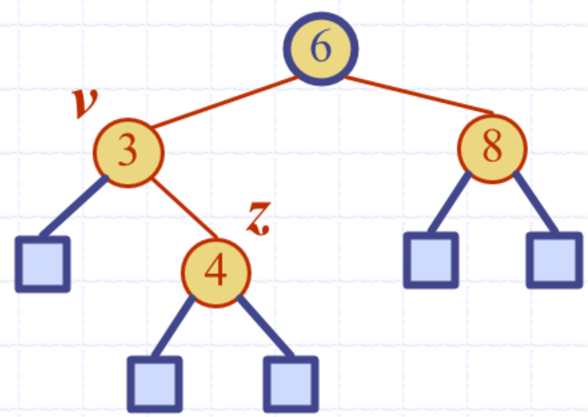
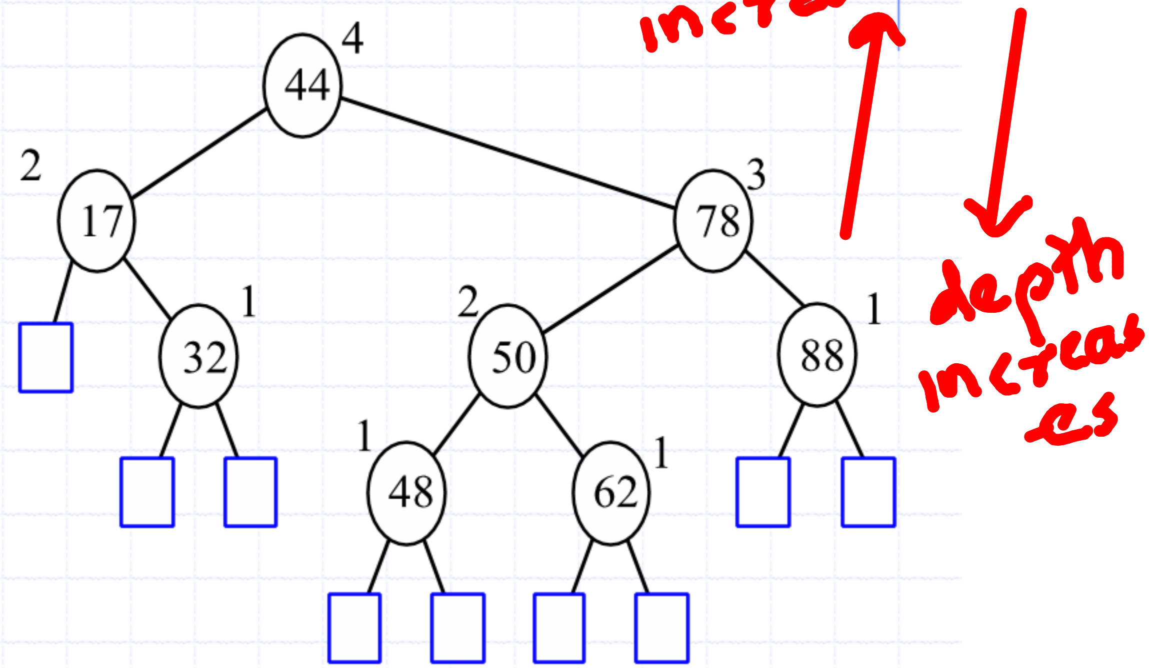
AVL Trees

(An effort to realise $h = O(\log n)$ in BST)



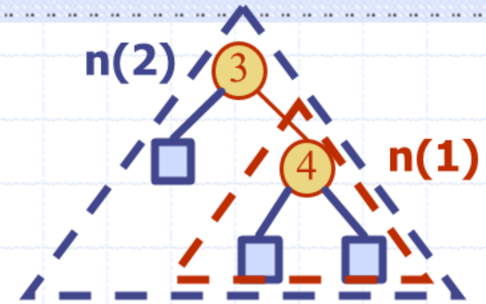
AVL Tree Definition (§ 9.2)

- **AVL trees are balanced.**
- An AVL Tree is a ***binary search tree*** such that for every internal node v of T, the *heights of the children of v can differ by at most 1.*



An example of an AVL tree where the heights are shown next to the nodes:

Height of an AVL Tree



- **Fact**: The ***height*** of an AVL tree storing n keys is O(log n).
- **Proof**: Let us bound **n(h):** the minimum number of internal nodes of an AVL tree of height h.

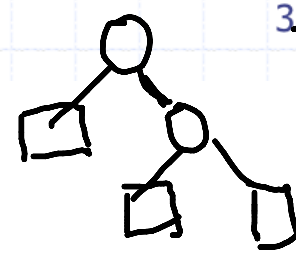
lower bnd

- We easily see that n(1) = 1 and n(2) = 2
- For n > 2, an AVL tree of height h contains the root node, one AVL subtree of height h-1 and another of height h-2.
- That is, n(h) = 1 + n(h-1) + n(h-2)

(∵ minimum)

- Knowing n(h-1) > n(h-2), we get n(h) > 2n(h-2). So
  n(h) > 2n(h-2), n(h) > 4n(h-4), n(h) > 8n(n-6), … (by induction),
  n(h) > 2^in(h-2i)
- Solving the base case we get: n(h) > $2^{h/2}$ $> 2^{h/2 - 1/2} = (\sqrt{2})^{h-1}$
- Taking logarithms: h < 2log n(h)
- Thus the height of an AVL tree is O(log n)

what if $\sqrt{2}$ is replaced with 'c'



n(2) = 2

Worst case:

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

$n(h-1) = n(h-2) + n(h-3) + 1$

$n(h-2) = n(h-3) + n(h-4) + 1$

$\Big\}$ $n(h) > 2n(h-2)$

$> 4n(h-4)$

$\vdots$

$\geq \underbrace{2^{i} n(h-2i)}$

$n(h) > 2^{h/2} \Leftarrow$ if $h-2i=2$ i.e $i = \frac{h}{2} - 1$, $n(h-2i) = 2$

n = # of nodes, in worst case $n = n(h)$

i.e $n \geq n(h) > 2^{h/2} \Rightarrow h < 2\log_2 n$

$h = O(\log_2 n)$

For BST: $h \leq n$. For AVL trees: $h \leq 2\log_2 n$

Recall that for binary trees: $h \geq \log_2(n+1)$

Another analysis for

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

Claim: $n(h) \geq c^{h-1}$ (2) (for some c we need to determine)

