بسم الله الرحمن الرحیم

ساختمان‌های داده

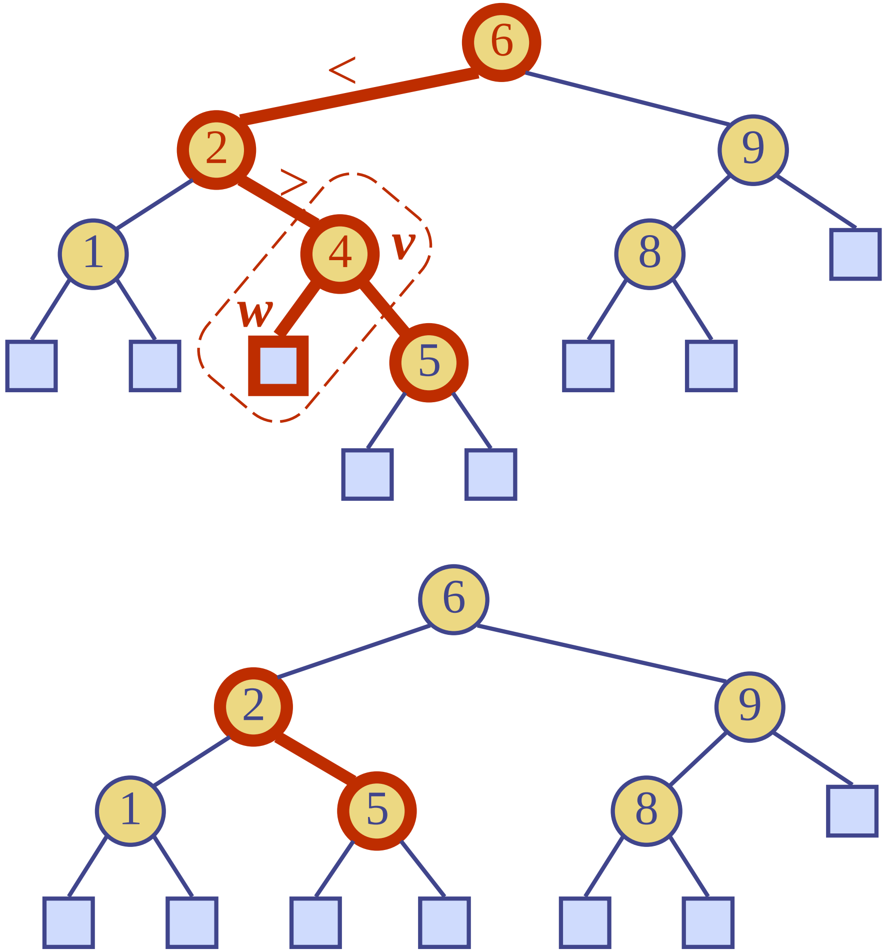
جلسه ۱۸

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

Deletion

- To perform operation erase(k), we search for key k
- Assume key k is in the tree, and let v be the node storing k
- Basic method
  - removeAboveExternal(w): removes w and its parent
- If node v has a leaf child w, we remove v and w from the tree with removeAboveExternal(w)
- What about “remove 1”?

Example: remove 4



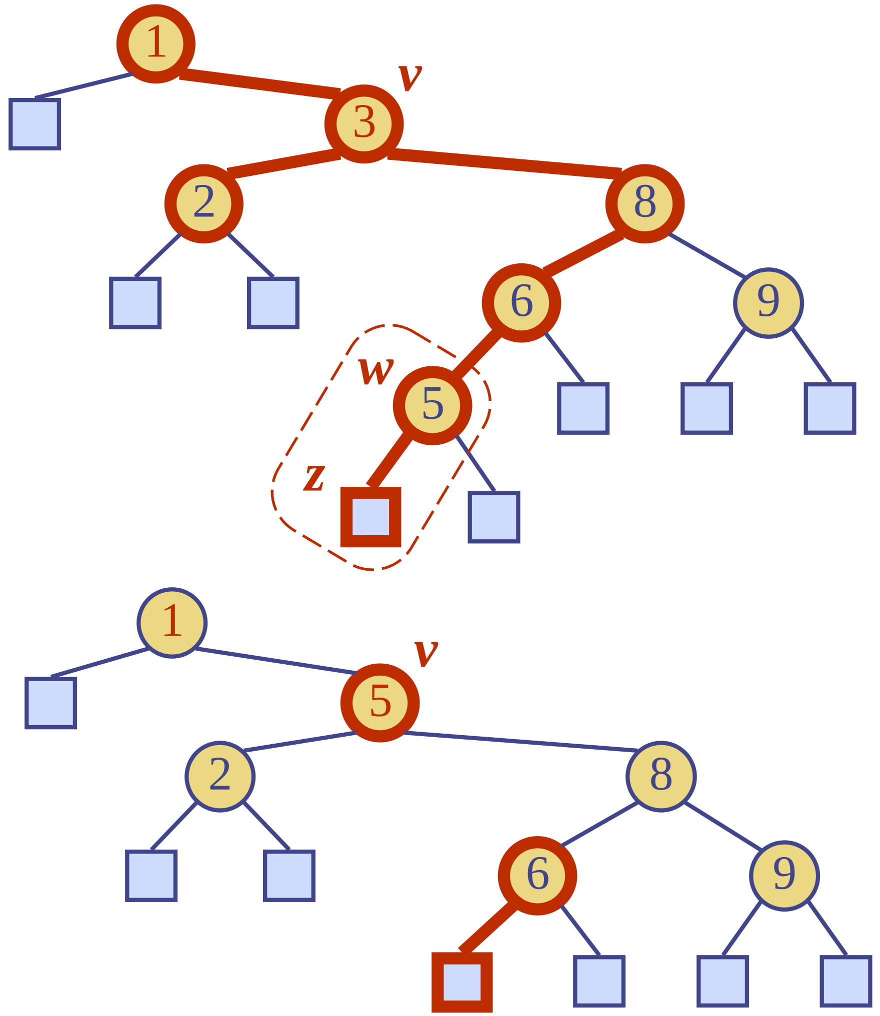
Deletion (cont.)

- Key ***k*** to be removed is stored at <u>a node ***v*** whose children are both internal</u>

- 1. Find the internal node ***w*** that follows ***v*** in an inorder traversal
  (find the smallest *w* larger than *v*)
- 2. Copy ***key***(***w***) into node ***v***
- 3. Remove node ***w*** and its left child ***z*** (which must be a leaf) by means of operation removeExternal(***z***)
  - Why left child z?

- *No other cases?*

Example: remove 3



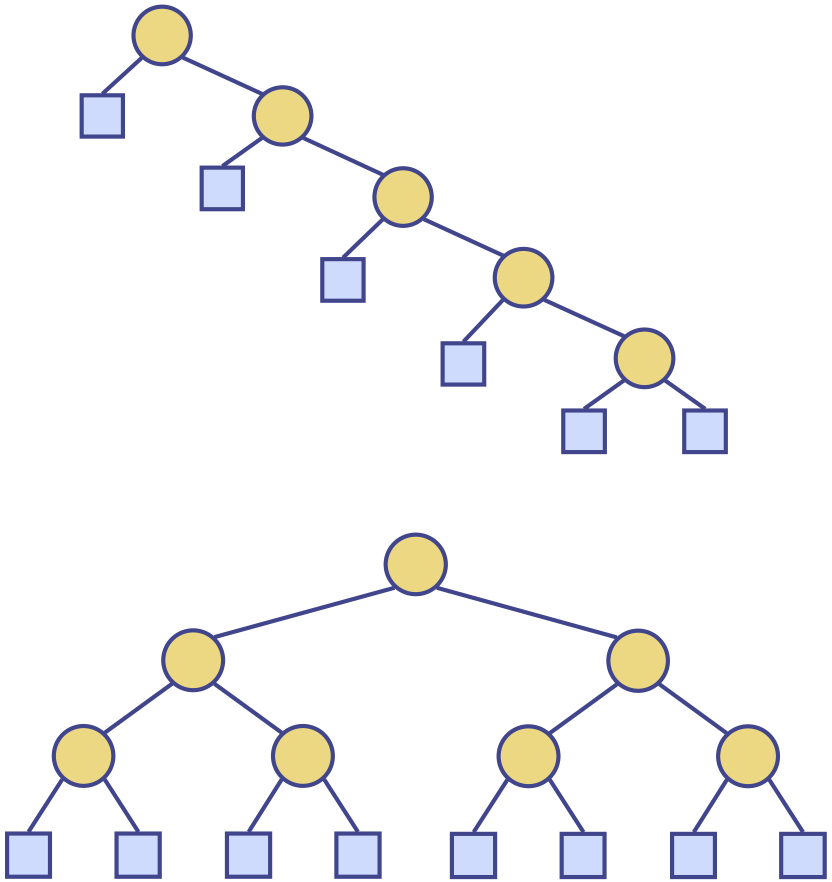
Performance

IUT-ECE

- Consider an ordered map with n items implemented by a binary search tree of height h
  - Space: $O(n)$
  - methods get, floorEntry, ceilingEntry, put and erase take $O(h)$ time

