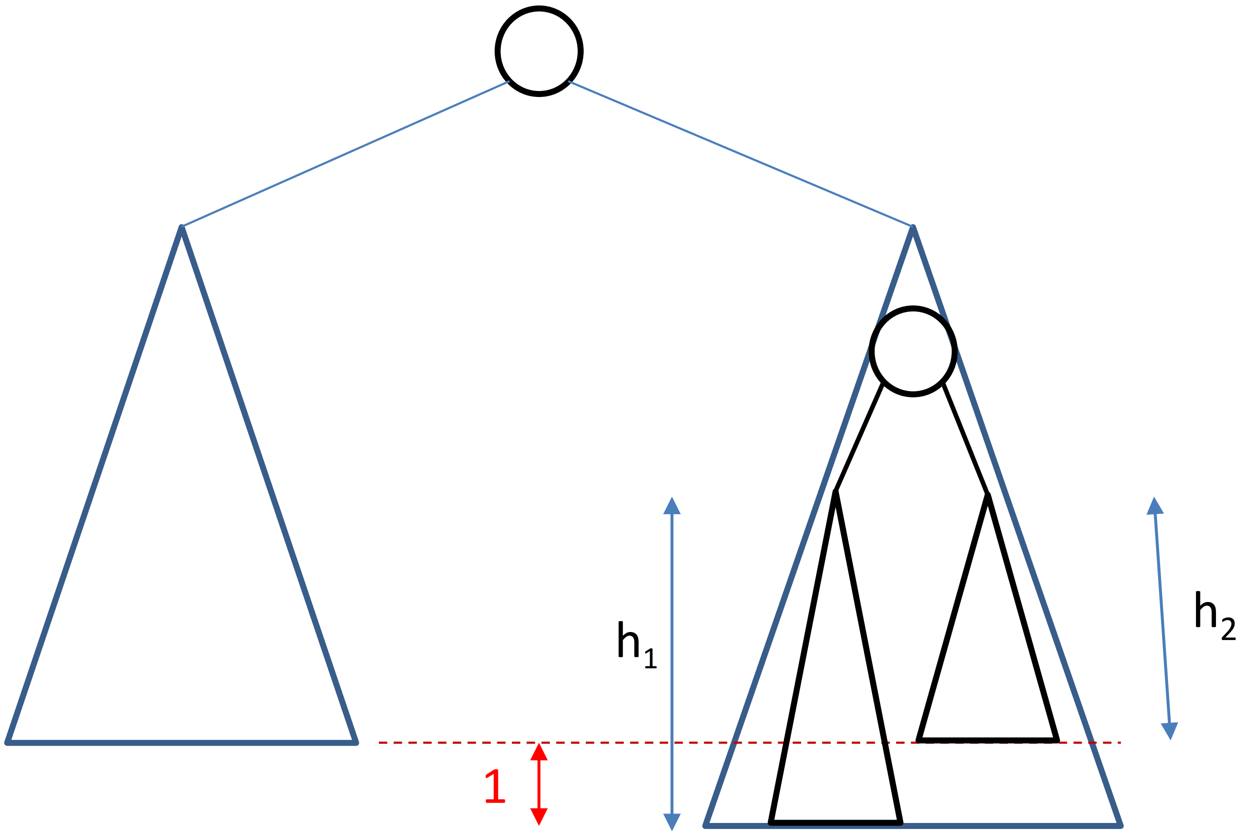
AVL Trees

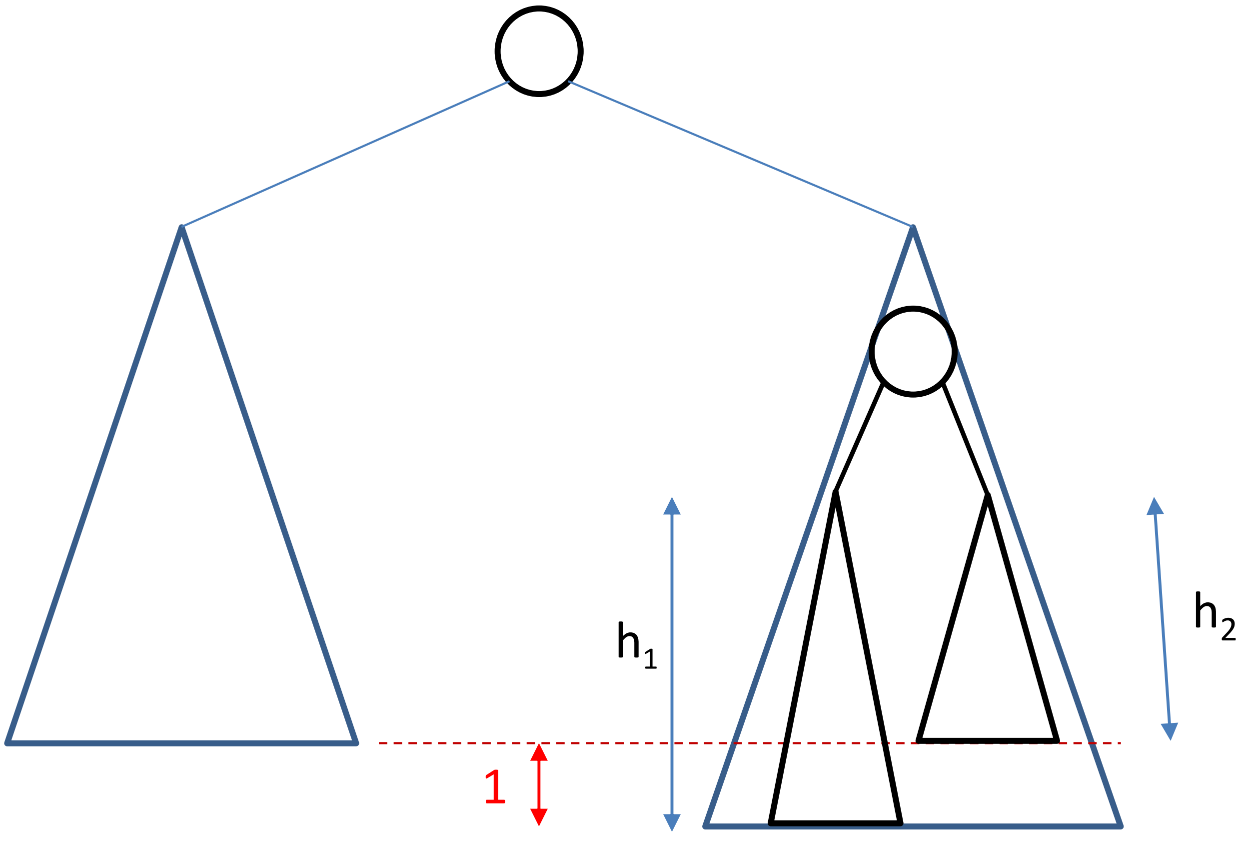
An AVL tree is a binary search tree in which for every internal node the heights of its two subtrees differ by at most 1.



