Introduction to Algorithms and Data Structures

Lecture 15: Data Structure (5) Dynamic Search Tree and its balancing

Professor Ryuhei Uehara,

School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

http://www.jaist.ac.jp/~uehara

Dynamic search and data structure

- Sometimes, we would like to search in dynamic data, i.e., we add/remove data in the data set.
- Example: Document management in university
  - New students: add to list
  - Alumni: remove from list
  - When you get credit: search the list

Q. Good data structure?

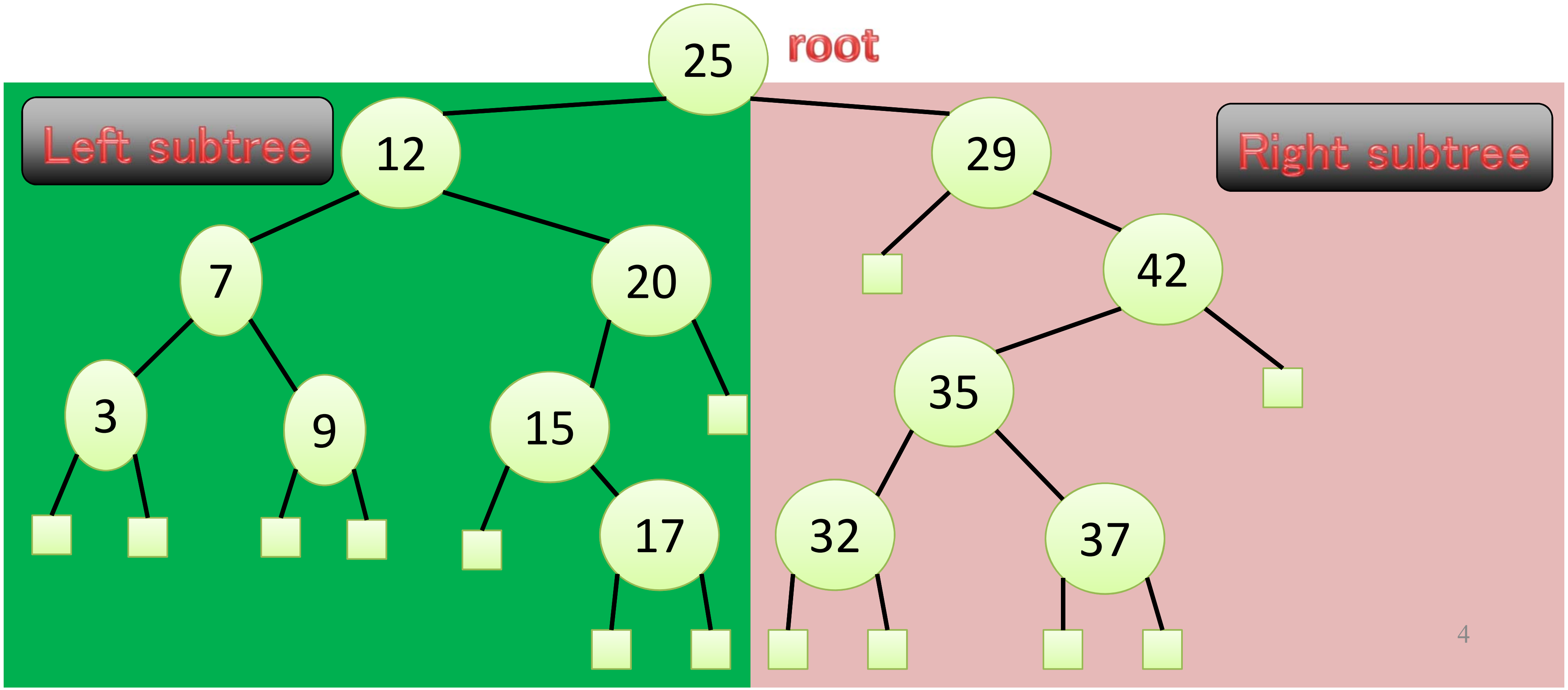
Naïve idea: array or linked list?