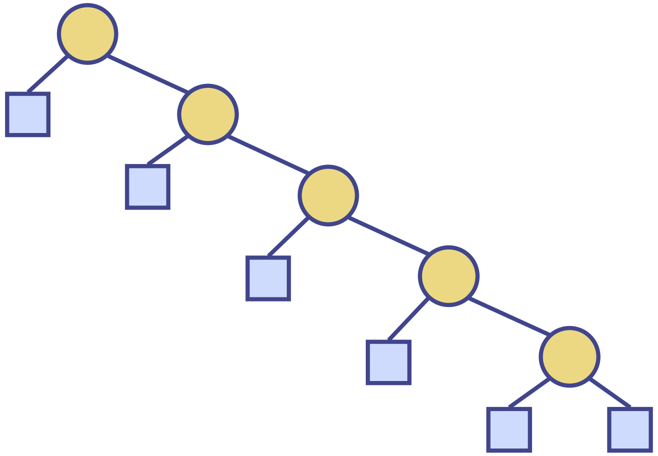
- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case

- Question: Can we find the algorithm with worst-case
  $O(\log n)$

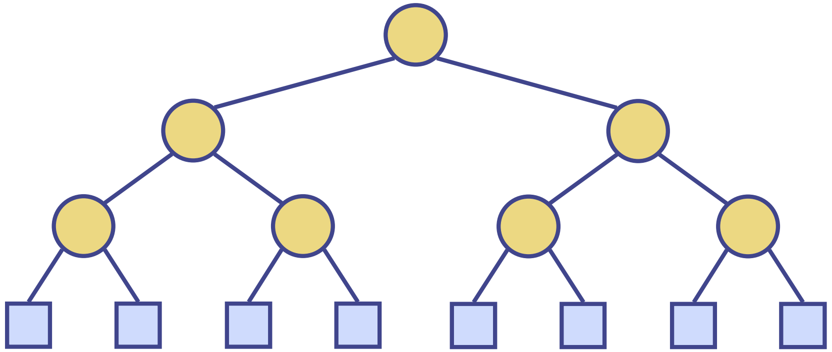


Performance

- Consider an ordered map with n items implemented by a binary search tree of height h
  - Space: $O(n)$
  - methods get, floorEntry, ceilingEntry, put and erase take $O(h)$ time



- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case



- Question: Can we find the algorithm with worst-case $O(\log n)$
  - Idea??? Balancing

Balance

- راهکارهایی برای اینکه BST را بالانس نگه داریم:
  - AVL
  - Red-Black
  - Splay
  - ....

AVL Trees

Adelson-Velskii, G.; E. M. Landis (1962). “"An algorithm for the organization of information"”. Proceedings of the USSR Academy of Sciences ***146****: 263–266.* (Russian) English translation by Myron J. Ricci in *Soviet Math. Doklady*, 3:1259–1263, 1962.

AVL tree

- AVL را self-balancing گوییم.
  - مرتبه زمانی بالانس کردن چقدر باید باشد؟

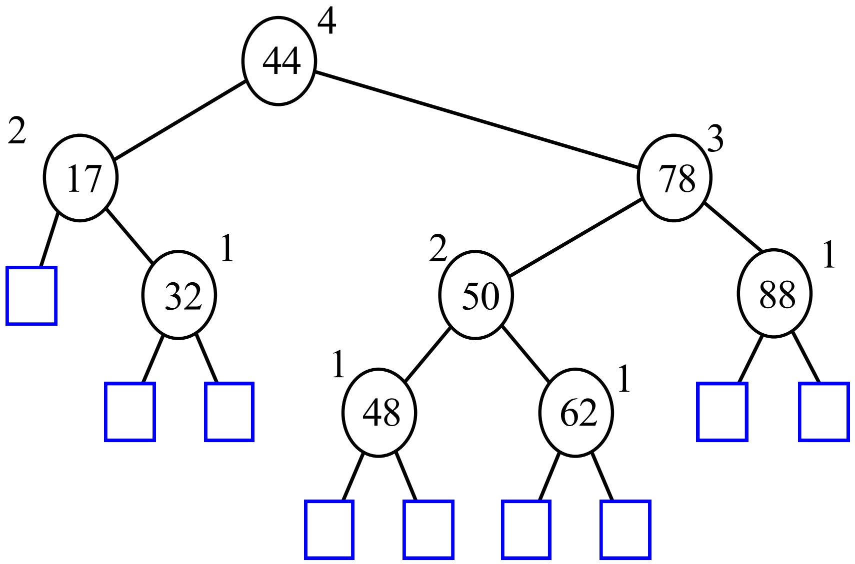
AVL tree

- AVL را self-balancing گوییم.
  - مرتبه زمانی بالانس کردن چقدر باید باشد؟ O(1) یا O(log n)

- آیا میتوانیم هر درختی را به صورت perfect بالانس کنیم؟

AVL Tree Definition

IUT-ECE

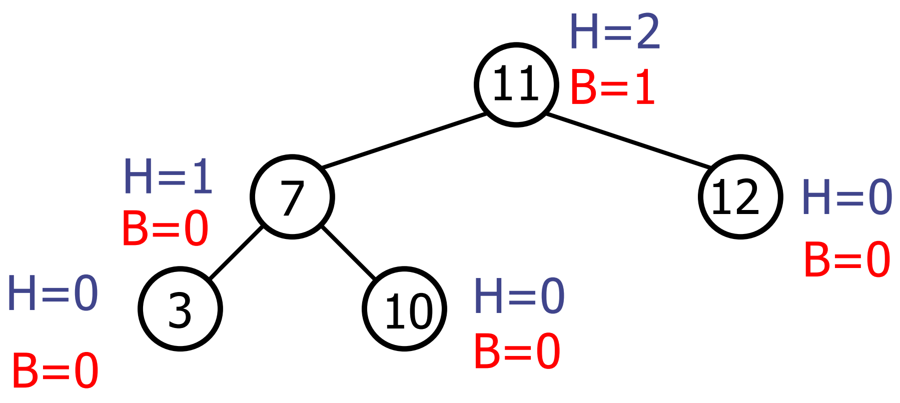


- An AVL Tree T is a binary search tree with the following property
  - *Height-Balance:*
    For every internal node v of T, the heights of the children of v can differ by at most 1
- This tree seems to be well-balanced
  - Height: O(log n)

AVL tree

○ AVL؟

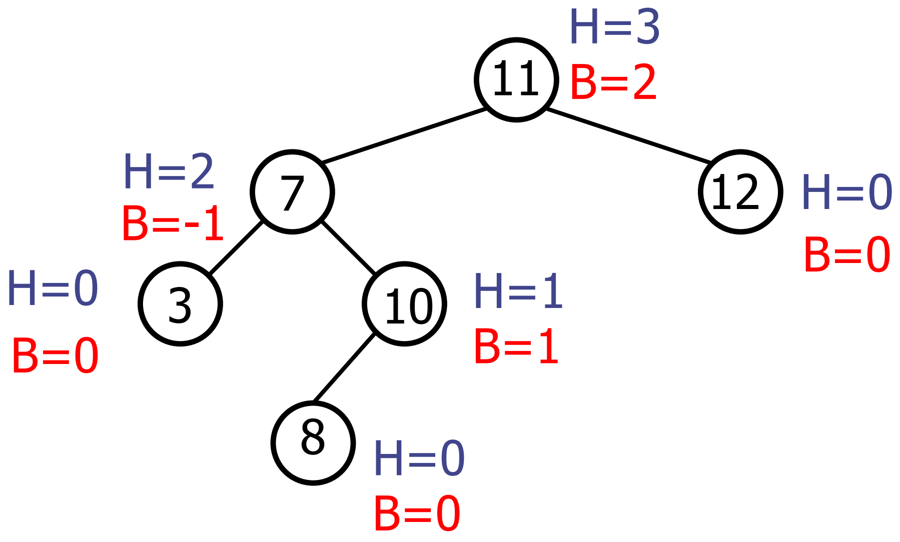
$-1 \leq \text{balance}(v)=\text{h(left)}-\text{h(right)} \leq 1$