AVL Trees

An AVL tree is a binary search tree in which for every internal node the heights of its two subtrees differ by at most 1.

What is the maximum height of an AVL tree?



Ordered Dictionary Implemented with Binary Search Trees

Operations

get(k)
smallest()
largest()
put(k,d)
remove(k)
successor(k)
predecessor(k)

O(height of tree) time complexity

What is the Maximum Height of an AVL Tree?

Let n(h) = minimum number of nodes in an AVL tree of height h.

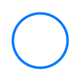
base case

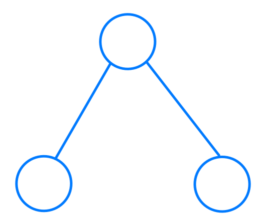
$n(0) = 1$

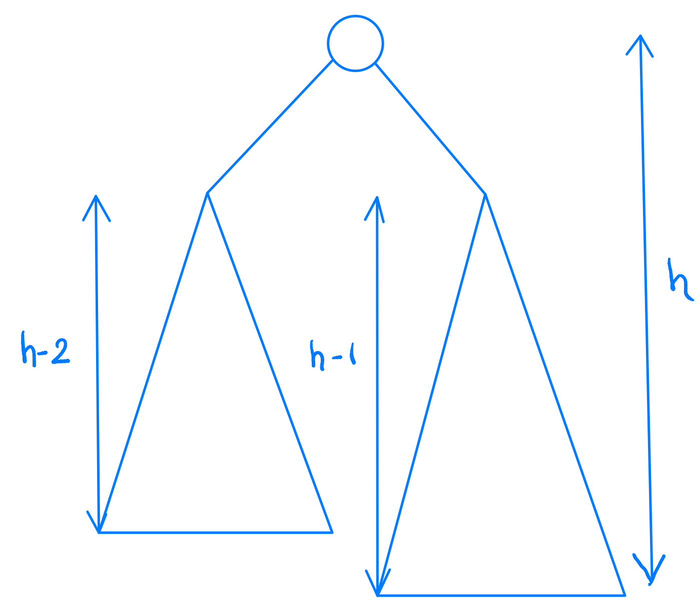
$n(1) = 3$

$n(h) = 1 + n(h-2) + n(h-1)$

$n(h) > 1 + 2n(h-2)$







Assume h is even

$n(0) = 1$

$n(h) > 1 + 2n(h-2)\ ,\ h > 0$

$2n(h-2) > 2[1+2n(h-4)]\ ,\ h > 0.$

$> 2 + 4n(h-4)$

$2^2 n(h-2\times 2) > 2^2(1+2n(h-2\times 3)) = 2^2 + 2^3 n(h-2\times 3)\ .$

$\vdots$

$2^x n(h-2x) > 2^x + 2^{x+1} n(h-2(x+1))$

$n(h) > 2^0 + 2^1 + \cdots + 2^x + 2^{x+1}.$

$= \sum_{i=0}^{i=x+1} 2^i = 2\frac{1-2^{x+1}}{1-2} = 2^{x+2} - 1.$

$n > 2^{i+2} - 1.\quad i = \frac{h}{2} - 1$

$n > 2^{\frac{h}{2}+1} - 1.$

$\log_2(n+1) > \frac{h}{2} + 1.$

$h < 2\log_2(n+1) - 2.$

Ordered Dictionary Implemented with AVL Trees

Operations

get(k)
smallest()
largest()
successor(k)
predecessor(k)
put(k,d)
remove(k)

