Quiz 4 Report

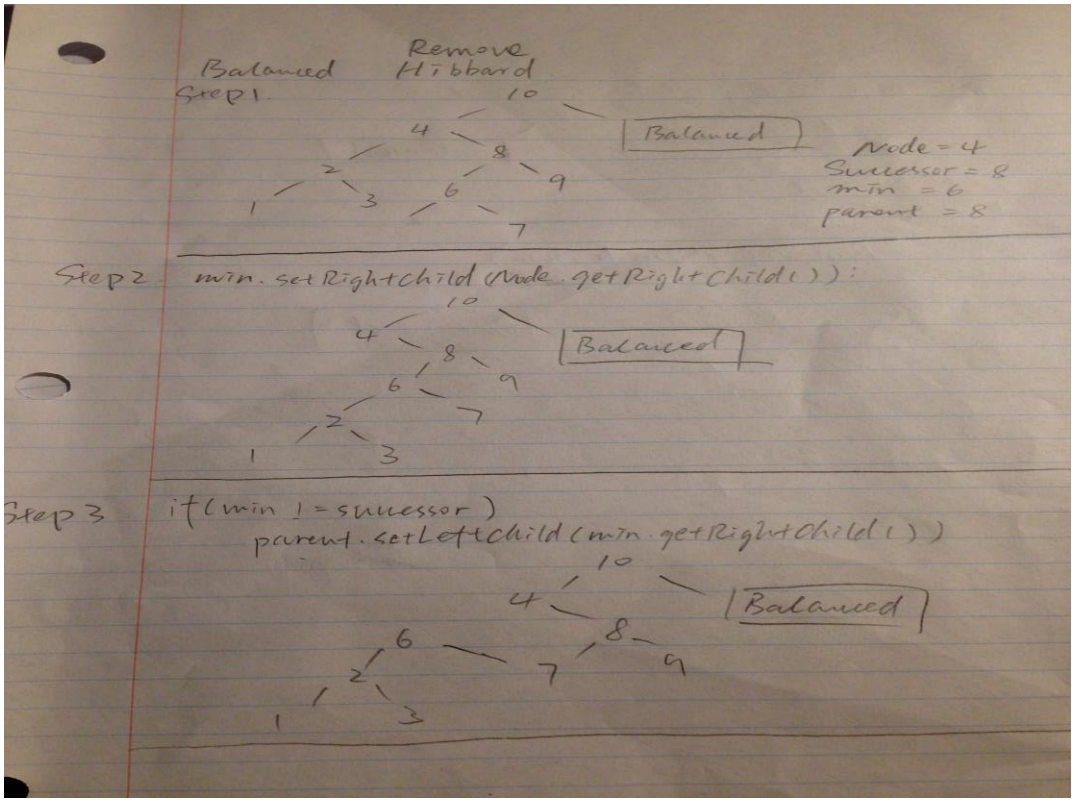
Qingyuan Zhang

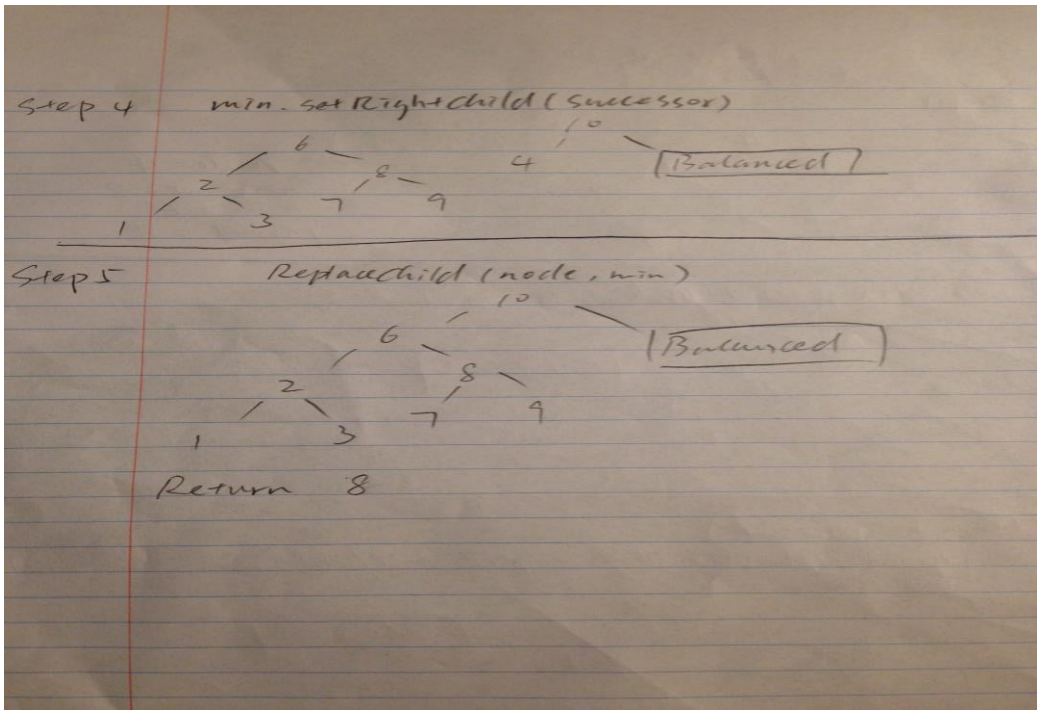
The remove method of AbstractBalanceBinarySearchTree is to delete the key in the Balance Binary Search Tree and keep the tree in a balance order. In Binary search trees, we can easily delete a node that has only one child or no child. If the key we want to delete is stored at a node has two children. We can transform the binary search tree and move the key to a place that we can easily delete with Hibbard remove method.