AVL tree

- AVL؟

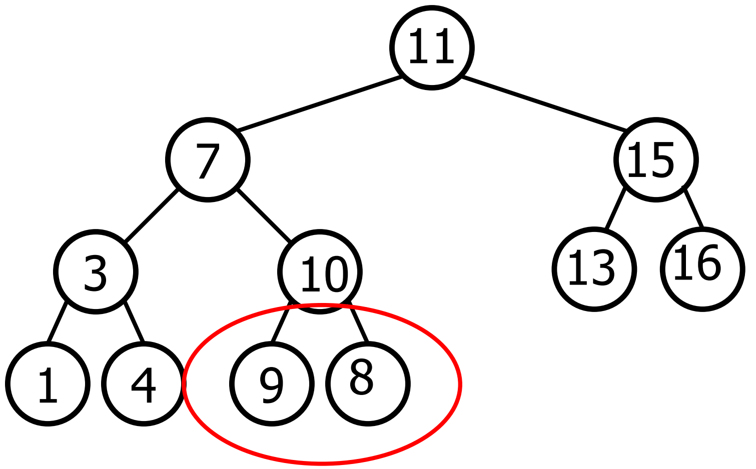
$-1 \leq \text{balance}(v)=\text{h(left)-h(right)} \leq 1$



No

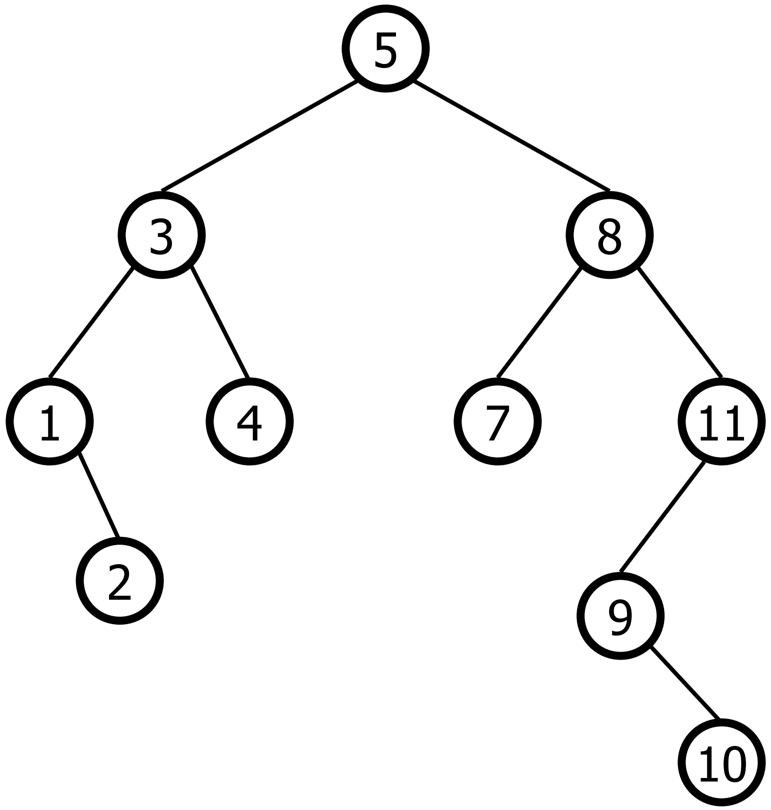
AVL tree

- AVL؟



AVL tree

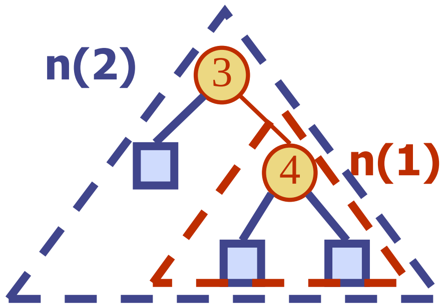
- AVL؟



Height of an AVL Tree (1)

IUT-ECE

- Fact: The height of an AVL tree storing *n* keys is *O(log n)*.
- Proof
  - *n(h)*: the minimum number of internal nodes of an AVL tree of height h.
  - Easily see that $n(1) = 1$ and $n(2) = 2$
  - For $h > 2$, an AVL tree of height *h* and the minimum number of nodes contains (i) the root node, (ii) one AVL subtree of height $h-1$ and (iii) another AVL subtree of height $h-2$.
  - That is, $n(h) = 1 + n(h-1) + n(h-2)$



Height of an AVL Tree (1)

- That is, $n(h) = 1 + n(h-1) + n(h-2)$

Fibonacci numbers

We define the ***Fibonacci numbers*** F_i, for $i \geq 0$, as follows:

$$F_i = \begin{cases} 0 & \text{if } i = 0 , \\ 1 & \text{if } i = 1 , \\ F_{i-1} + F_{i-2} & \text{if } i \geq 2 . \end{cases}$$

Height of an AVL Tree (1)

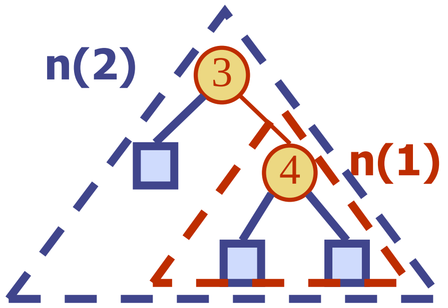
IUT-ECE

- Fact: The height of an AVL tree storing n keys is $O(\log n)$.
- Proof
  - $n(h)$: the minimum number of internal nodes of an AVL tree of height h.
  - Easily see that $n(1) = 1$ and $n(2) = 2$
  - For $h > 2$, an AVL tree of height h and the minimum number of nodes contains (i) the root node, (ii) one AVL subtree of height $h\text{-}1$ and (iii) another AVL subtree of height $h\text{-}2$.
  - That is, $n(h) = 1 + n(h\text{-}1) + n(h\text{-}2)$

- Knowing $n(h\text{-}1) > n(h\text{-}2)$, we get $n(h) > 2n(h\text{-}2)$. So

  $n(h) > 2n(h\text{-}2)$, $n(h) > 4n(h\text{-}4)$, $n(h) > 8n(n\text{-}6)$, … (by induction),

  $n(h) > 2^i n(h\text{-}2i)$



Height of an AVL Tree (2)

IUT-ECE

- *$n(h) > 2n(h-2)$, $n(h) > 4n(h-4)$, $n(h) > 8n(n-6)$, … (by induction),*

  $n(h) > 2^i n(h-2i)$ (for any integer i, such that $h-2i \geq 1$)
- We pick *i* so that *$h-2i = 1$ or 2* (base case)

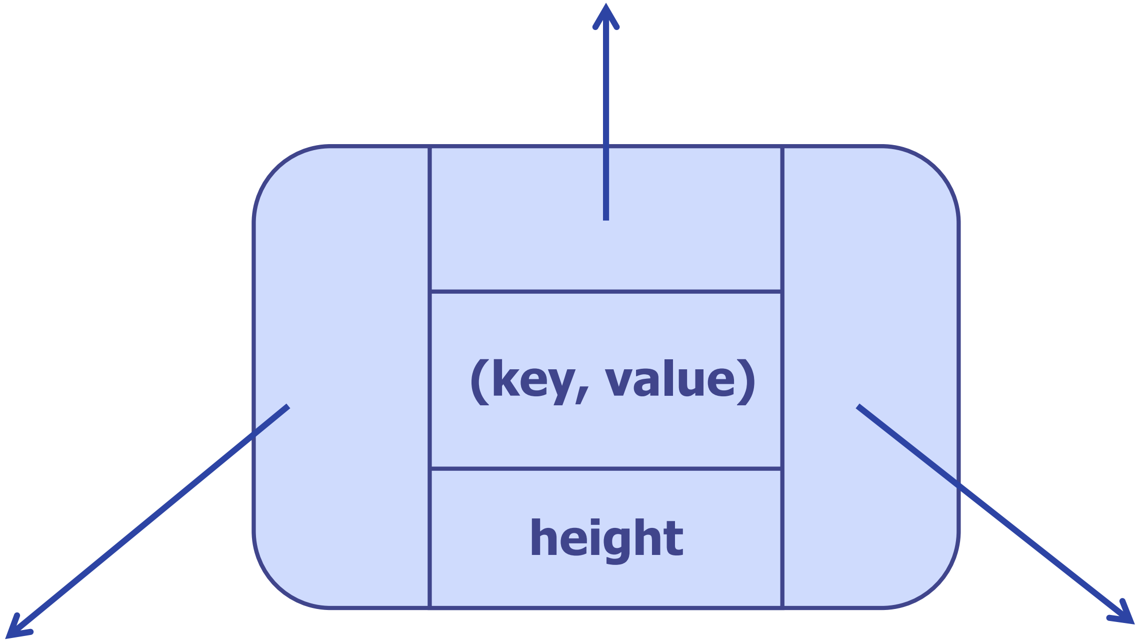
$$i = \left\lceil \frac{h}{2} \right\rceil - 1.$$