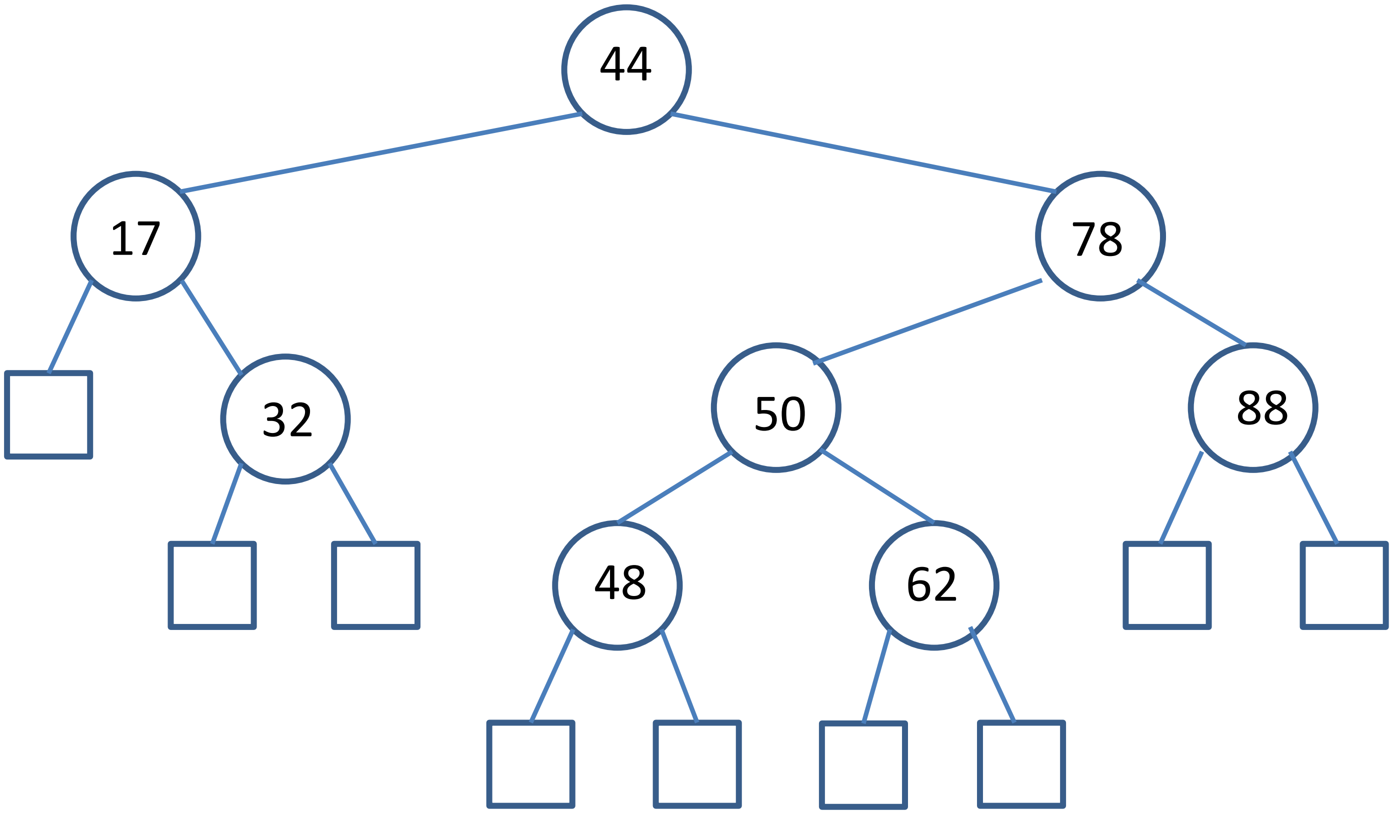
get(k), smallest(), largest(), successor(k), predecessor(k): O(height of tree) = O(log n)

put(k,d), remove(k): O(height of tree) = O(log n)?

=> same as methods in BST.

AVL Tree

get (62)



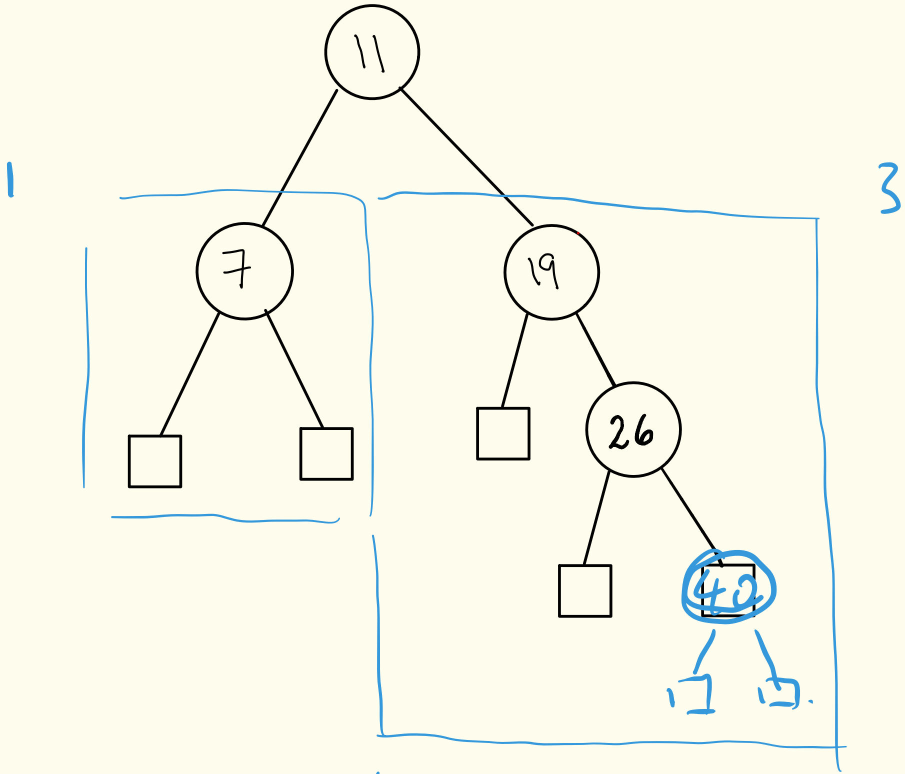
Ordered Dictionary Implemented with AVL Trees

Operations

Query operations: $\left.\begin{array}{l}\text{get(k)}\\ \text{smallest()}\\ \text{largest()}\\ \text{successor(k)}\\ \text{predecessor(k)}\end{array}\right\}$ O(height of tree) = O(log n)

$\left.\begin{array}{l}\text{put(k,d)}\\ \text{remove(k)}\end{array}\right\}$ O(height of tree) = O(log n)?

