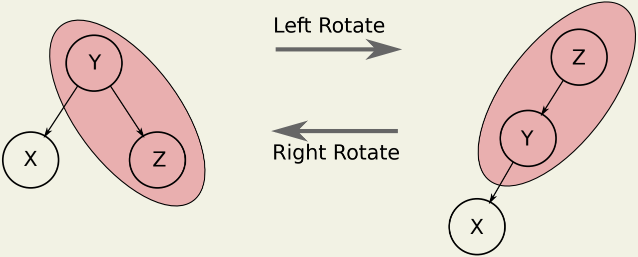
Rotations are operations on nodes of a BST. They are of two variants:

1. Left Rotate.
2. Right Rotate.

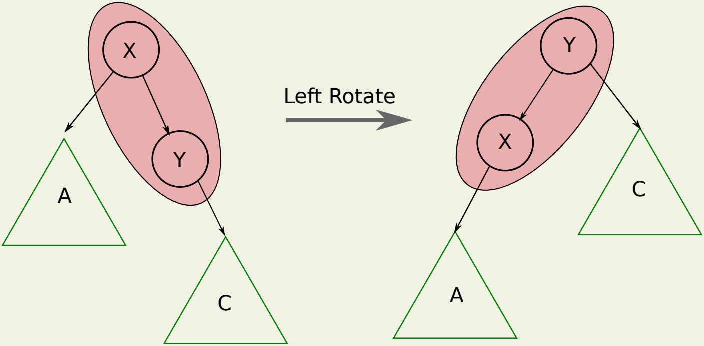
Left Rotate



Left Rotate



Left Rotate



Left Rotate