- Then, we have

$$\begin{aligned} n(h) &> 2^{\lceil \frac{h}{2} \rceil - 1} \cdot n\left(h - 2\left\lceil \frac{h}{2} \right\rceil + 2\right) \\ &\geq 2^{\lceil \frac{h}{2} \rceil - 1} n(1) \\ &\geq 2^{\frac{h}{2} - 1}. \end{aligned}$$

- Taking logarithms: *$h < 2\log n(h) + 2$*
- Thus, the height of an AVL tree is *$O(\log n)$*

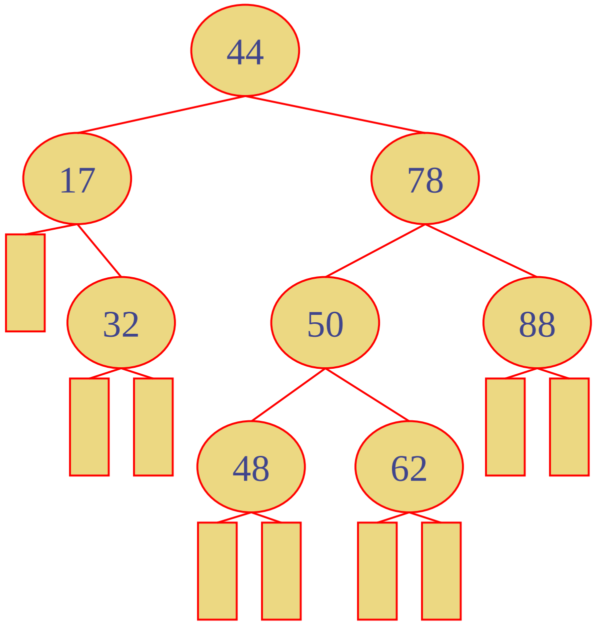
Node



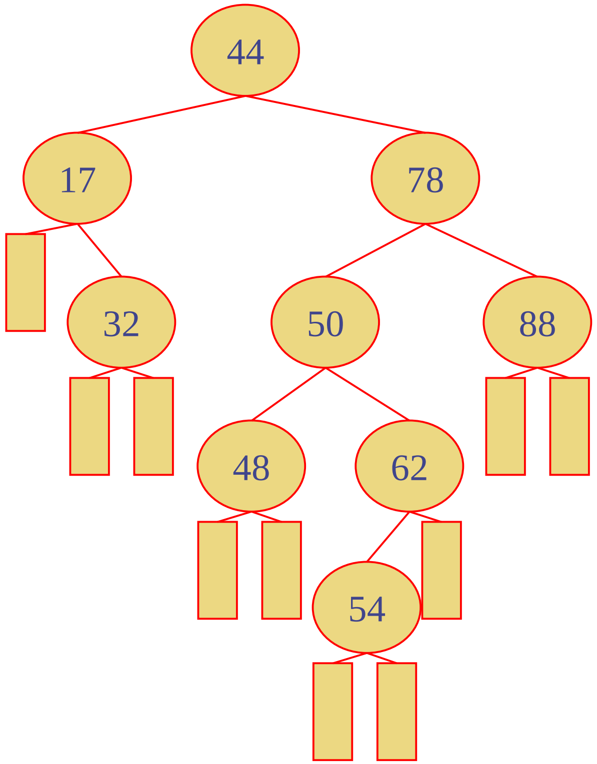
IUT-ECE

Insertion

- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example of insertion 54. What's the problem?



before insertion 54



after insertion 54

Rebalancing Needed

IUT-ECE

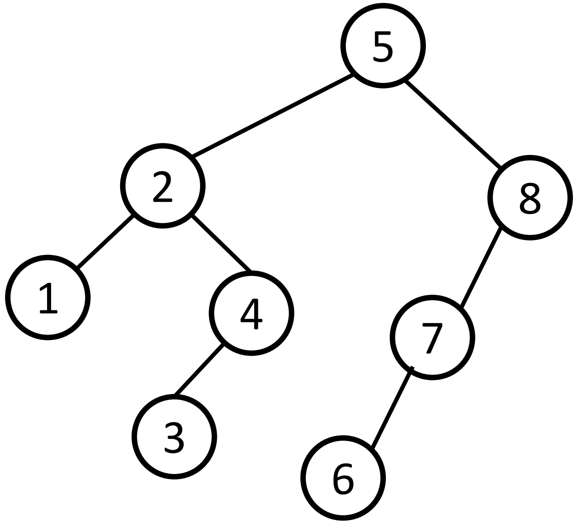
- How should we do this?
  - (1) Take some examples
  - (2) Find difference cases
  - (3) Make each sub-algorithm for each case
  - (4) Make an entire algorithm
  - (5) Run it with some inputs
  - (6) Find out it is not working perfectly, and say "What the hell is this?" "How should I do?"