Base case(s): $n(1) = 1 \geq 1$

$n(2) = 2 \geq c$

Proof by induction: Assume (1) holds for $k = 1 \ldots h-1$. Now invoking (2)

$$n(h) \geq c^{h-2} + c^{h-1} + 1 > c^{h-2} + c^{h-1}$$

$c^h > c^{h-2} + c^{h-1}$ if $c \in \left[\frac{1+\sqrt{5}}{2}, 2\right]$

$$\left\{ \begin{array}{l} (c^2 - c - 1 > 0) \\ (c - r_1)(c - r_2) \\ > 0 \end{array} \right.$$

If $n \geq n(h) \geq c^{h-1}$

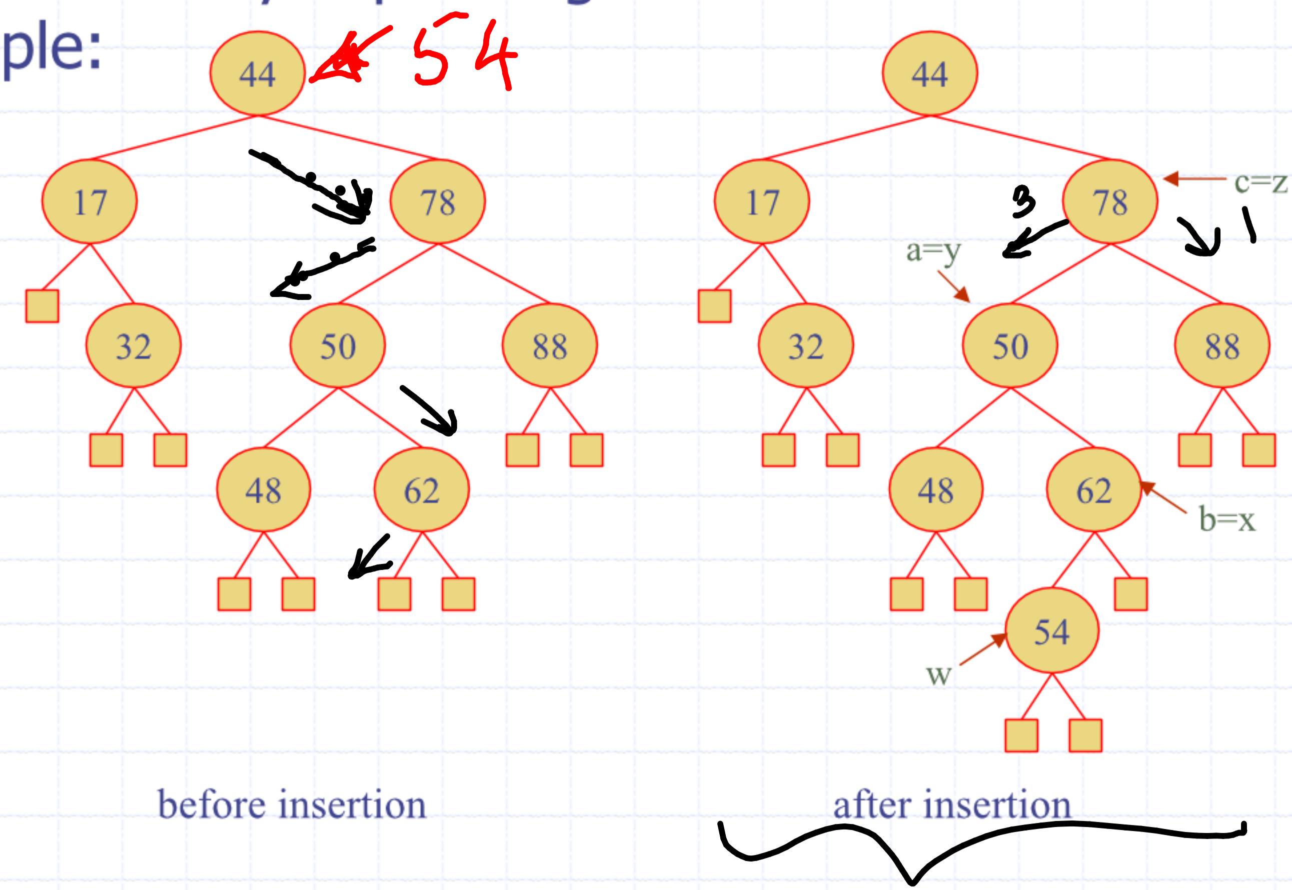
then $h \leq \log_c [n] + 1$

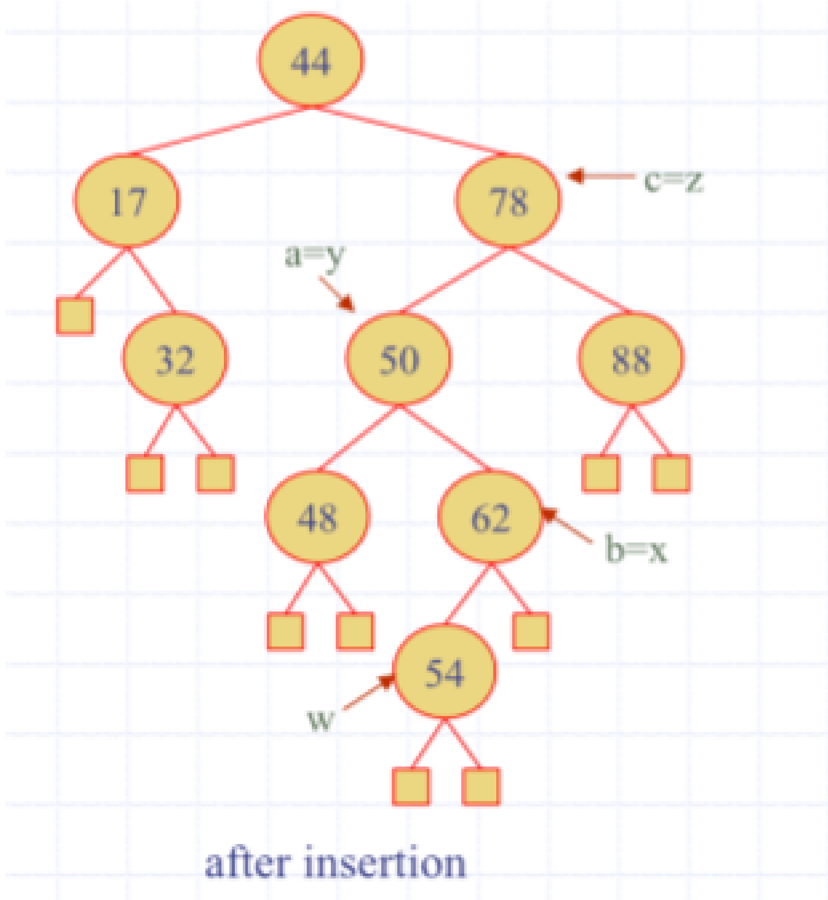
$$= \log_c 2 * \log_2 [n] + 1$$

i.e $h = O(\log_2 n)$

Insertion in an AVL Tree

- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example:



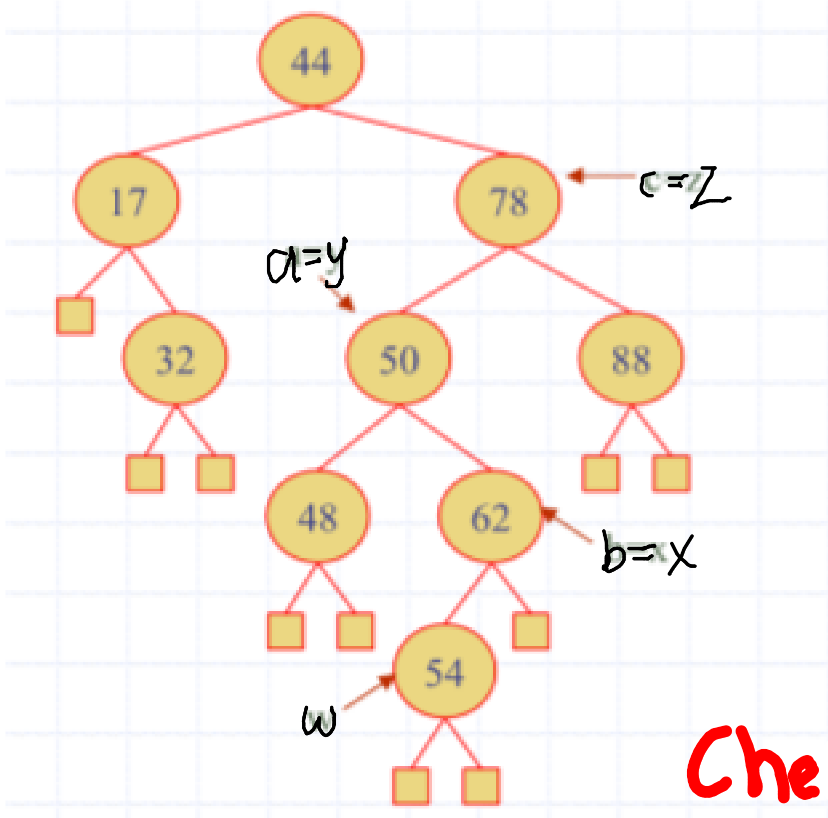


THINK

Q: What will be the "general" procedure for "rebalancing" an imbalanced tree following an insertion?

[Hint: Look at what you would do in this example in terms of x, y, z / a, b, c

WRITE DOWN ALL THE STEPS OF YOUR GENERAL PROCEDURE



after insertion

Q: What will be the "general" procedure for "rebalancing" an imbalanced tree following an insertion?

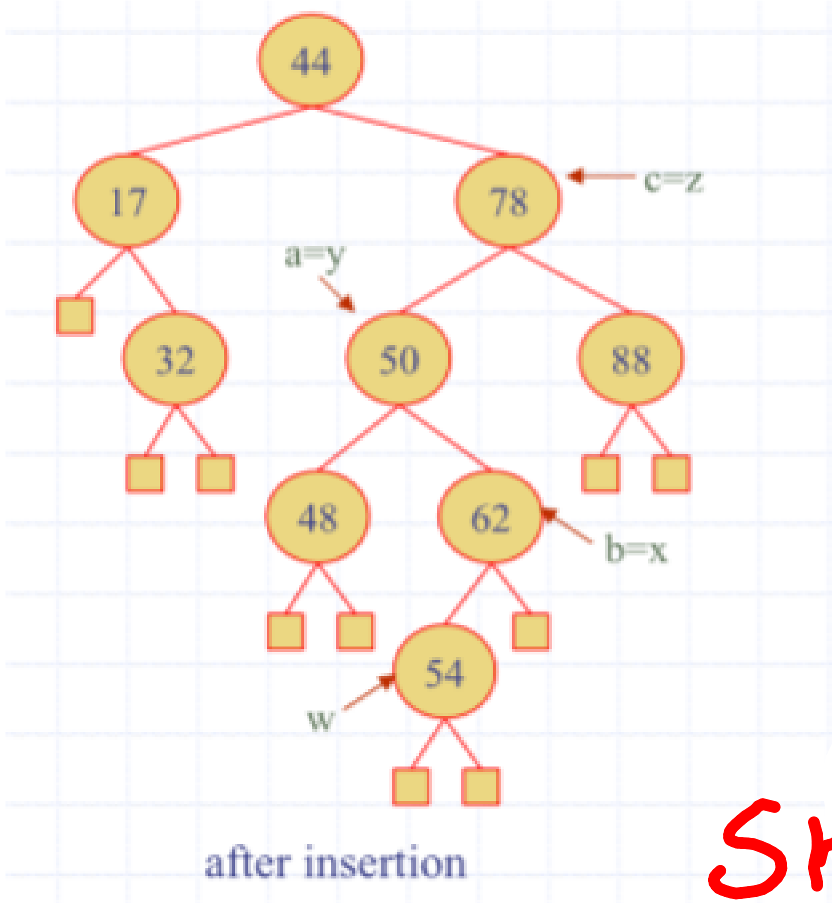
Check your neighbour's soln in terms of the following 3 points:

(a) whether the tree is balanced

(b) whether the BST property is satisfied after rebalancing. Discuss and come up with a common solution

PAIR

(c) Positives & negatives of yours & nbrs solns



Q: What will be the "general" procedure for "rebalancing" an imbalanced tree following an insertion?

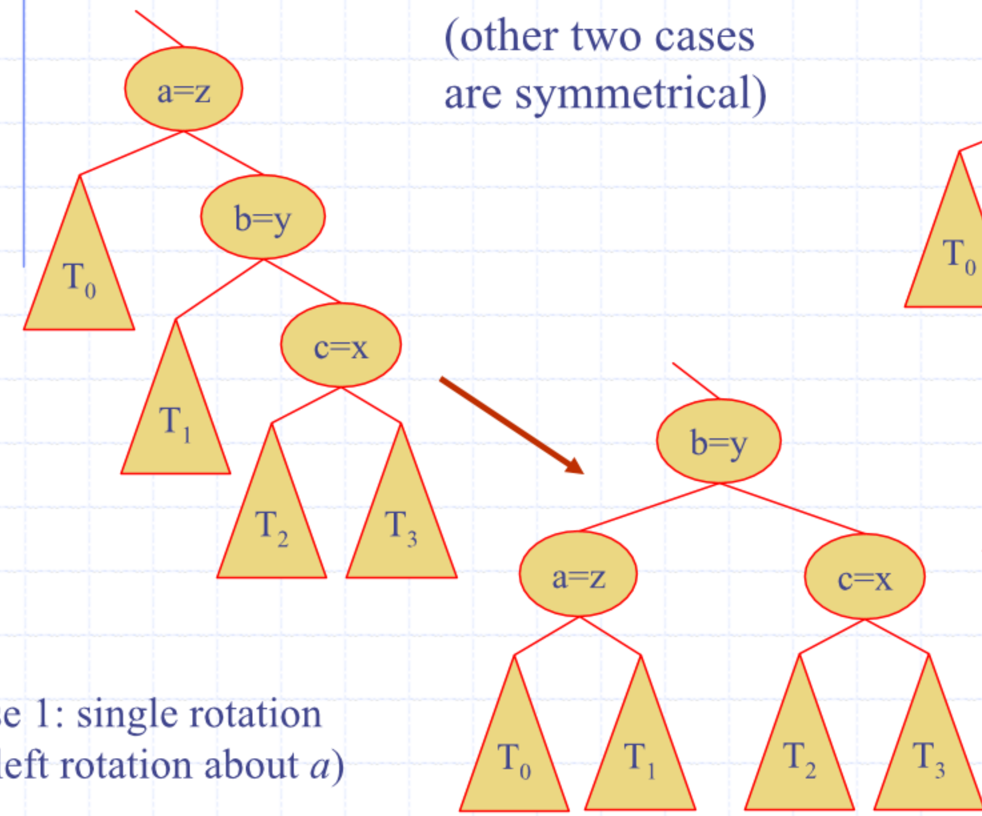
SHARE SOME OF YOUR SOLUTIONS WITH THE CLASS

SHARE

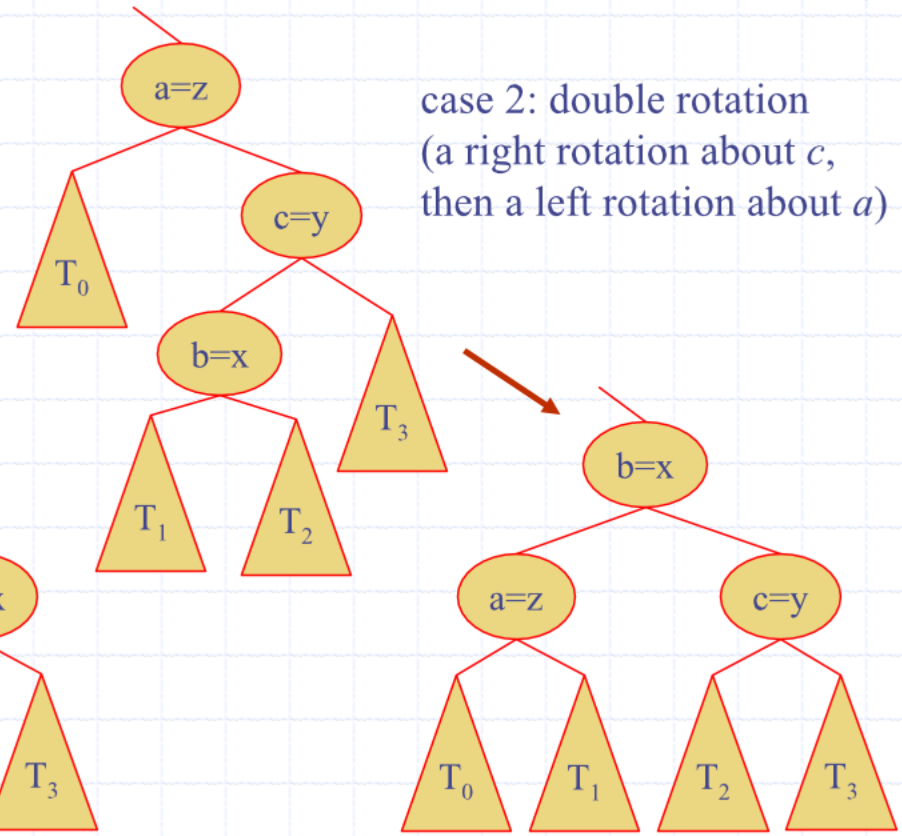
Trinode Restructuring

- let (*a*,*b*,*c*) be an inorder listing of *x*, *y*, *z* (inserted node → x → y → z)
- perform the rotations needed to make *b* the topmost node of the three