- Data in order:
  - Search: binary search in O(log n) time
  - Add and remove: O(n) time per data
- Data not in order:
  - Search and remove: O(n) time per data
  - Add: in O(1) time

Imagine: you have 10000 students, and you have 300 new students!

Better idea: binary search tree

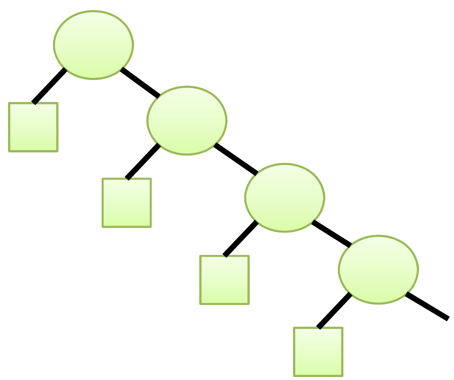
- For every vertex v, we have the following;
  - Data in v > any data in a vertex in left subtree
  - Data in v < any data in a vertex in right subtree



Better idea: binary search tree

- When data is random:
  - Depth of the tree: O(log n)
  - Search, add, remove: O(log n) time.

- In the worst case:
  - Depth of the tree: n
  - When data is given in order,
    we have the worst case.
  - Search, add, remove: O(n) time…



Nice idea:
(Self-)Balanced Binary Search Tree

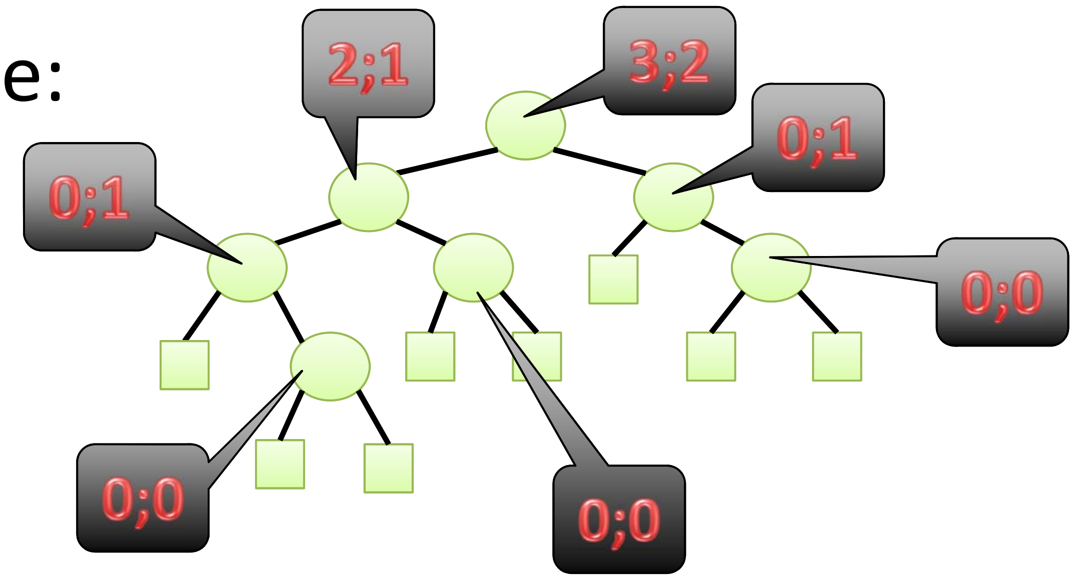
- There are some algorithms that maintain to take balance of tree in depth $O(\log n)$.
  - e.g., AVL tree, 2-3 tree, 2-color tree (red-black tree)

Georgy M. Adelson-Velsky
(1922–2014)

Evgenii M. Landis
(1921–1997)