- Lessons
  - Let's summarize them later

Rebalancing

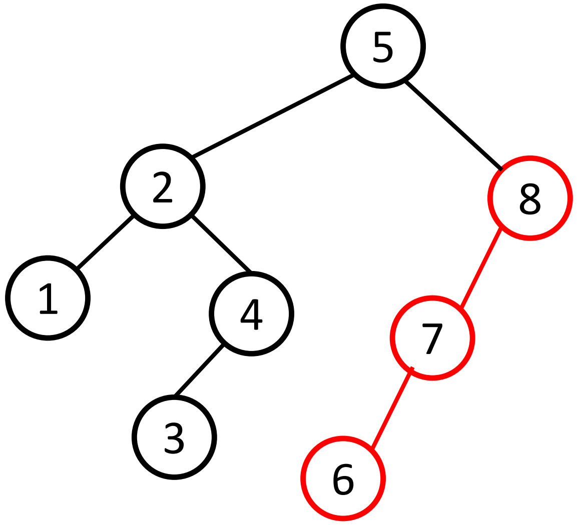
IUT-ECE

- Insert(6)
- بالانس؟



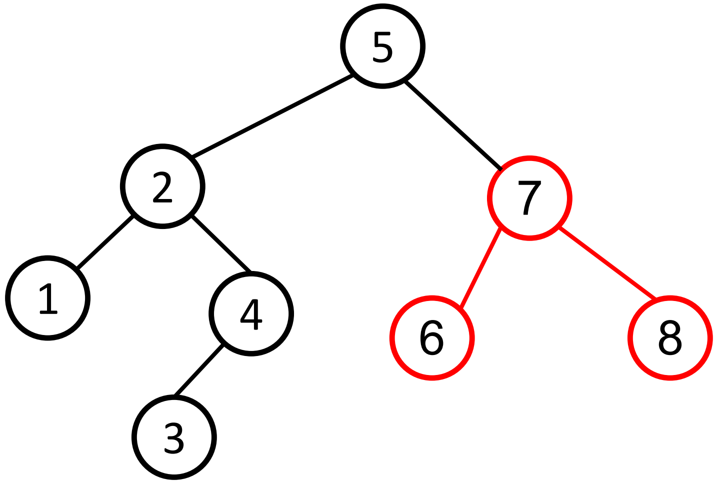
Rebalancing

- Insert(6)
- بالانس؟



Rebalancing

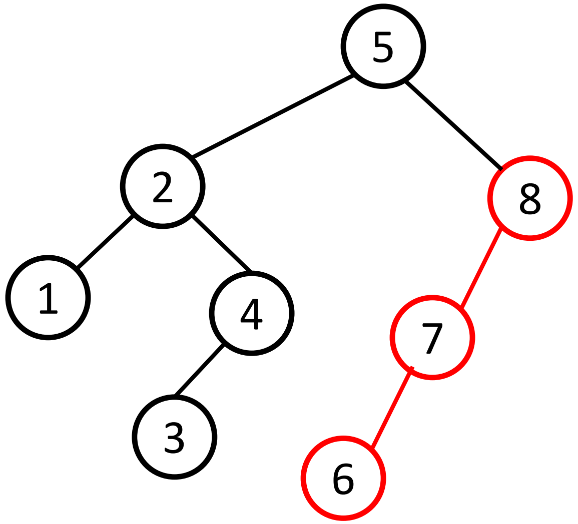
- Insert(6)
- بالانس؟



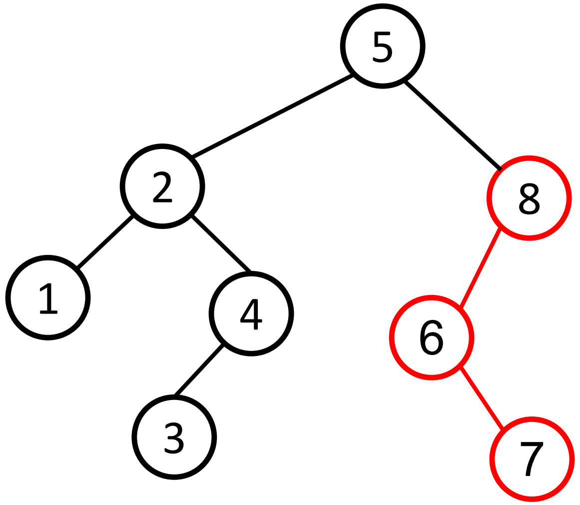
Rebalancing

IUT-ECE

- حالات دیگر؟



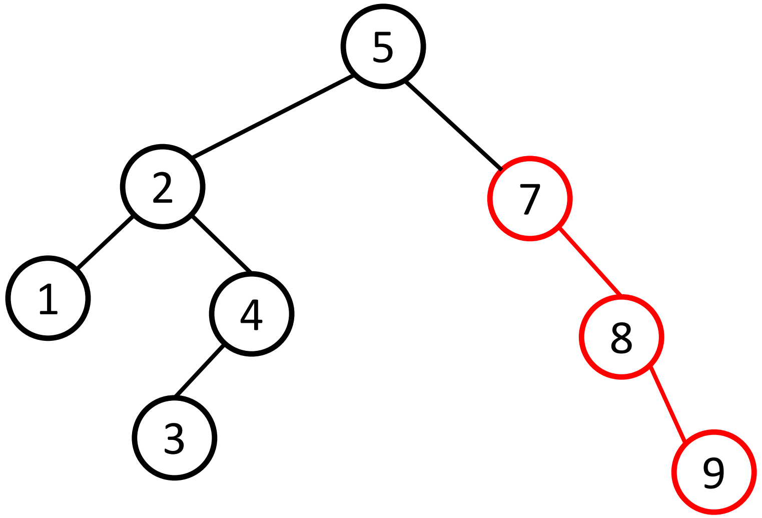
LL



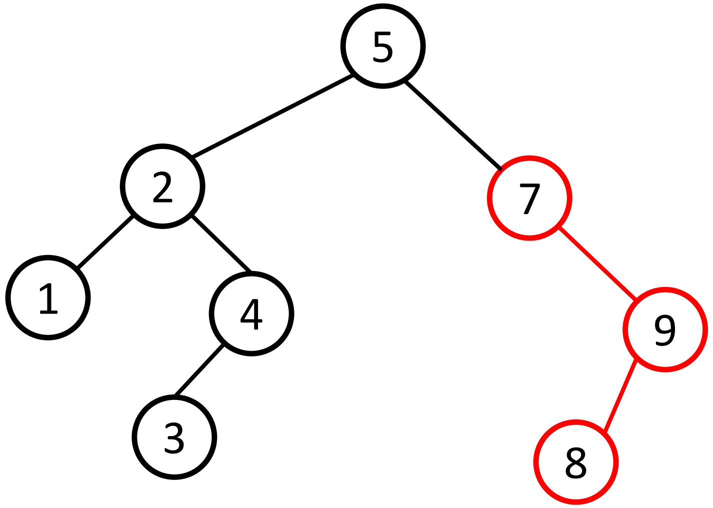
LR

Rebalancing

- حالات دیگر؟



RR

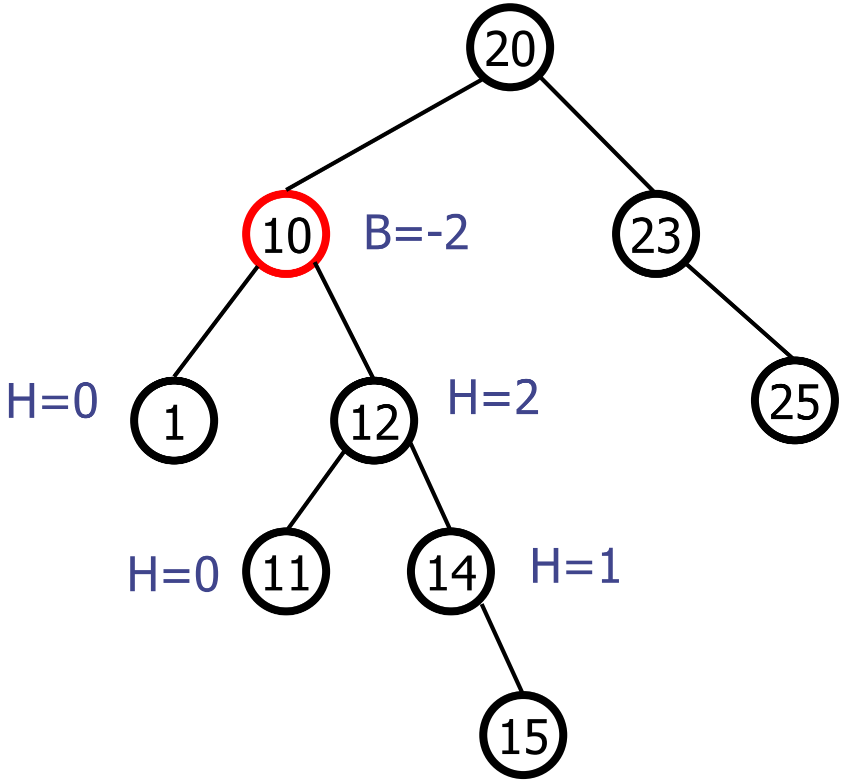


RL

Rebalancing

○ مثال دیگر

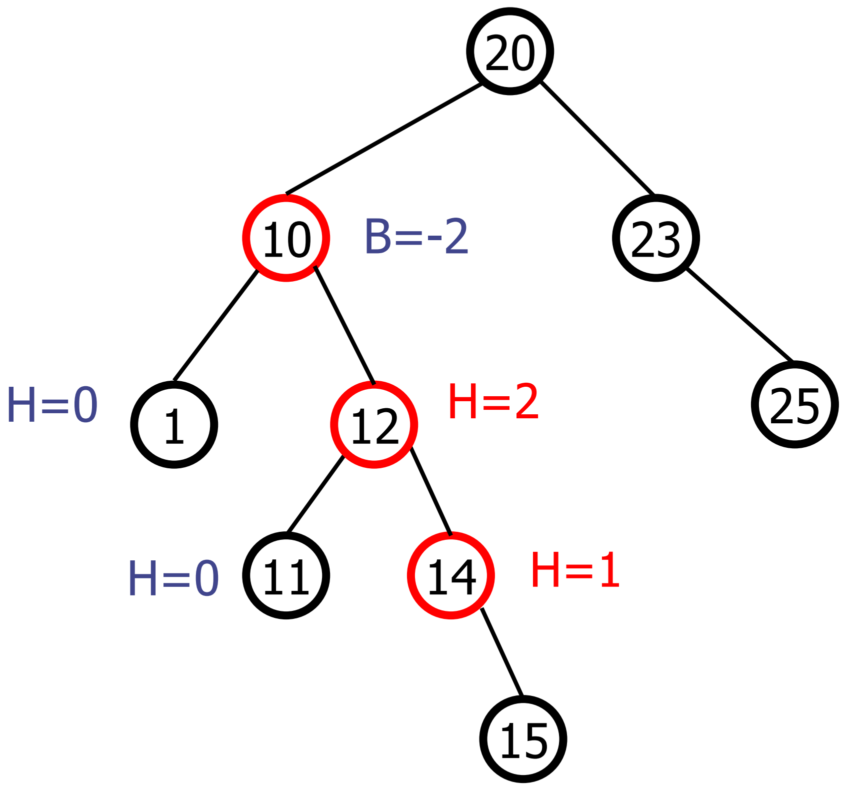
Insert(15)



