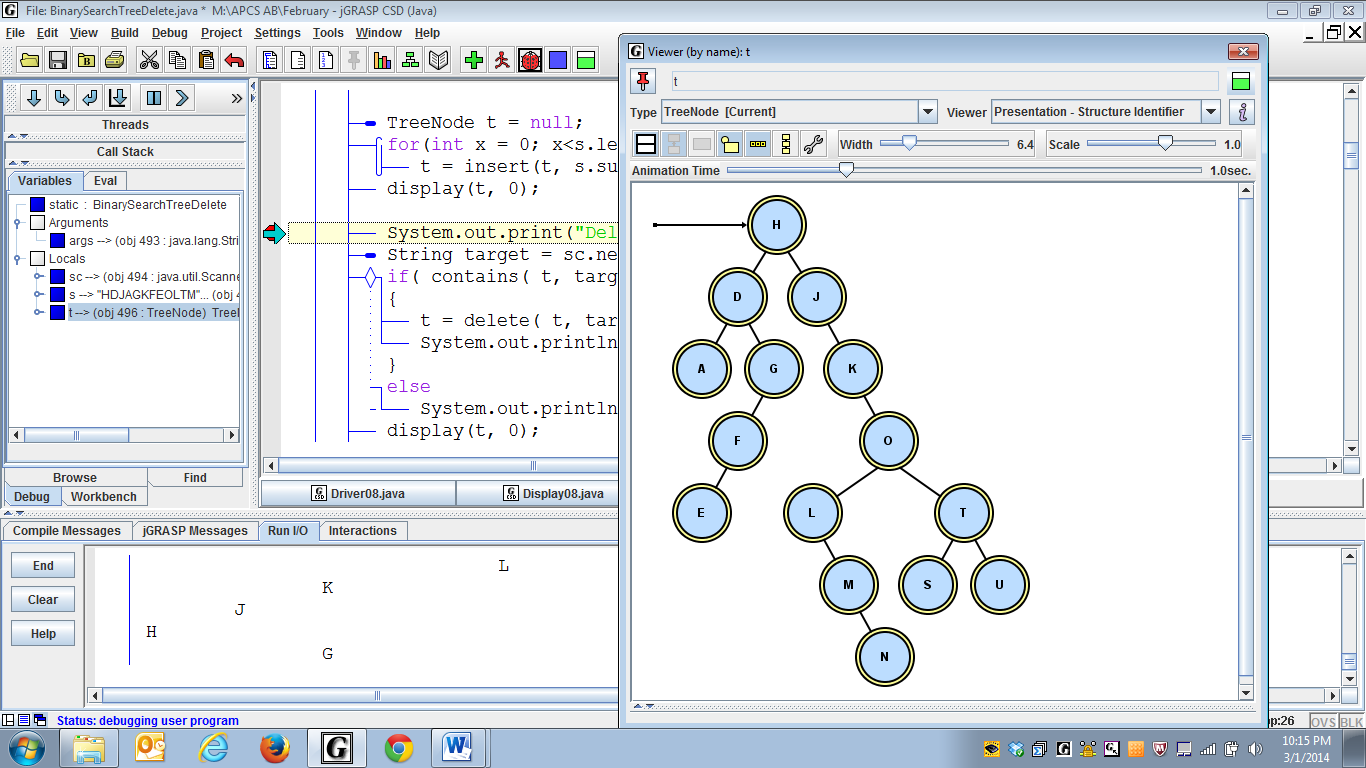
**BST Remove**

1. Is this a binary search tree? Y/N How do you know?



Yes. Every right node is greater than its parent and every left node is less than its parent.

1. When you remove a node, you have to preserve the BST order. Suppose you remove O. What letter(s) should go in its place? N or S

How do you find that letter?

You pick the largest in the left subtree and the smallest in the right smallest

1. Suppose you remove G. What letter(s) should go in its place? F

How do you find that letter? G’s parent points to G’s child

1. Suppose you remove D. What letter(s) should go in its place? A or E

How do you find that letter?

1. Suppose you remove K. What letter(s) should go in its place? O

How do you find that letter?

1. Suppose you remove U. What should go in its place? Null

How do you find that letter?

1. Suppose you remove S. What should go in its place? Null

How do you find that letter?

1. Suppose you remove the root, H. What should go in its place? G

Is deleting the root a special case or not?

Yes, but maybe no. But probably yes

1. What case(s) have we missed?
2. Describe and organize all the different cases:

**Algorithm for Removing a Node from a BST**

Most people keep track of two pointers, a parent and a current.

Find the node to remove, updating parent and current as you go. Then identify which case:

(1) If the node is a leaf, just set its parent pointer to null.

(2) If the node has just one child (left or right), point the parent to current's child (in case the root is also a leaf and we need to remove it just set root to null).

(3) If the node has two children, find the node containing the largest item in the node's left subtree and *replace* (use setValue) the node's object with the largest item. Identify the parent of the largest item. Set the parent’s pointer either to null or to the removed node’s left child.

Return the root.

Let's try it out. For each N in each tree, remove the N and re-draw the pointers:

For each **N** in each BST, remove it properly.

CHALLENGE: do it recursively! The basic idea: if this node contains the element to be removed, find the value that will replace it, setValue to that value, then use a recursive call to remove *the new value* from the appropriate subtree. (For example, in case 3a, we would set the N node to M, then remove M from the subtree starting with F.) Can you work out the details?

Let’s try it again:

Case 1 a: if target is a leaf (but not the root). . .

b: if the target is a leaf and a root…

Case 2 a: if target has one child to right . . .

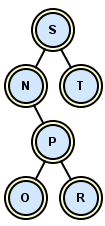
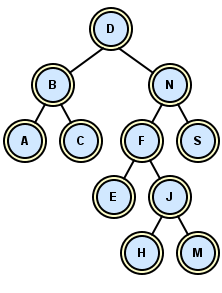
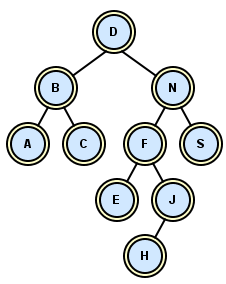
b: if target has one child to left . . .

c & d: if target has one child and is the root…

Case 3: target has two children, even if the target is the root.

a: if the maximum of the left subtree is a leaf…

b, c, d: if the maximum of the left subtree is not a leaf…

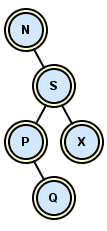
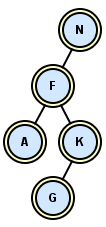
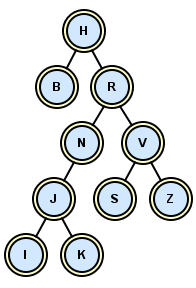
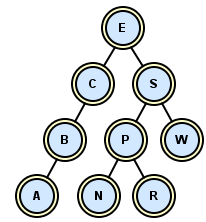


Case 1b)

Case 2a)

Case 2d)

Case 1a)



Case 3c)

root

root

root

root

root

root

root

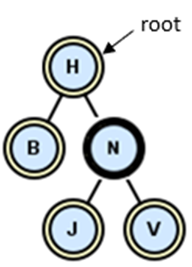
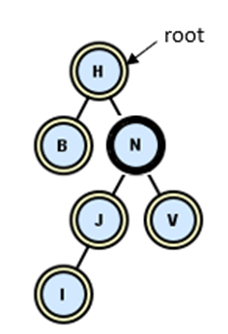
Case 2c)

root

Case 3a)

Case 3b)

Case 2b)



Case 3d)