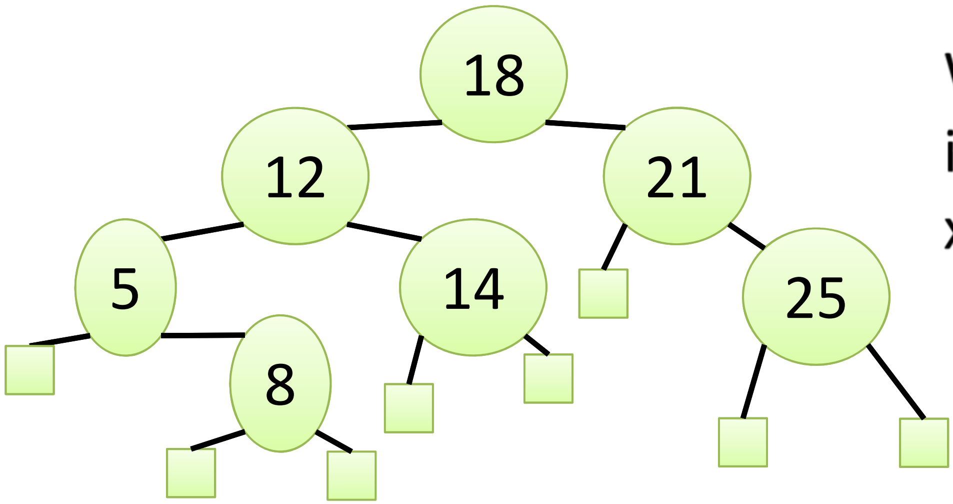
AVL tree [G.M. Adelson-Velskii and E.M. Landis ‘62]

- Property (or assertion): at each vertex, the depth of left subtree and right subtree differs at most 1.
- Example:



AVL tree: Insertion of data

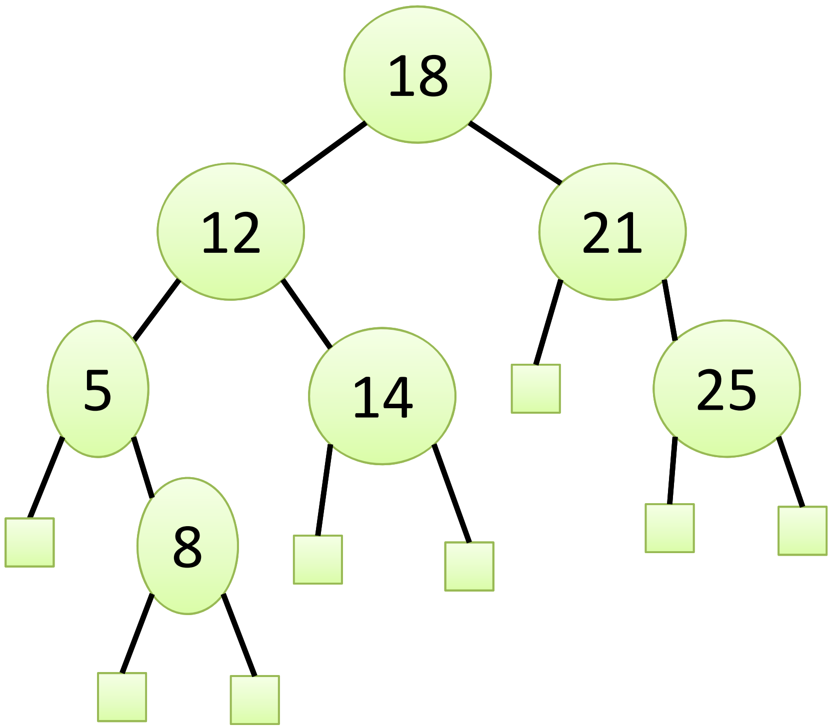
- Find a leaf v for a new data x
- Store data x into v (v is not a leaf any more)
- Check the change of balance by insertion of x
- From v to the root, check the balance at each vertex, and rebalance (rotation) if necessary.