Rebalancing

○ مثال دیگر

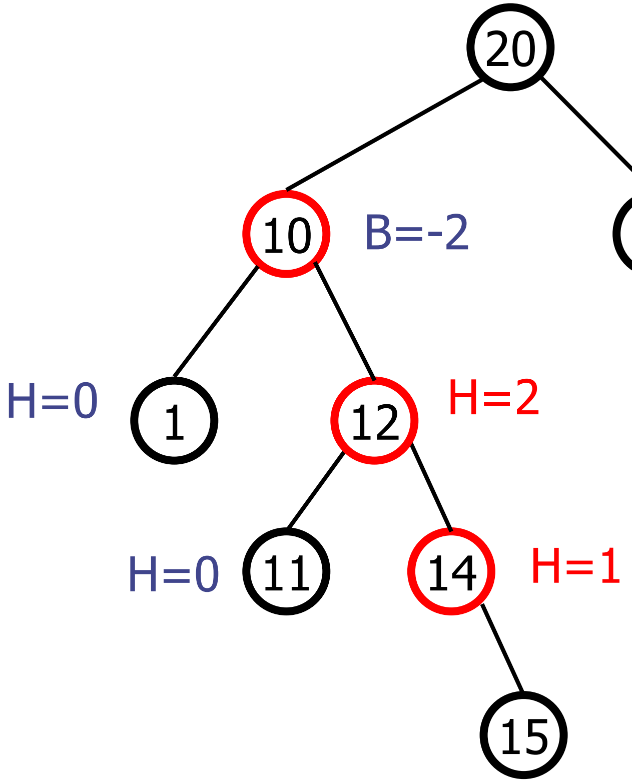
Insert(15)

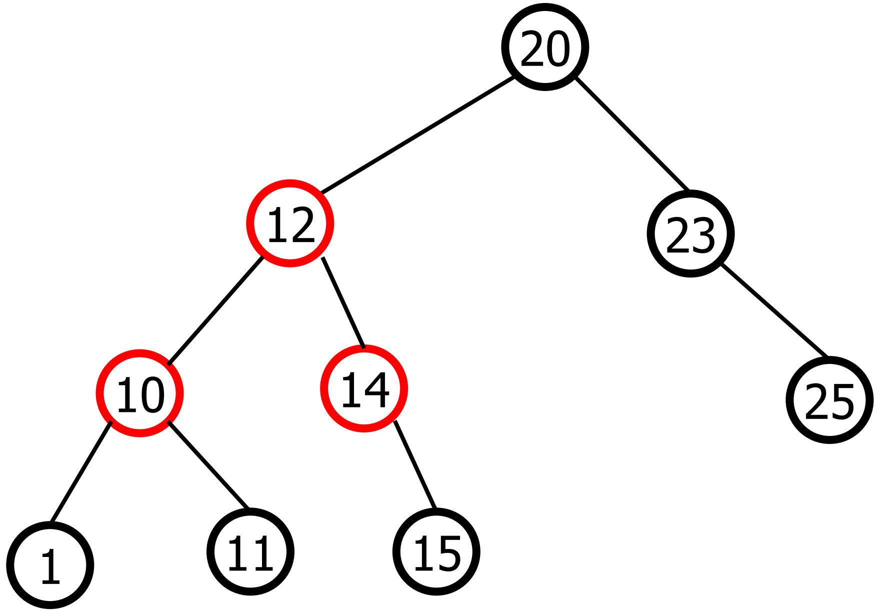


Rebalancing

○ مثال دیگر

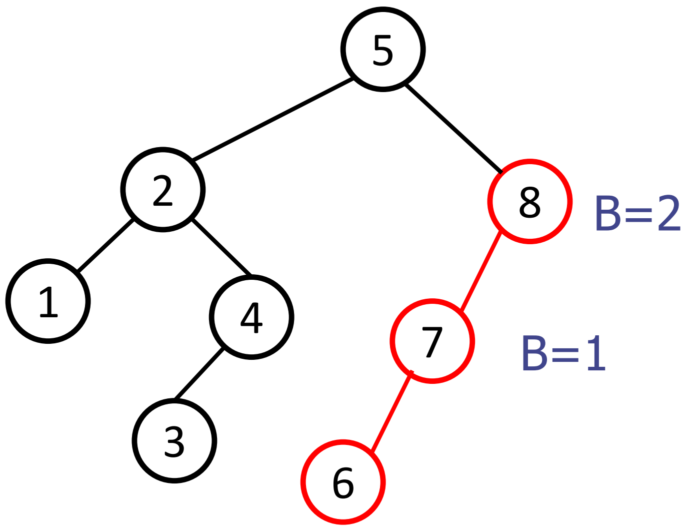
Insert(15)





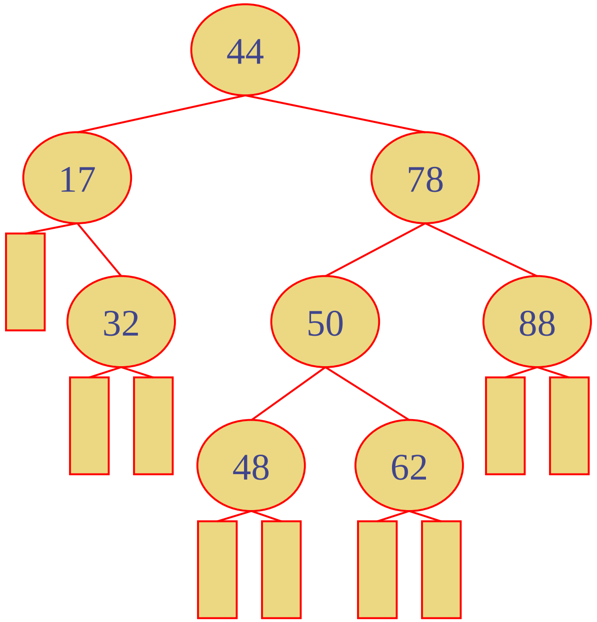
Rebalancing

- یک الگوریتم برای rebalancing
  - اضافه کردن یک گره جدید تنها در مسیر به سمت گره باعث عدم تعادل میشود.
  - میزان این عدم تعادل، یک واحد است.
  - نهایتا دو گره پایین آمدن (چگونه؟ با توجه به ارتفاع فرزند)

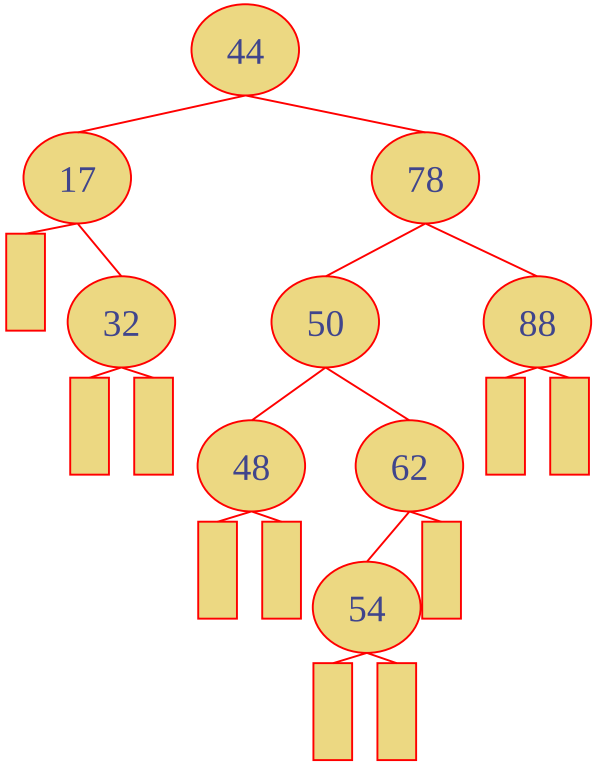


Insertion

- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example of insertion 54. What's the problem?



