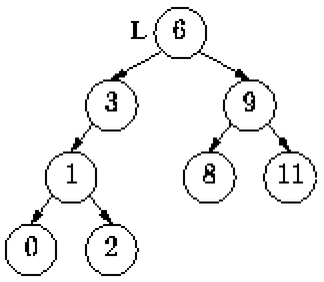
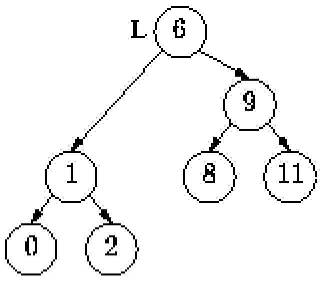
BST deletion algorithm

Of course, if we are trying to delete a leaf, there is no problem. We just delete it and the rest of the tree is exactly as it was, so it is still a BST.

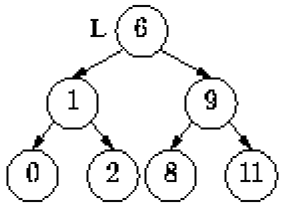
There is another simple situation: suppose the node we're deleting has only one subtree. In the following example, '3' has only 1 subtree.



To delete a node with 1 subtree, we just 'link past' the node, i.e. connect the parent of the node directly to the node's only subtree. This always works (think about it), whether the one subtree is on the left or on the right. Deleting '3' gives us:



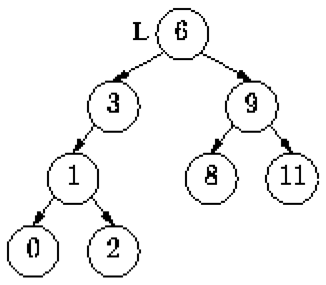
which we normally draw:



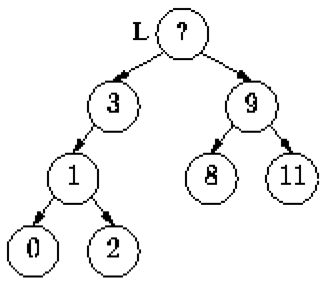
Finally, let us consider the only remaining case: how to delete a node having two subtrees. For example, how to delete '6'? We'd like to do this with minimum amount of work and disruption to the structure of the tree.

The standard solution is based on this idea: we leave the node containing '6' exactly where it is, but we get rid of the value 6 and find another value to store in the '6' node. This value is taken from a node below the '6's node, and it is *that* node that is actually removed from the tree.

So, here is the plan. Starting with:



Erase 6, but keep its node:

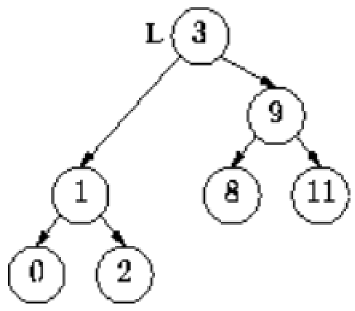


Now, what value can we move into the vacated node and have a binary search tree? Well, here's how to figure it out. If we choose value X, then:

1. everything in the left subtree must be smaller than X.
2. everything in the right subtree must be bigger than X.

Let's suppose we're going to get X from the left subtree. (2) is guaranteed because everything in the left subtree is smaller than everything in the right subtree. What about (1)? If X is coming from the left subtree, (1) says that there is a unique choice for X - we must choose X to be the largest value in the left subtree. In our example, 3 is the largest value in the left subtree. So if we put 3 in the vacated node and delete it from its current position we will have a BST with 6 deleted.

Here it is:



So our general algorithm is: to delete N, if it has two subtrees, replace the value in N with the largest value in its left subtree and then delete the node with the largest value from its left subtree.

Note: The largest value in the left subtree will never have two subtrees. Why? Because if it's the largest value it cannot have a *right* subtree.

Finally, there is nothing special about the left subtree. We could do the same thing with the right subtree: just use the *smallest* value in the right subtree.