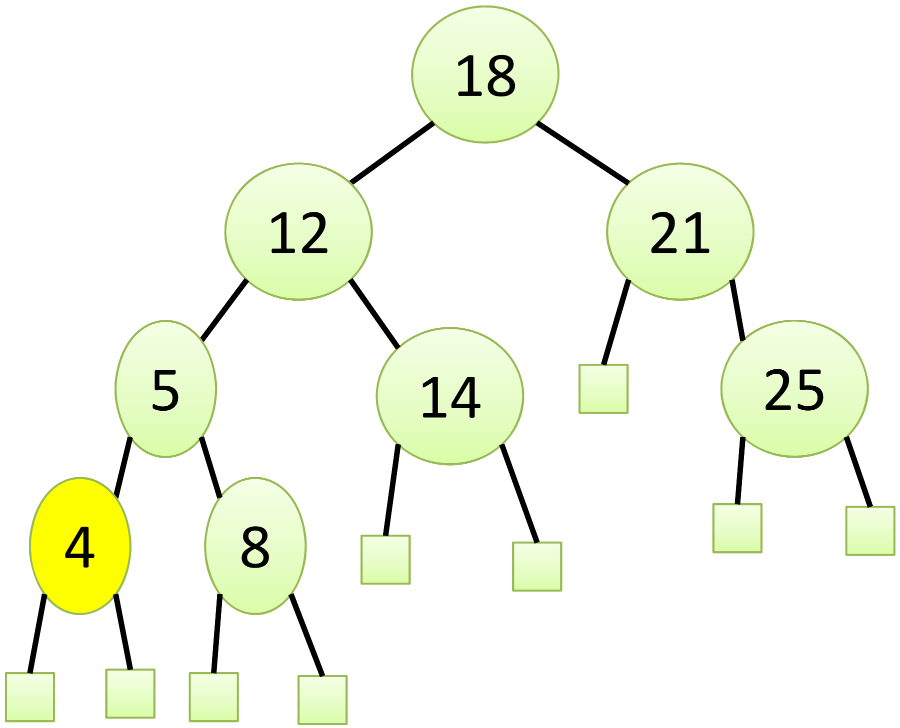
What happens if you insert x=4? How about x=10, x=20, x=23?