before insertion 54



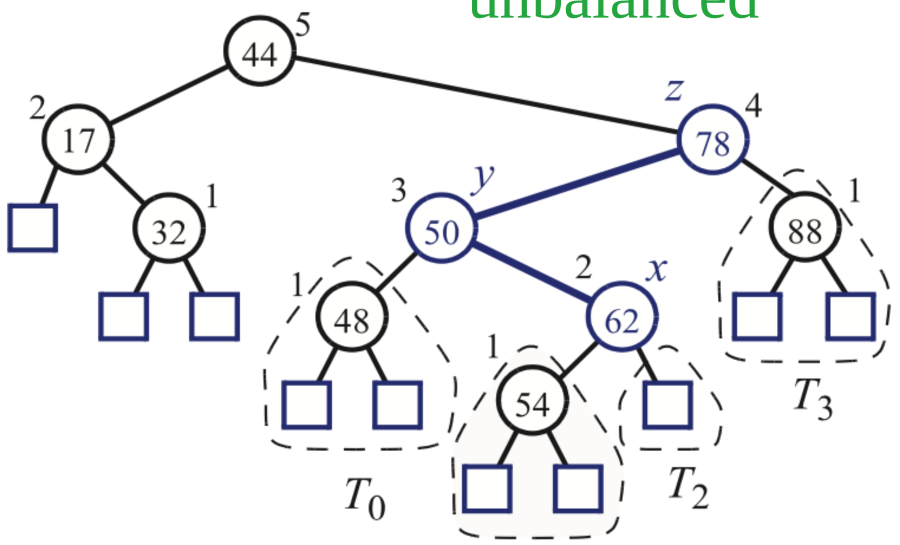
after insertion 54

Rebalancing Example: Insertion of w=54

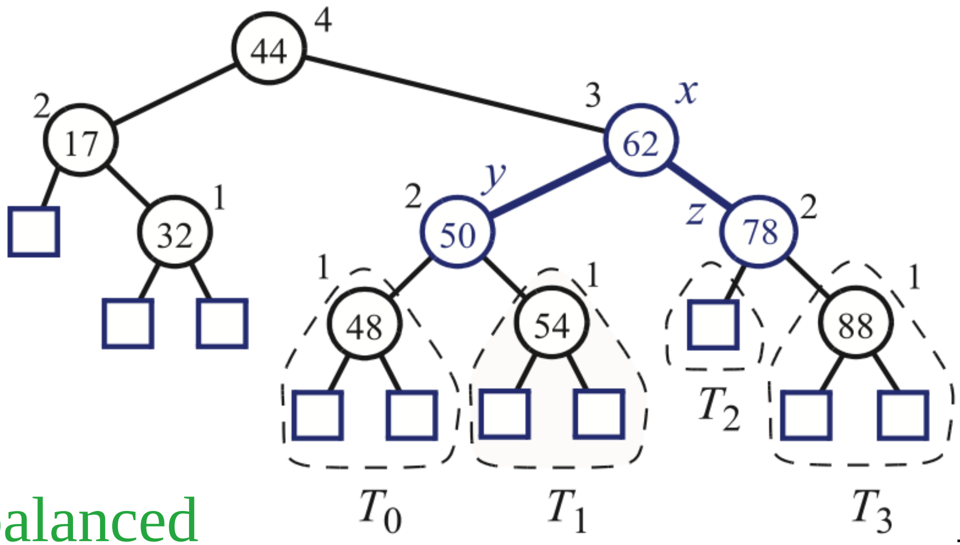
IUT-ECE

- “Search-and-Repair” strategy
- ***z***: first node we encounter in going up from w toward the root such that ***z*** is unbalanced
- ***y***: the child of ***z*** with higher height (note that ***y*** must be an ancestor of w)
- ***x***: the child of ***y*** with higher height (there cannot be a tie and node ***x*** must be an ancestor of ***w***)

unbalanced



- What are we doing for balancing?
- Can we do this systematically?
- What are other cases?

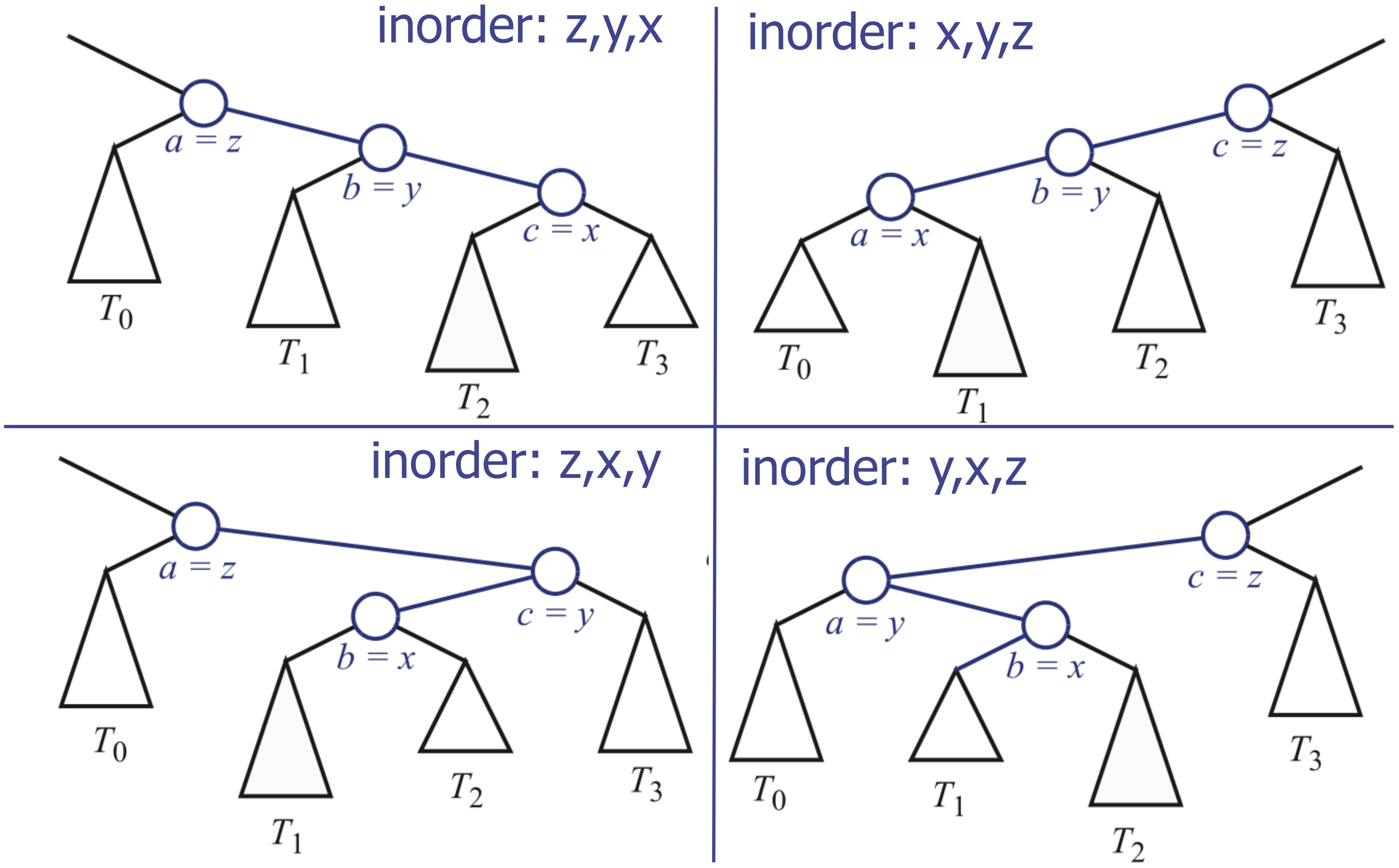


balanced

Please remember the notations! z, y, z

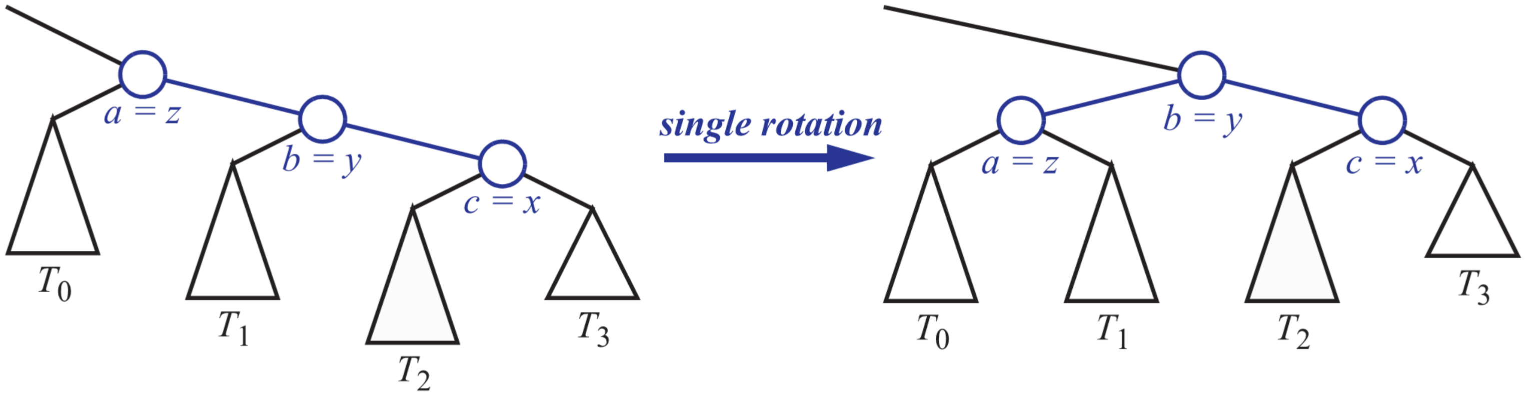
- ***z***: first node we encounter in going up from w toward the root such that ***z*** is unbalanced
- ***y***: the child of ***z*** with higher height
- ***x***: the child of ***y*** with higher height
- Rename ***x,y,z*** as ***a,b,c*** so that ***a*** precedes ***b*** and ***b*** precedes ***c*** in “inorder traversal”
  - We can make many combinations

