**Deleting an item from a Binary Search Tree**

Recall, we had 4 cases that we had to consider when deleting an item from a Binary Search Tree (please refer to your notes). Also, we had developed an algorithm called transplant which replaces one subtree, rooted a *u*, with a subtree rooted at *v* – node *u*’s parent becomes node *v*’s parent (again, refer to your notes). With this algorithm, we can now develop an algorithm for deleting an item:

Tree-Delete(T, z) /T is the tree and z is a pointer to the item to delete

if (z.left = NULL) /Case 1

Transplant(T, z, z.right)

else

if (z.right = NULL) /Case 2

Transplant(T, z, z.left)

else

y = Tree-Min(z.right) /z’s successor smallest in right subtree

if (y.parent != z) /Case 4

Transplant(T, y, y.right)

y.right = z.right

y.right.parent = y

endif

Transplant(T, z, y) /Case 3, must be done at end of Case 4

y.left = z.left

y.left.parent = y

endif

endif

To find the successor node:

Tree-Successor(x)

if (x.right != NULL)

return Tree-Min(x.right)

else

y = x.parent

while (y != NULL and x = y.right)

x = y

y = y.parent

endwhile

endif

The two cases for finding the successor node:

1) if the right subtree of x is not empty, then the successor of x is the leftmost item in x’s subtree (minimum)

2) if the right subtree of x is empty and x has successor y, then y is the lowest *ancestor* of x whose left child is also an ancestor of x – go up the tree from x until you encounter a node that is the left child of its parent

Finding the predecessor is the opposite of finding the successor.