24. Trees :: Binary Search Trees :: *remove*()

remove() searches the BST for an element with key *key* and if found, removes that element from the BST. There is a bit of work involved, as removing a node requires "rewiring" part of the tree.

A few examples will help make clear what operations need to be performed. First, consider removing a leaf node:

So the only operation is to set *parent.leftChild* or *parent.rightChild* to null. Now consider removing an interior node that has both a parent and a right child:

So the only operation is to set *parent.rightChild* to *node.rightChild*.

24. Trees :: Binary Search Trees :: *remove*()

Removing the root requires us to replace it with some other node:

One approach would be to find the node *max* containing the maximum key in the left subtree of the root (assuming the left subtree exists), copy *key* and *data* from *max* to the root node, and set the *rightChild* reference of the parent of *max* to *max.leftChild*. If the root does not have a left child, then the root's right child would simply become root.

24. Trees :: Binary Search Trees :: *remove*()

The most problematic operation is removing an interior node that has both a parent and a left child.

The basic procedure is to find the successor node *succ*, i.e., the node with the next smallest key, and copy *data* and *key* from *succ* to the node being removed. If *succ* has a right child, that child must become the left child of the parent of *succ*.