The remove method first run the method findNode in AbstractBinarySearchTree. It search the key whether it is in the tree or not. The remove method start to search the node with the root of the binary search tree. If it is larger than the key in the node it will recursively search its right child until there is no right child. While if it is smaller than the key in the node it will search its left child.

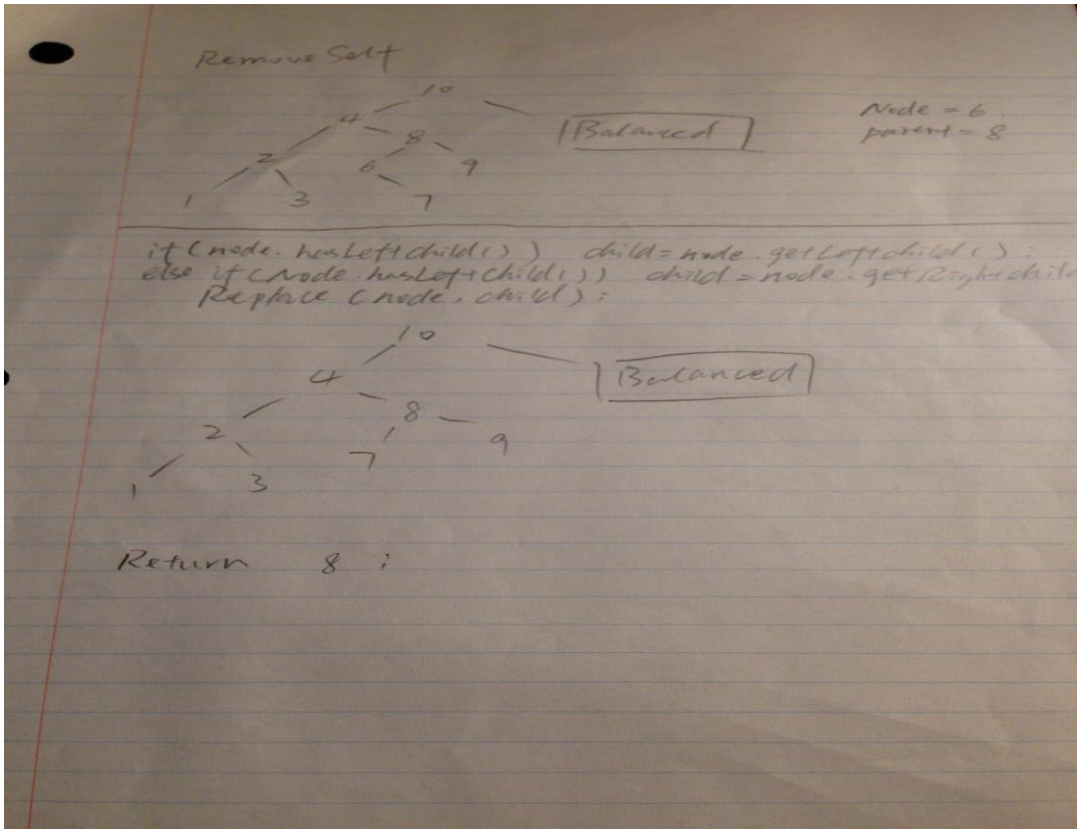
If there is a node contain the key that we search, we go to an if statement to assign the “lowest”. If the node that contain the key has both its left and right children, we call remove Hibbard method. If the node does not have both children, we call removeself method instead. The “lowest” is the parent of the node of our deletion or the parent of our keys inorder predecessor. It is the lowest height node that its subtree has been changed.