put 40



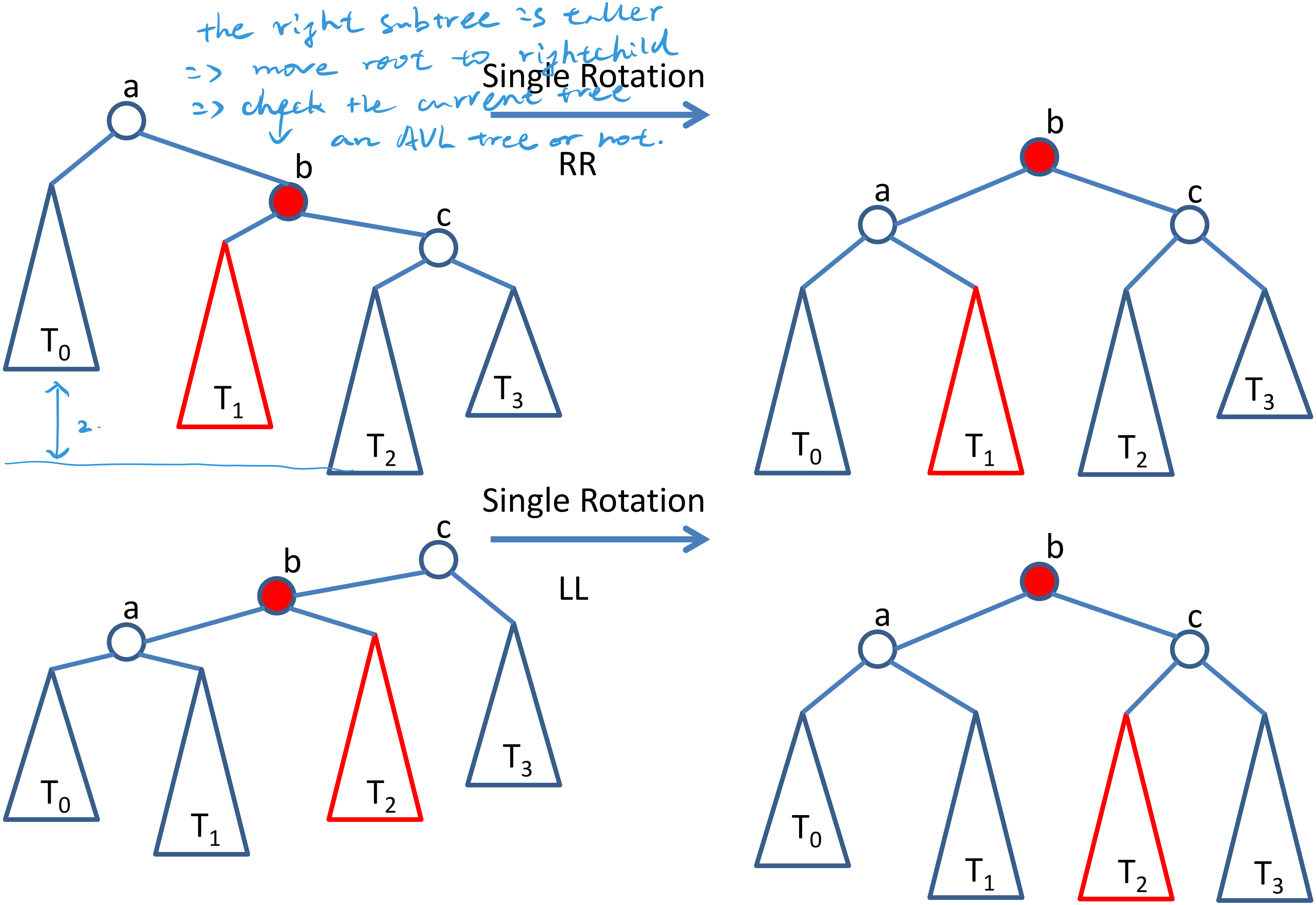
Fix :

after adding 40, the tree is no longer an AVL tree.

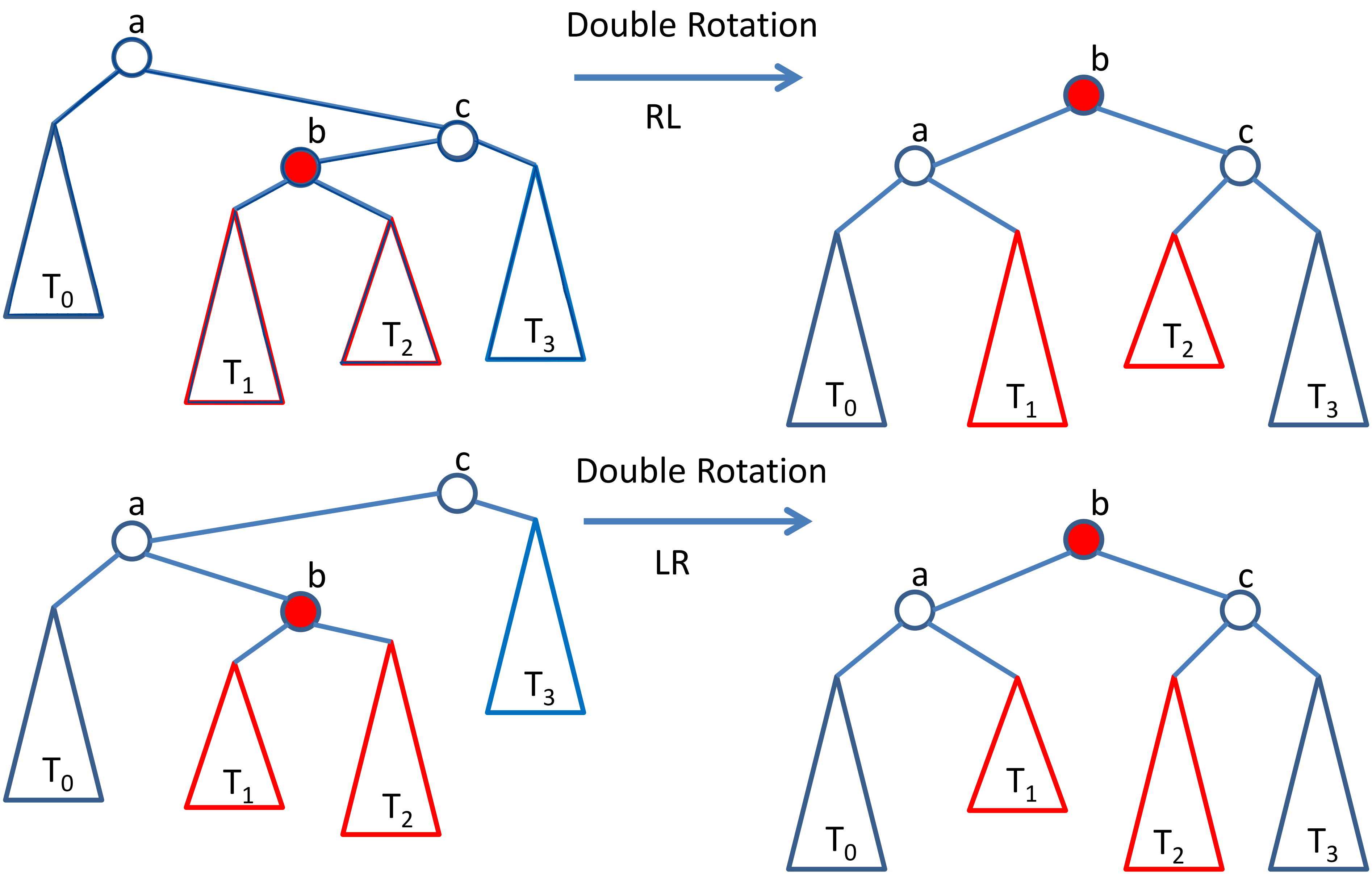
Re-Balancing AVL Trees

To re-balance an AVL tree we always rebalance the smallest un-balanced subtree.