AVL tree: Insertion of data
Insert x=4

before



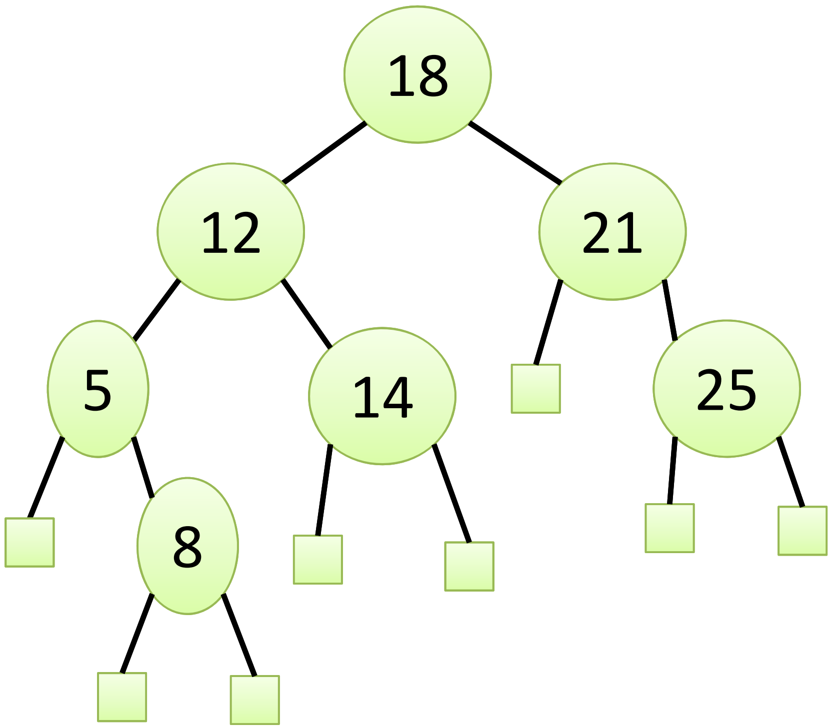
after



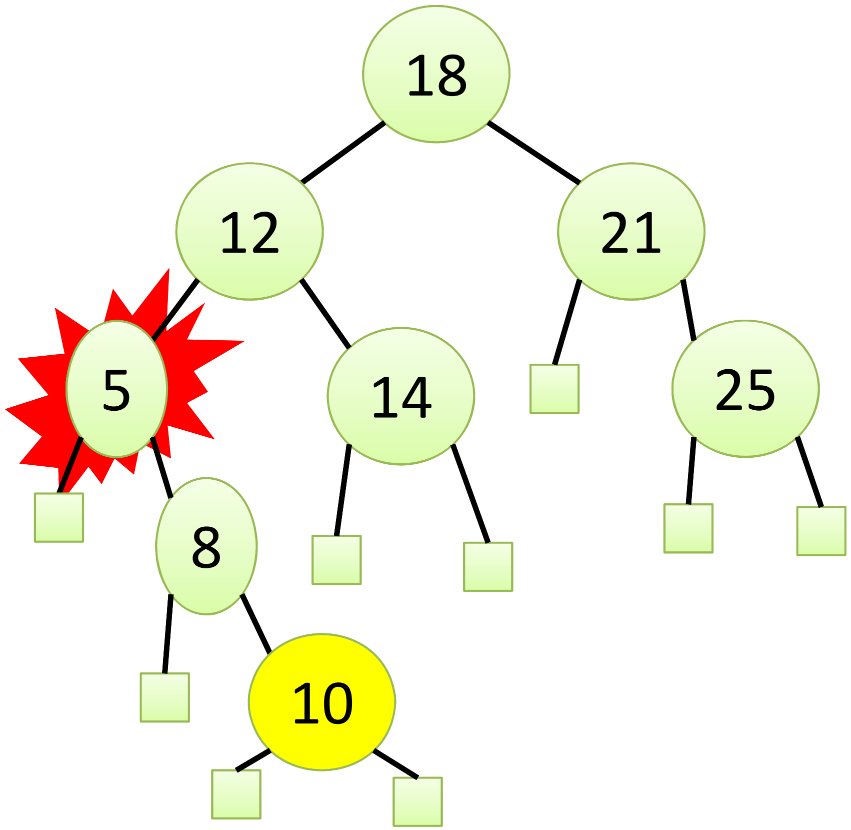
Balance: OK

AVL tree: Insertion of data
Insert x=10

before