RemoveHibbard method perform as follow graph:





Hibbardremove is the method that remove the key which do not have external child. Selfremove is easier since it is to remove the key which contain the external child (only have one child or no child)

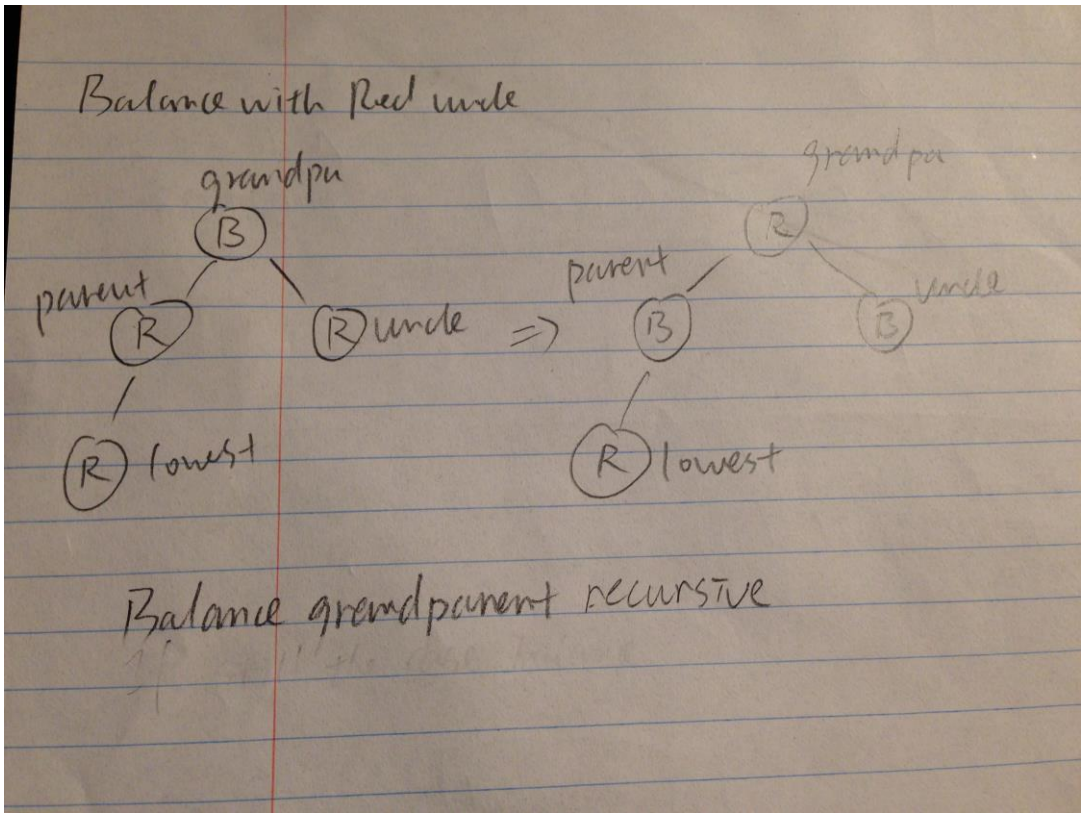


From above two remove method we get the “lowest”, which is the lowest height node whose subtree is upgraded. When we run balance method with AVL, our main focus is to rotate according to the balance factor of the “lowest”.

Since we remove the node in the balanced binary search tree, the balance factor of “lowest” cannot exceed |2|. In AVL balance method, we check the balance factor of lowest. If it is equal to 2, then the tree under lowest has more height in left side, we rotate it to right. While. If it is equal to -2, then the right part has more height, we rotate left. In other case, which means factor = (-1,0,1) we keep check its parent’s balanced factor and do rotate as above.

As for the Red and Black method, the balance method is more complicated. Balancing the trees during removal from red-black tree requires considering several cases. Deleting a black element from the tree creates the possibility that some path in the tree has too few black nodes, breaking the law the every path from a given node to any of is descendant leaves contain the same number of black nodes or every red node must have two black child nodes.

If our “lowest” is red and its parent is also red, we have to do balance to the red and black tree. In the red and black balance method, we have to check the uncle of “lowest” is black or red. Uncle has the same parent as parent of lowest. If the uncle is red, we just simply change the color and recursively check whether the grandparent satisfy the red and black tree requirement.



If the uncle of “lowest” is black, there are several case to be consider. Here is one case that parent is the left child of grandparent and lowest is the right child of parent. For other case, we just simply check whether it needs to rotate or not and rotate correctly. After the first rotation, we also needs to check the situation of grandparent and might do rotate again in order to get balanced.