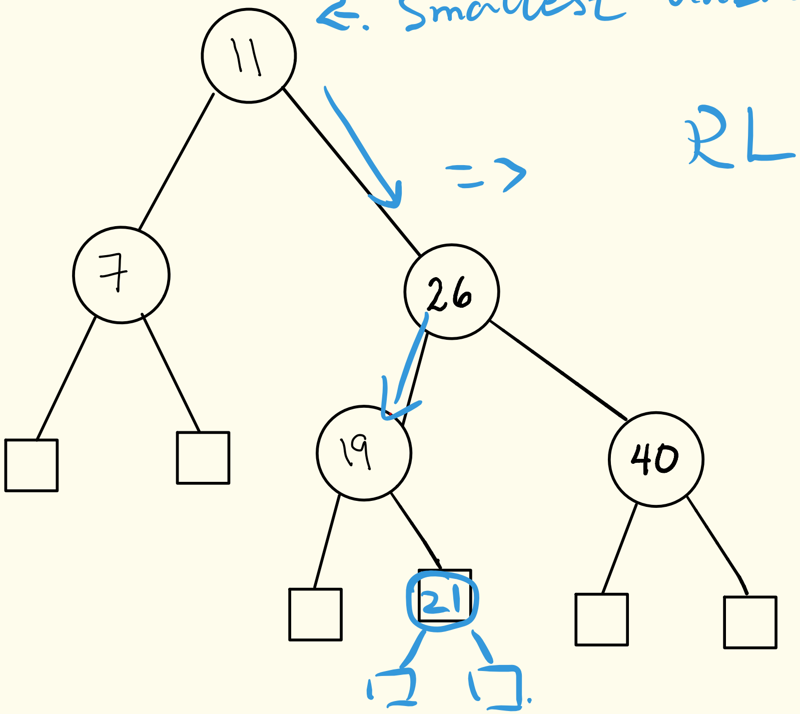
Single Rotations



Double Rotations



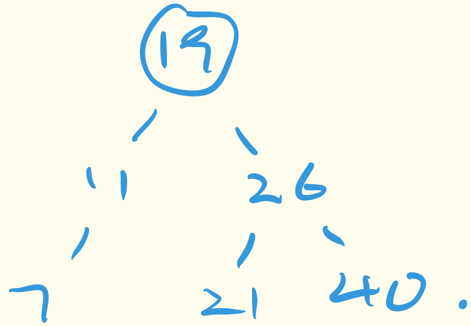
put 21, 5



← Smallest unbalanced tree

=>

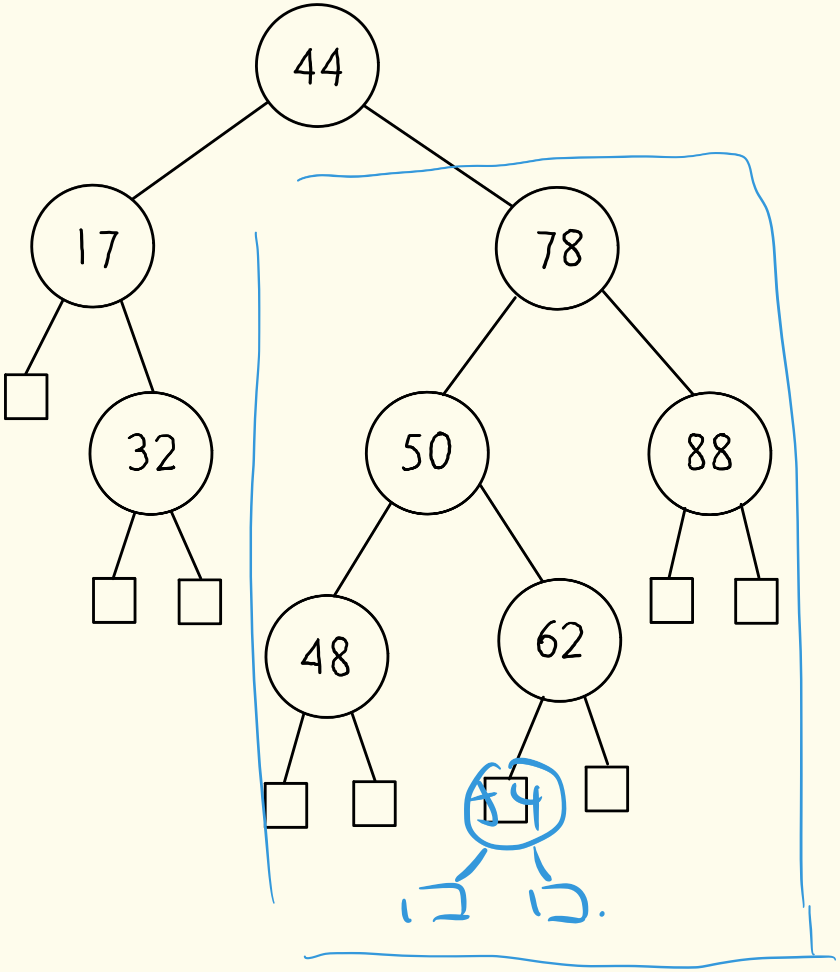
RL double rotation.