4 Combinations

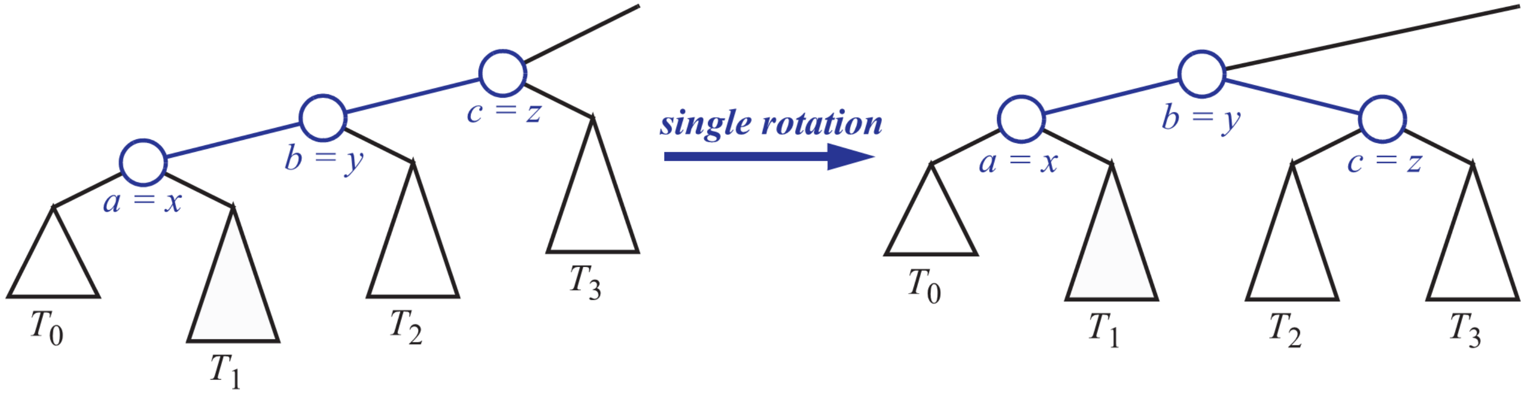


Restructuring (as Single Rotations)

- Single Rotations:

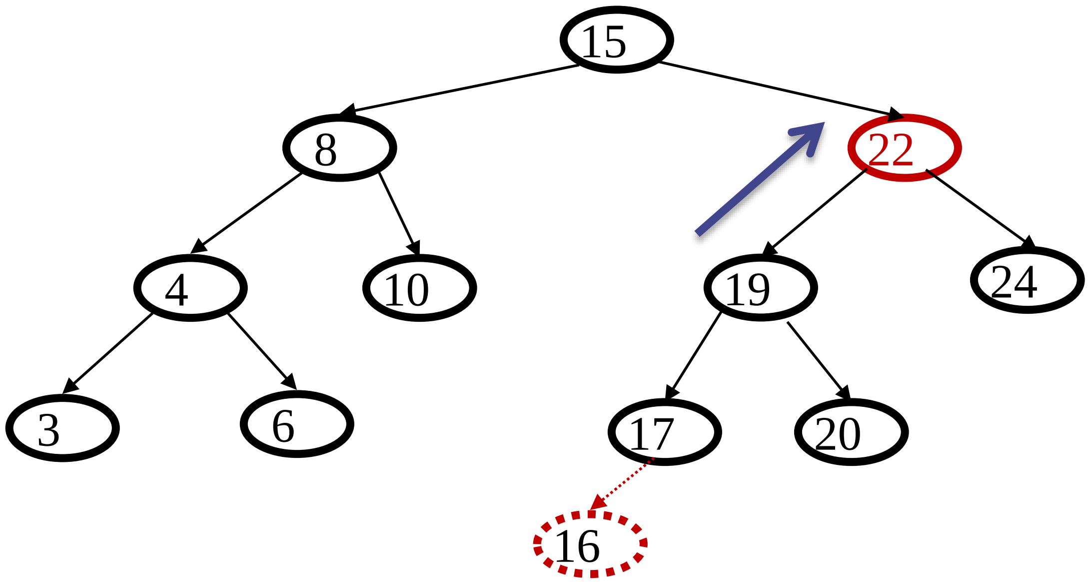


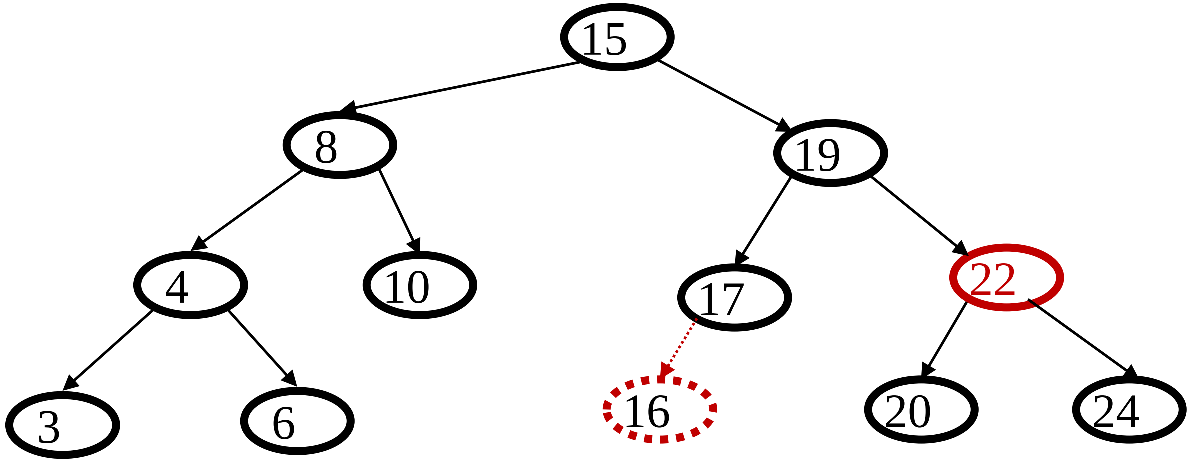
(a)



Example

Insert(16)

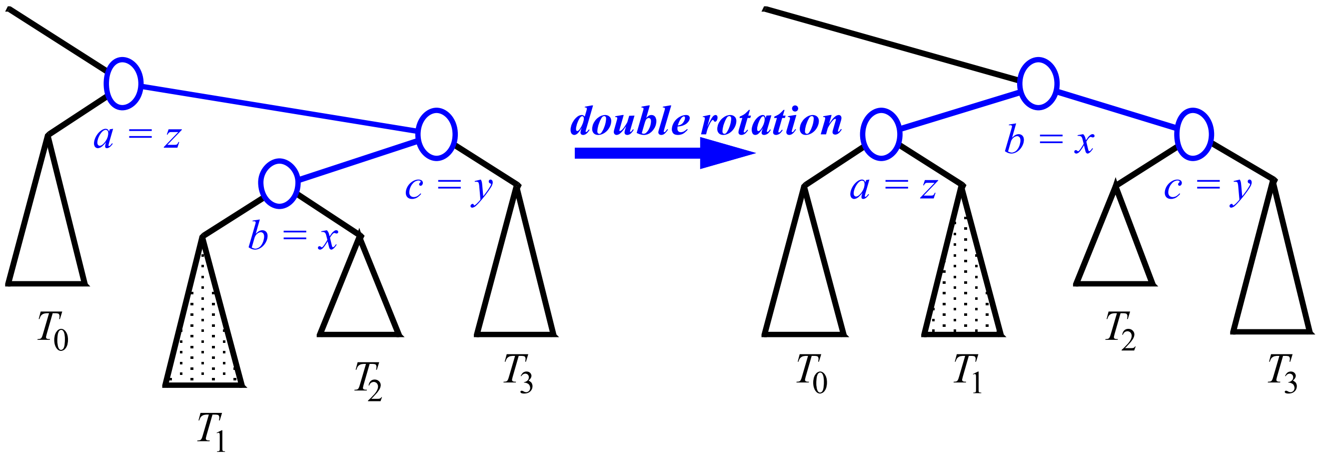




Restructuring (as Double Rotations)

- double rotations:

Right rotation about y
and left rotation about z



Left rotation about y
and right rotation about z