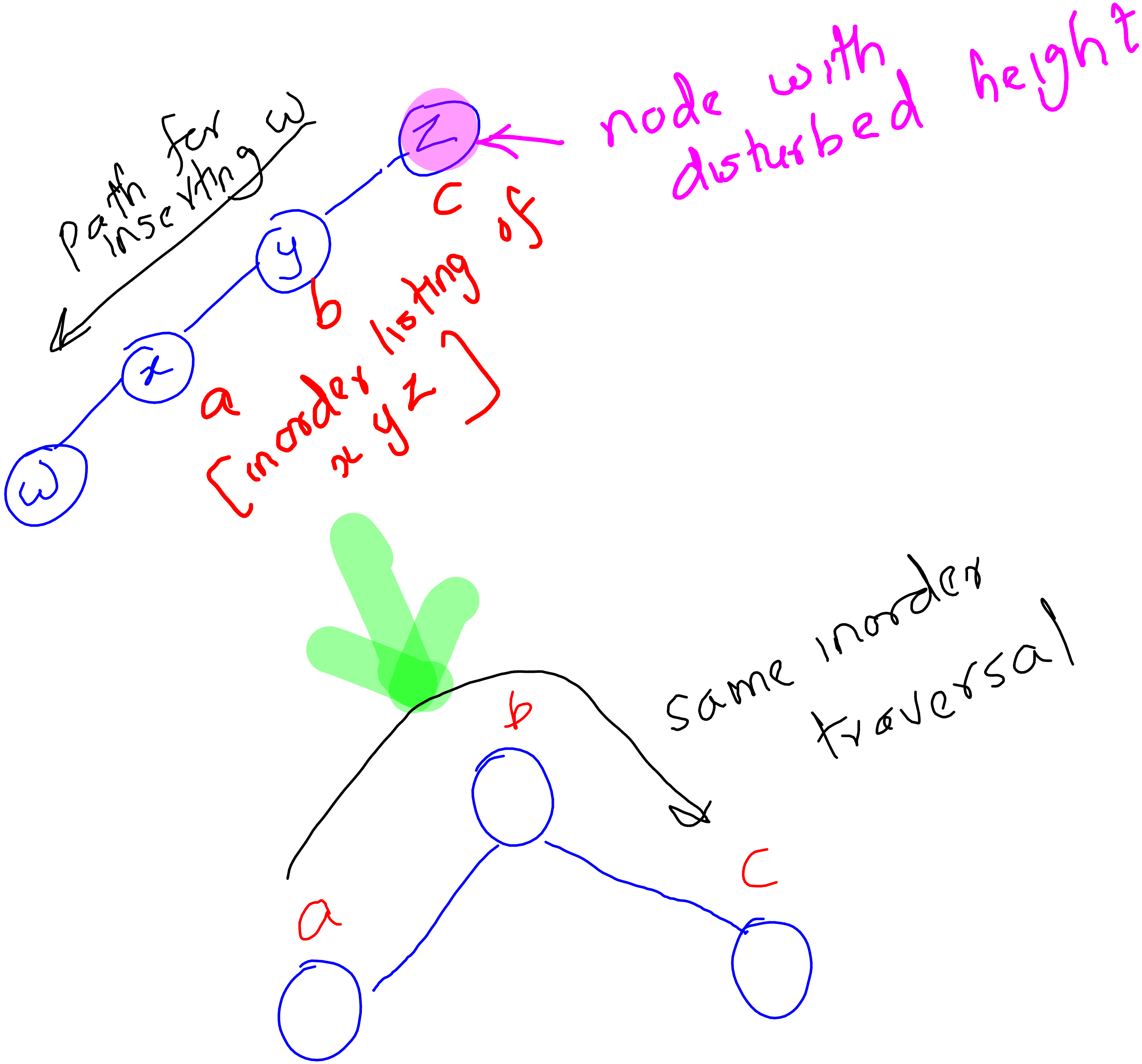
(other two cases are symmetrical)



case 1: single rotation
(a left rotation about *a*)

