

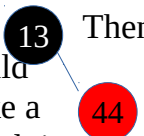
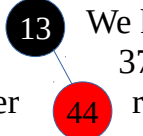
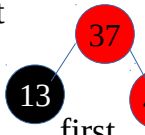
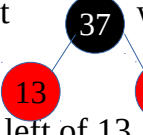
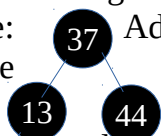


Assignment 8

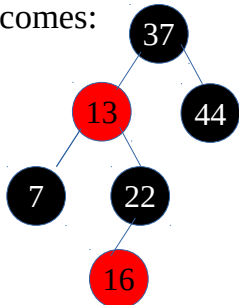
Jovan Shandro

Problem 1.

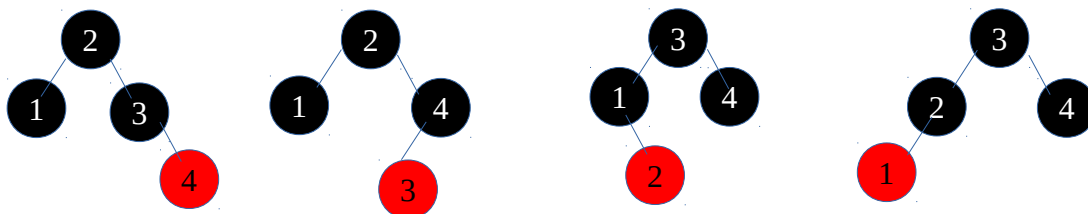
a) (I am not including the null children as they can be easily pictured in the mind) We first insert 13 as a single red node.  Since it is root, it is recolored to black. So tree is now.  Next, 44 is entered as a red node on the right of the root node 13. Tree is now:  Then adding 37 initially as red node we get:  We have a red node with a red child so it is not a valid tree. Since the node 37 has a black uncle so we make a left rotation and get  And after recoloring we get  which is now a valid tree.

Adding then 7, it is first inserted as a red node in the left of 13. Since now 13 is red and has a red child the tree is not valid. The uncle of node 7 is red (44) so we recolor the parent, grandparent and uncle and since after recoloring the root node becomes red it is again recolored to black so we now get the tree:  Adding 22 then, it is first added as a red node in the right side of the node 13 and the tree we get is a valid tree.

Now in the end adding 16, it is inserted as a red node in the left of node 22 and since 22 is a red node that has a red child, the tree is not valid. The node 16 has a red uncle (7), so we do a recoloring of the parent, grandparent and uncle nodes. The tree becomes:



b) To find all possible valid red black trees we can get with 1, 2, 3, 4, we first start considering the root node. If the root node is either 1 or 4, then all other nodes will be on one side and since one the the children of the root node is Null, no matter how we color the other three there is no possible way to have only one black node out of 2 3 and the other one as every red node must have black children. So the root node can only be 2 or 3. There are 4 possible configurations and each can be colored in a single possible way to guarantee that the tree is valid. The configurations are:



c) (Bonus) I will use induction to prove that there will be at least one red node.

As a base case take $n = 2$. In this case we have the root node and another node connected to it. Since by our algorithm the nodes are always red when they are first inserted, so the second node is red when it gets first inserted, but since the tree is a valid red black tree, then no color change happens so the child is red. Thus the tree has a red node

Induction step: Assume that for a red black tree with n nodes, there is at least one red node. Now we prove that for a red black tree with $n+1$ nodes there is also at least one red node, and this would end the proof. Since all $n+1$ are inserted in turns on an empty tree, taking into consideration the last $((n+1)^{\text{th}})$ node that was inserted, we have two subcases.

First case is when this node is inserted as a child of a black node. In this case the tree still remains valid so no need to re-change any colors of the tree will have at least one red node.

Second case is when this node is when the node is inserted as a child of a red node.

As we have discussed also in the pseudocode, the insert fix up can be one of 3 cases for a node that is a child of a red parent node.

i) this node remains red after recoloring the parent, grandparent and uncle nodes. So in this case again we have at least one red node

ii) the node remains red after rotating the parent about the grandparent node. After then recoloring the parent and grandparent, the node has still the same color so we have at least one red node.

iii) the parent of our node remains red after a left-right or right-left rotation of our node about the parent and grandparent nodes. After recoloring our node and the node that previously was a grandparent, the previously parent node remains red, so we still have at least one red node.

So in all cases the tree will have at least one red node, so by induction, this ends the proof.

REFERENCES

“Introduction to algorithms” book.