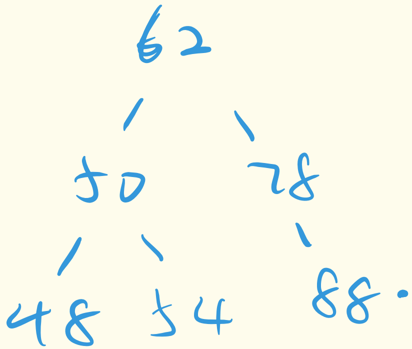


Re-Balancing AVL Trees

If the tree becomes unbalanced due to an insertion ONE rotation will re-balance the tree.

put(54)





Algorithm putAVL (r, k, data)

In: Root r of an AVL tree, record (k,data)

Out: {Insert (k,data) and re-balance if needed}

$O(\text{height}) = O(\log n)$ $p \leftarrow put(r, k, data)$ // Algorithm to add information
// to a binary search tree

locate unbalance pare of the tree $O(c_1 \times \log n)$.

while ($p \neq null$) and (difference in height between subtree of p is at most 1) do

$p \leftarrow p.parent$

if $p \neq null$ then

Rebalance the tree rooted at p by one rotation.

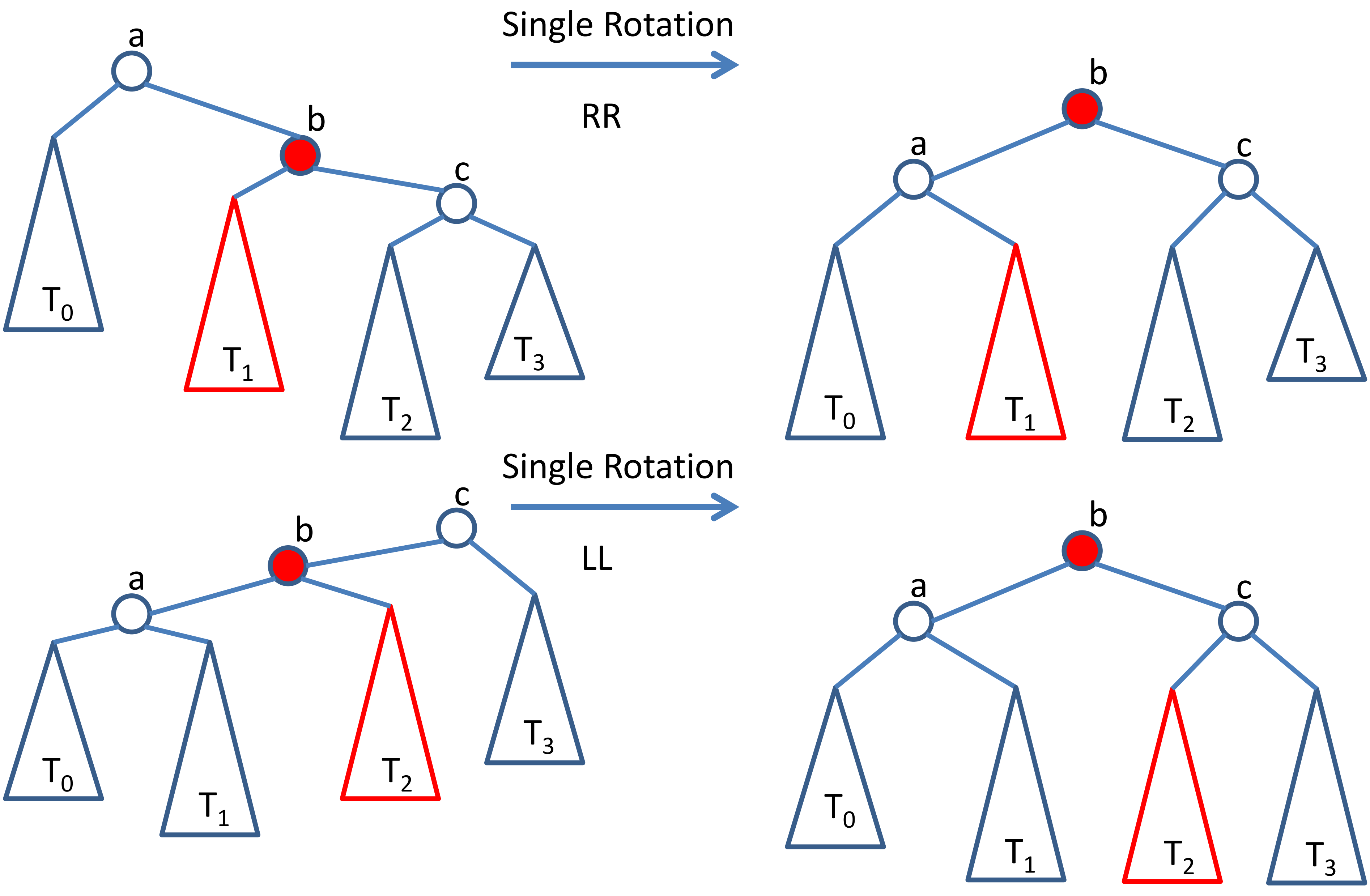
the operation changes at most 3 links in single rotation.