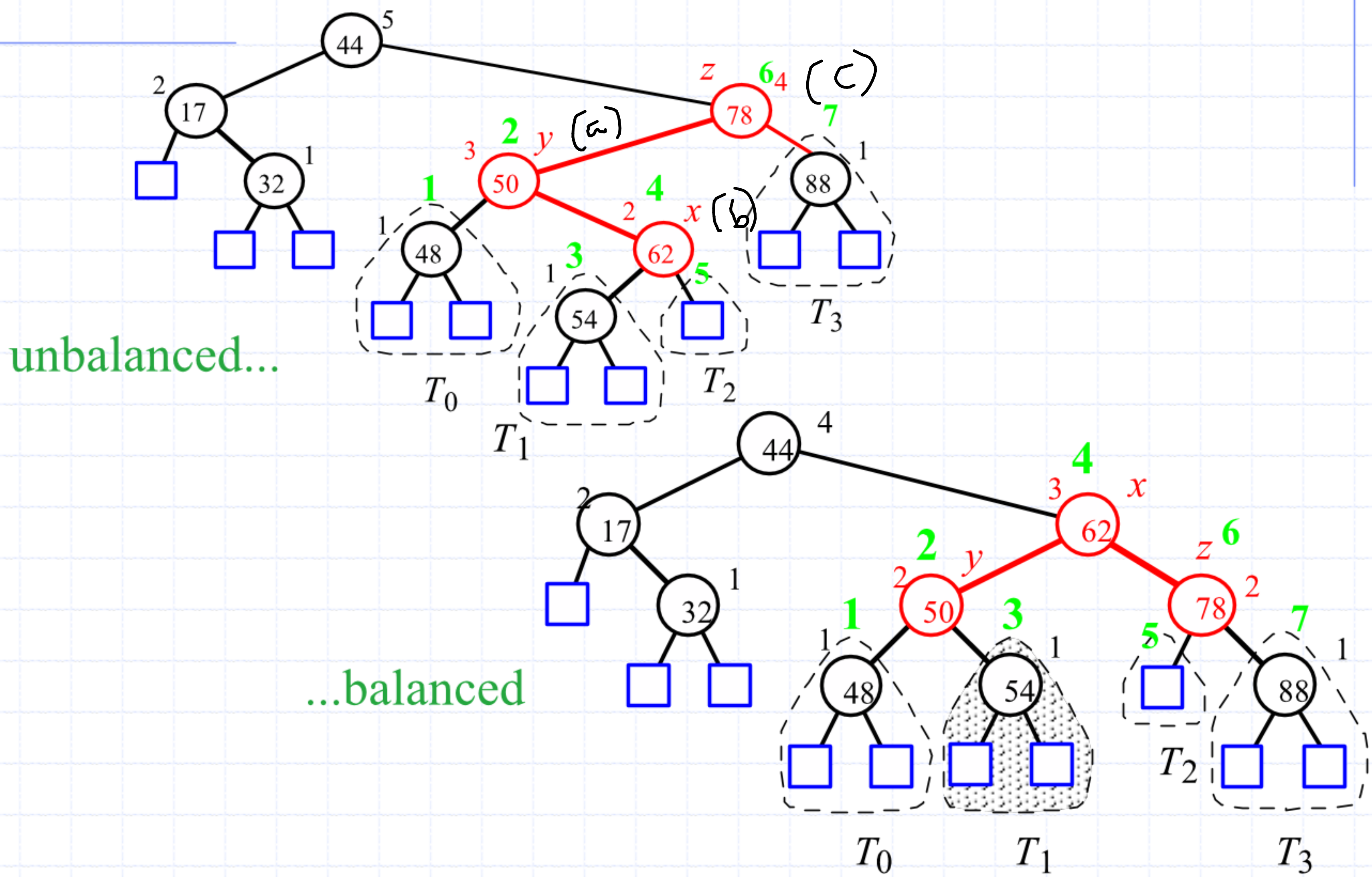


case 2: double rotation
(a right rotation about *c*, then a left rotation about *a*)



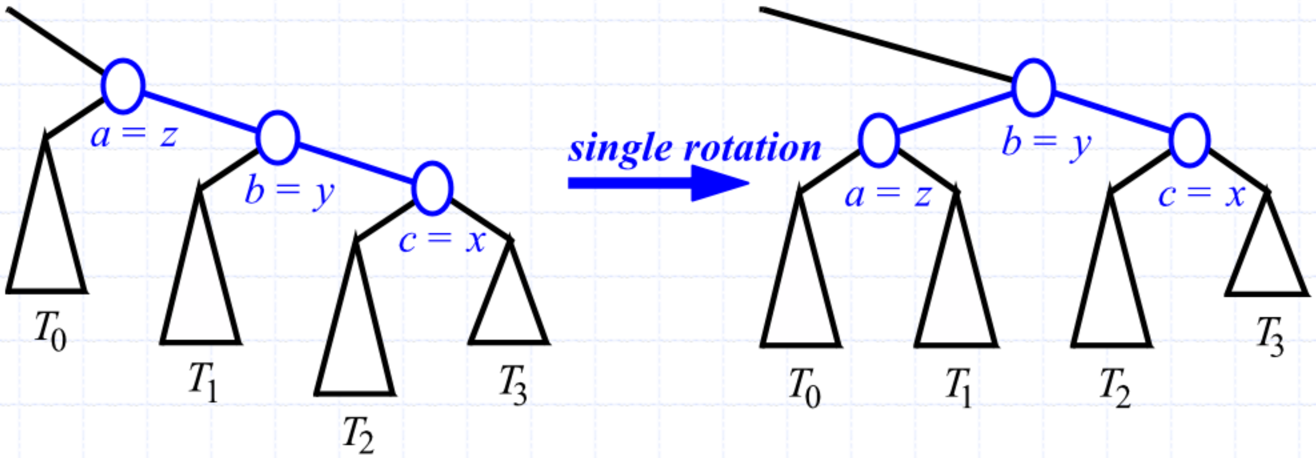
path for inserting w
z
y
x
w
node with disturbed height
c
b
a
[inorder listing of x y z]
same inorder traversal
b
a
c

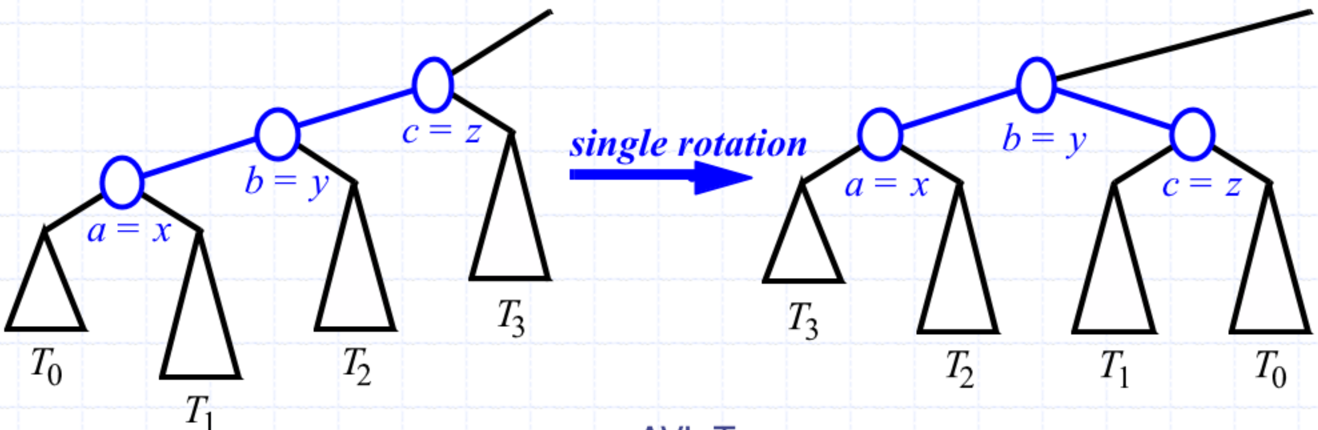
Insertion Example, continued



Restructuring (as Single Rotations)

