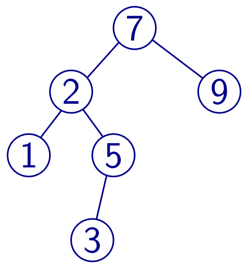
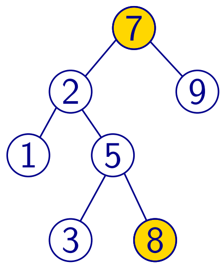
Binary Search Trees

A binary search tree is a a binary tree where each node stores a key and the value that belongs to this key.

Search-tree ordering: If k is the key stored in a node v, then the keys in v's left subtree are all smaller than k, and the keys in v's right subtree are all larger than k.



Binary Search Tree



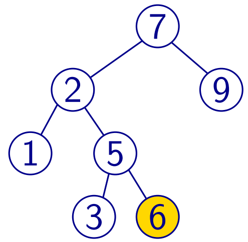
Not Binary Search Tree

Binary search tree operations

`get(key)` and `contains(key)`: Just follow the path from the root until we find the key or reach an empty subtree.

`firstkey()`: Follow the leftmost path.
`lastkey()`: Follow the rightmost path.

`put(key, value)`: Search for the key. If it does not yet exist, then add a new leaf.



`remove(key)`: Hardest operation, implemented like in rank tree (distinguish case of 2 children).

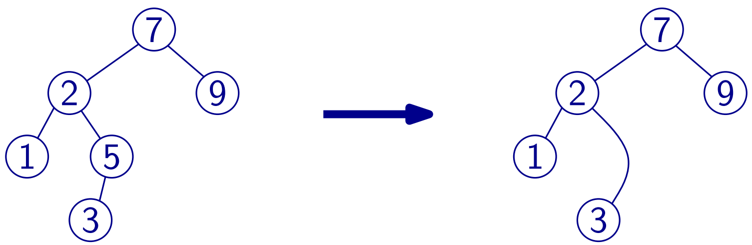
The remove operation

The hardest operation: `remove(key)`.

We use the same strategy as for the `RankTree`:

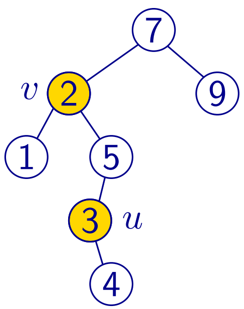
First find the node v containing `key`.
Then there are three cases:

1. Easy case: v is a leaf node.
2. Slightly harder case: v has one child



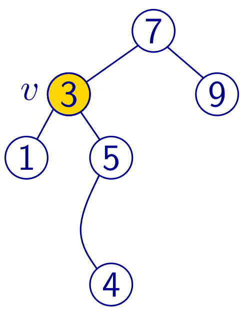
The remove operation

3. If v has two children, then find the leftmost node u in the right subtree of v. Replace the key and value stored at v with the key and value from u. Finally, remove the node u.



The remove operation

3. If v has two children, then find the leftmost node u in the right subtree of v. Replace the key and value stored at v with the key and value from u. Finally, remove the node u.



Binary search tree analysis

The running time of all operations is $O(h)$, where h is the height of the tree.

Unfortunately, we cannot guarantee that the height of the tree remains small. It depends on the order in which the keys are inserted.