4 in double rotation

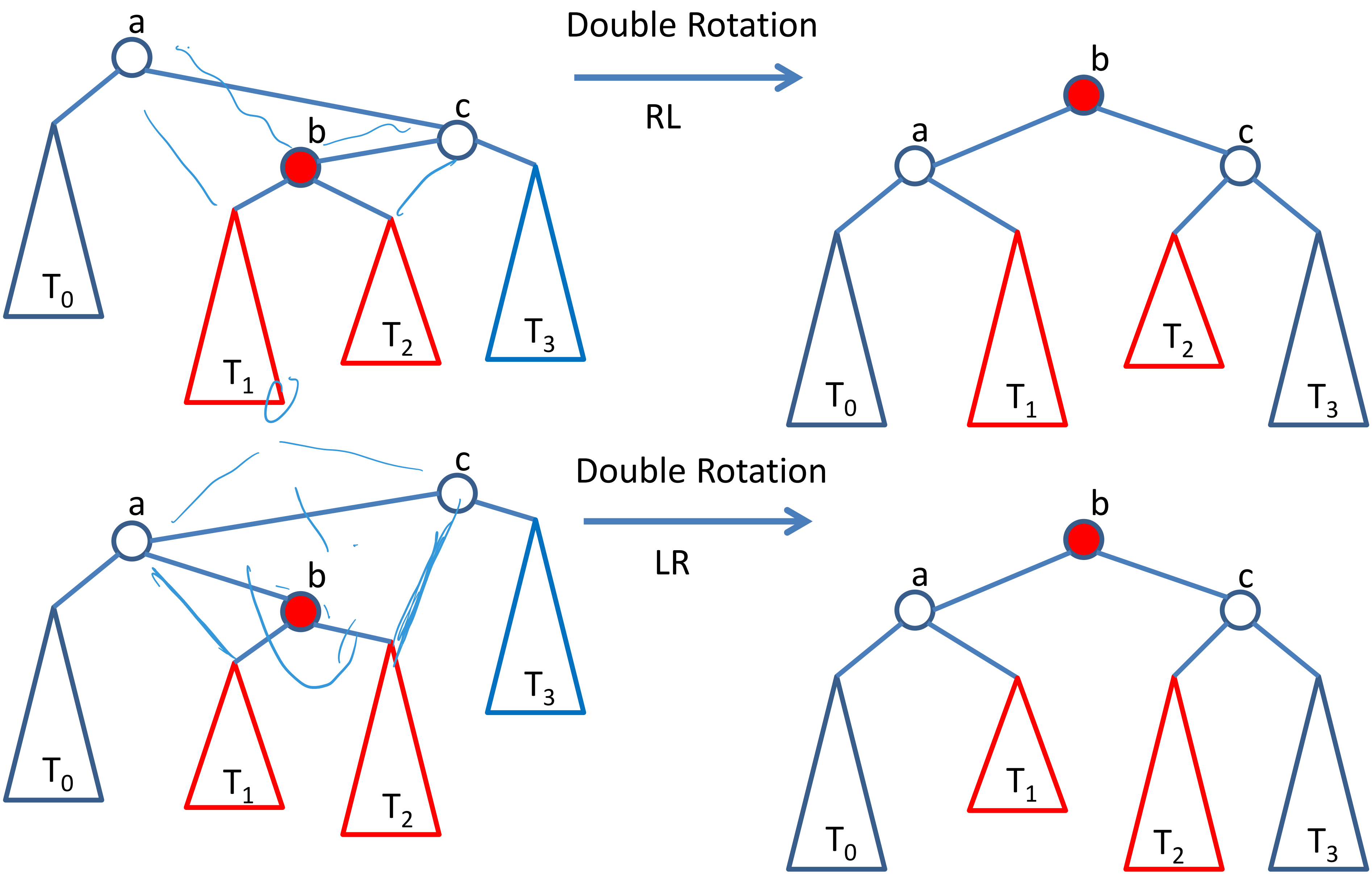
$\log n + \log n + C$.

$f(n) = O(\log n)$.

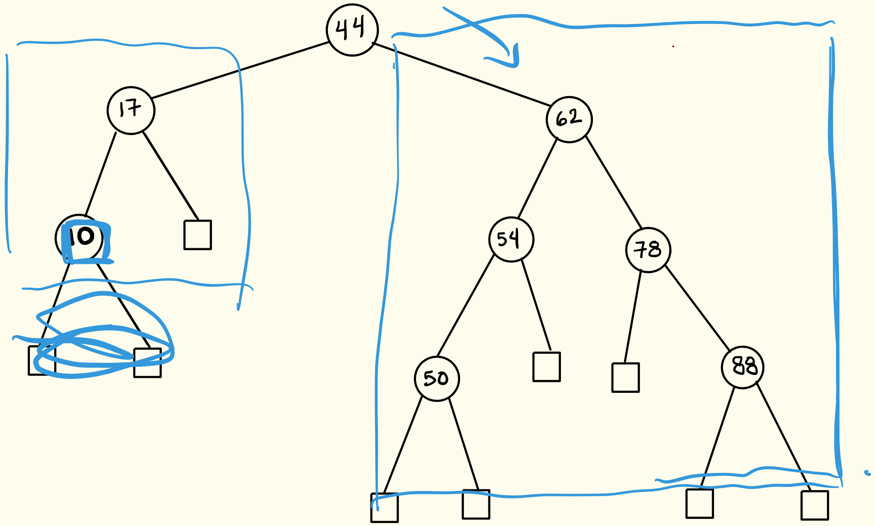
Single Rotations Complexity



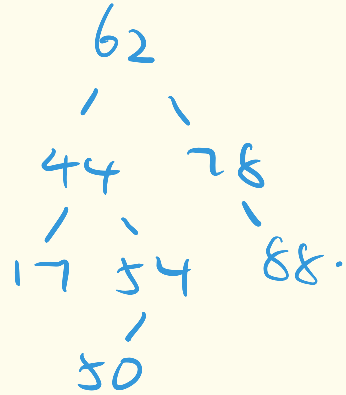
Double Rotations Complexity



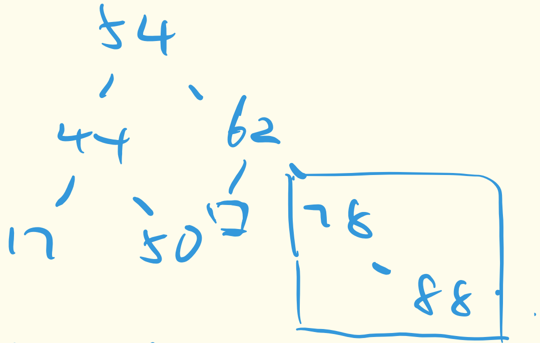
remove(10)



R.



RL



It is not an AVL tree!

Re-Balancing AVL Trees

When a single and a double rotation can be applied to an un-balanced subtree the single rotation always re-balances the subtree.