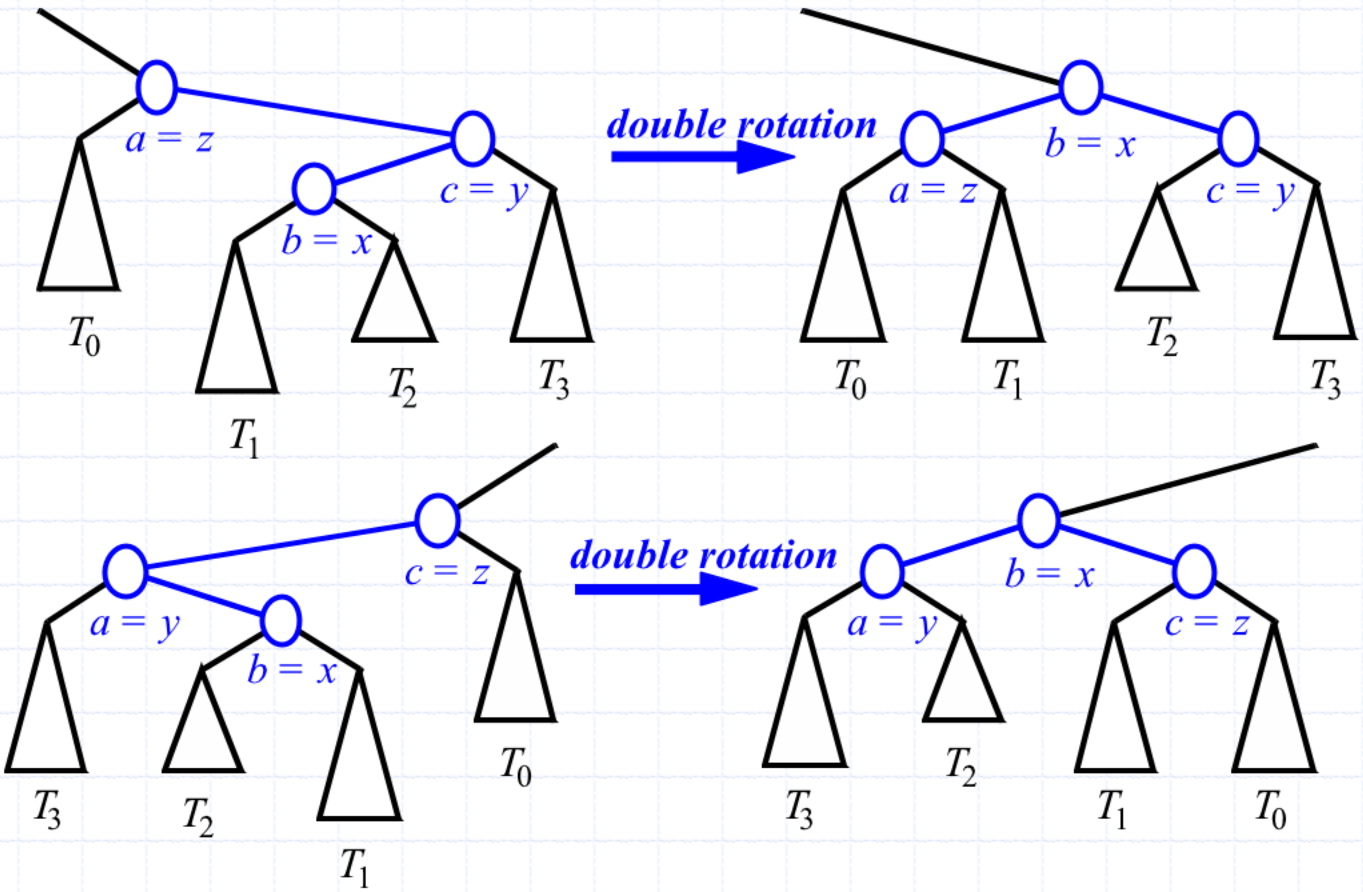
- Single Rotations:





Restructuring (as Double Rotations)

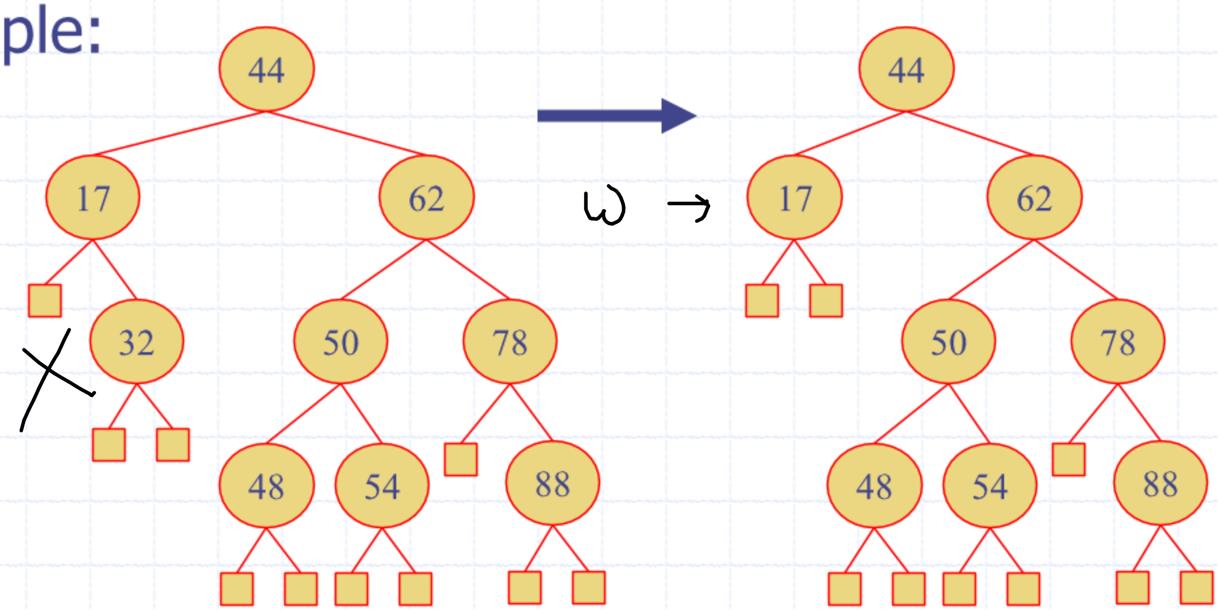
- double rotations:



In insertion: You know the specific "disturbed" path
Not the case in deletion!

Removal in an AVL Tree

- Removal begins as in a binary search tree, which means the node removed will become an empty external node. Its parent, w, may cause an imbalance.
- Example:



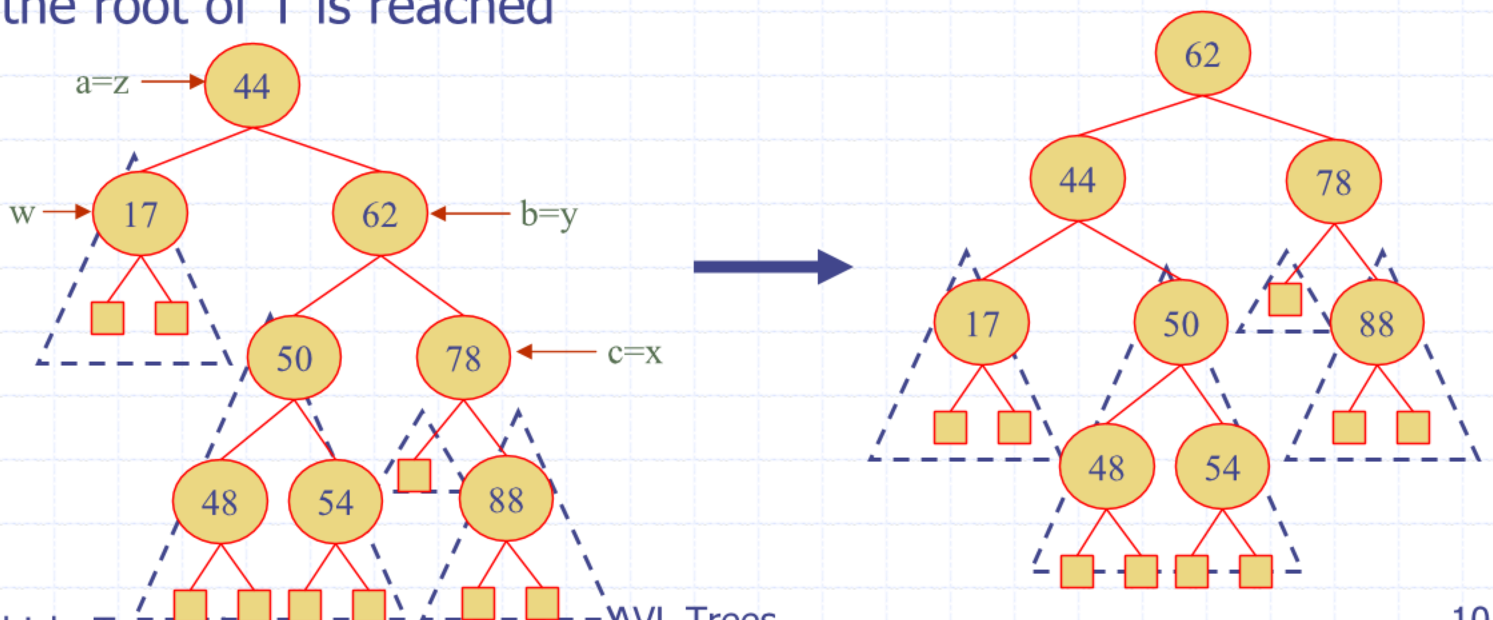
before deletion of 32

after deletion

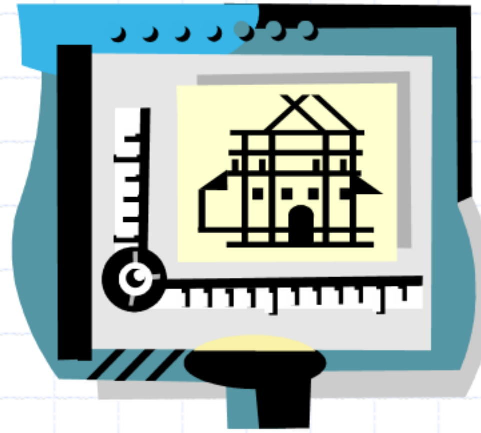
Rebalancing after a Removal

Not path of deletion

- Let *z* be the first unbalanced node encountered while travelling up the tree from w. Also, let y be the child of z with the larger height, and let x be the child of y with the larger height.
- We perform restructure(x) to restore balance at z.
- As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of T is reached



Running Times for AVL Trees



- a single restructure is O(1)
  - using a linked-structure binary tree
- find is O(log n)
  - height of tree is O(log n), no restructures needed
- insert is O(log n)
  - initial find is O(log n)
  - Restructuring up the tree, maintaining heights is O(log n)
- remove is O(log n)
  - initial find is O(log n)
  - Restructuring up the tree, maintaining heights is O(log n)

since you need to find z where height was disturbed