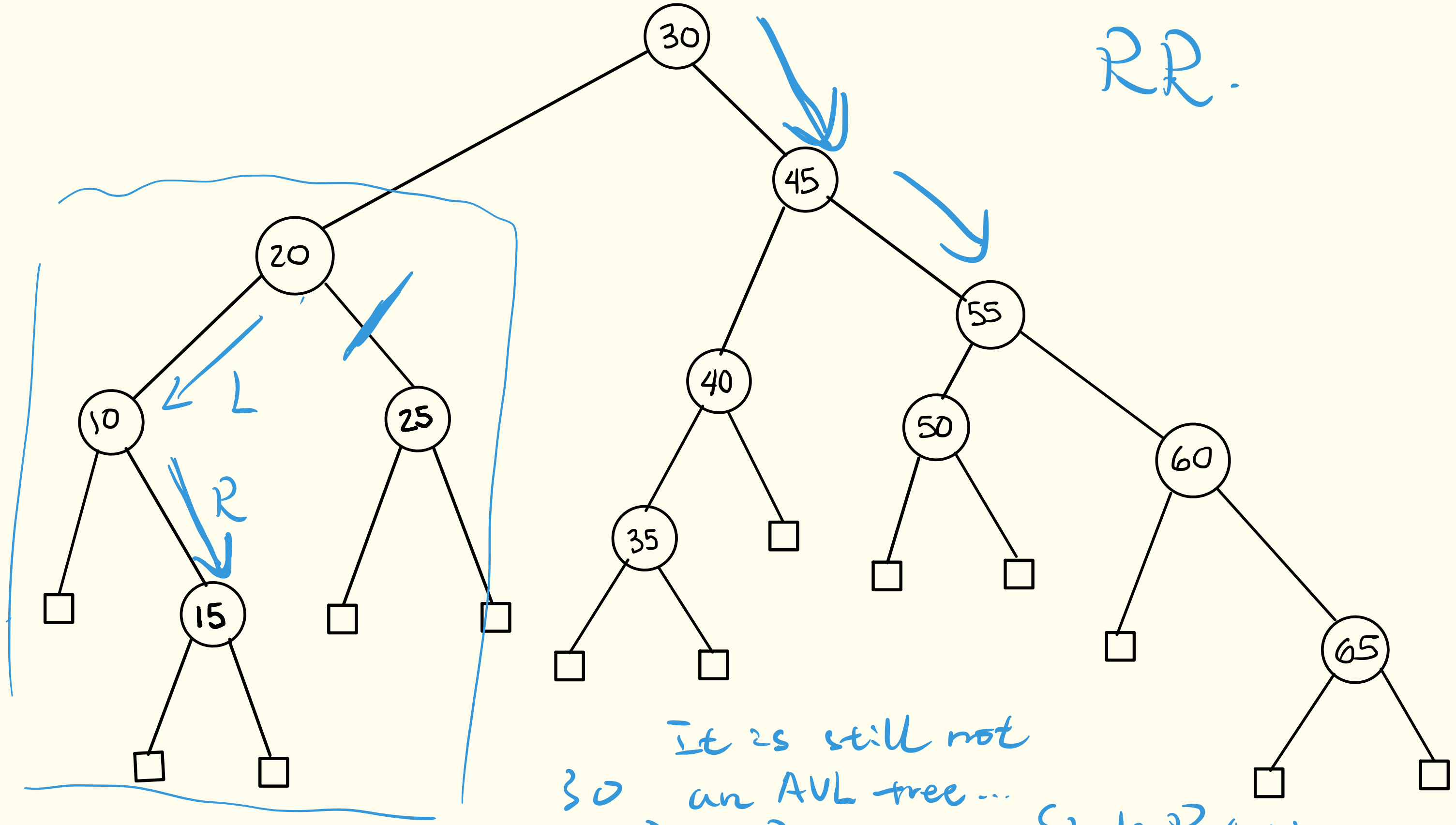
Re-Balancing AVL Trees

If the tree becomes unbalanced due to a removal SEVERAL rotations might be needed to re-balance the tree.

Find the smallest subtree.

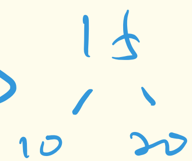
remove(25)

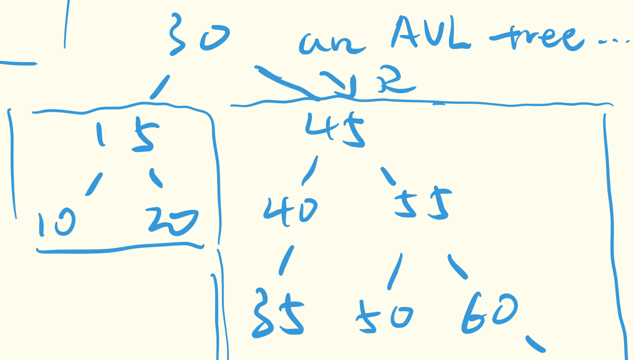
RR.



It is still not an AVL tree...

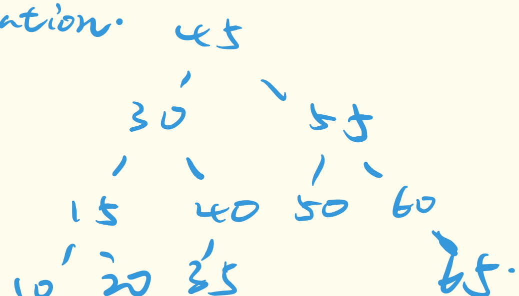
LR =>





Single Rotation.

RR =>



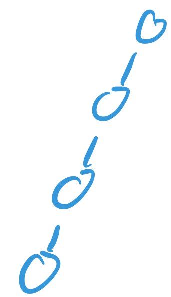
65

Algorithm removeAVL (r, k)

In: Root r of an AVL tree, key k to remove

Out: {Remove k and re-balance if needed}

$p \leftarrow$ remove(r,k) } Algorithm for binary search trees

max : height => 

$f(n) = O(\log n)$

while ($p \neq$ null) **do {** : $O(\log n)$.

  if the two subtrees of p differ in height > 1 **then**

    rebalance subtree rooted at p by performing

    appropriate rotation

  p = parent of p

}

Ordered Dictionary Implemented with AVL Trees

Operations

get(k)
smallest()
largest()
successor(k)
predecessor(k)

O(height of tree) = O(log n)

put(k,d)
remove(k)

O(height of tree) = O(log n)