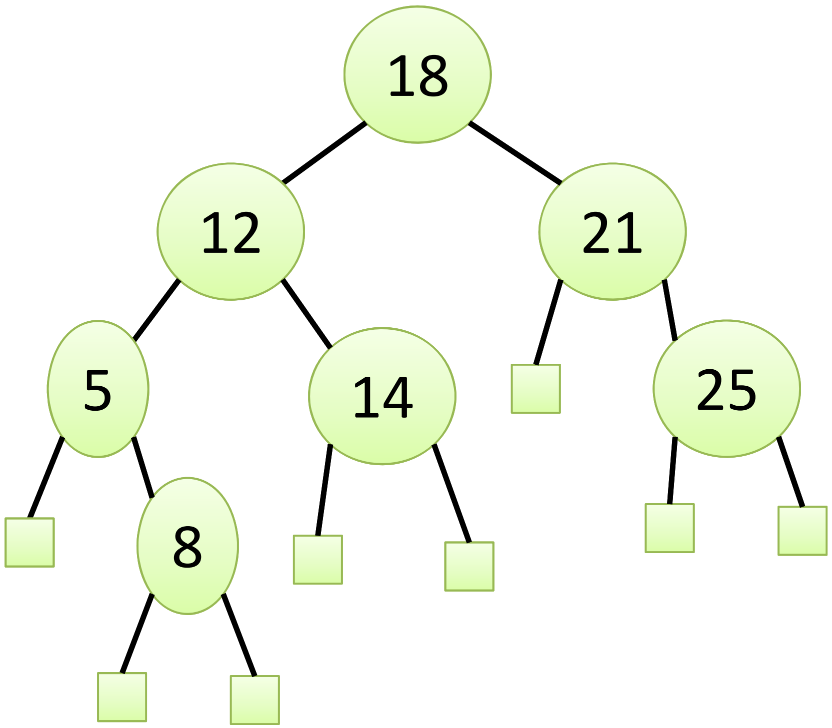
after



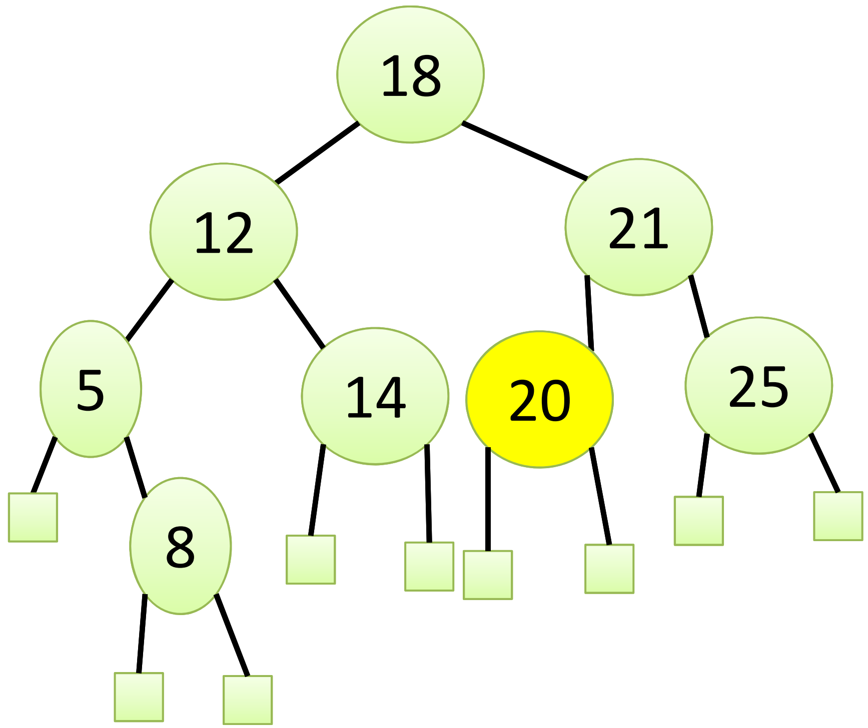
0;2@vertex 5

AVL tree: Insertion of data
Insert x=20

before



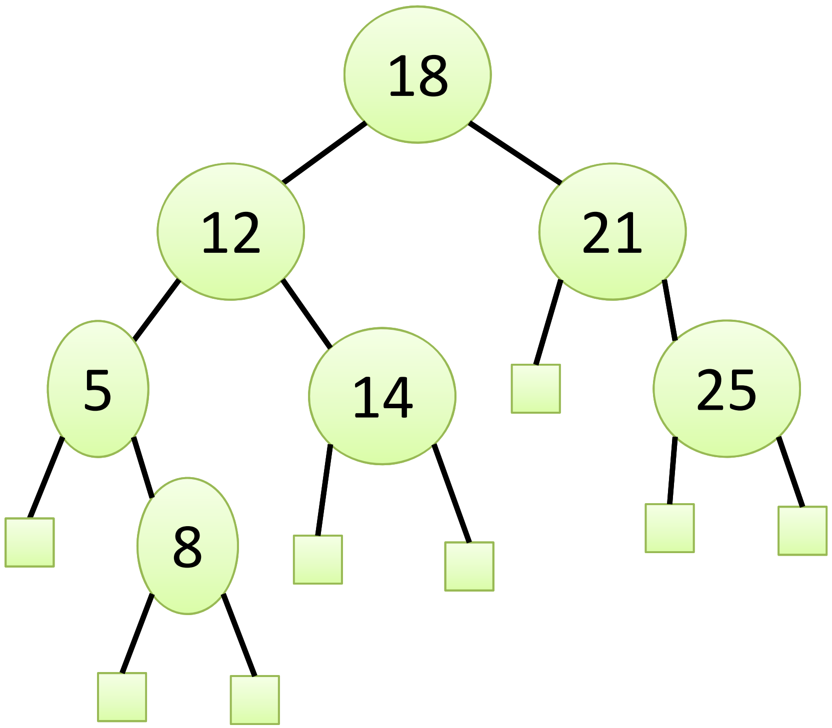
after



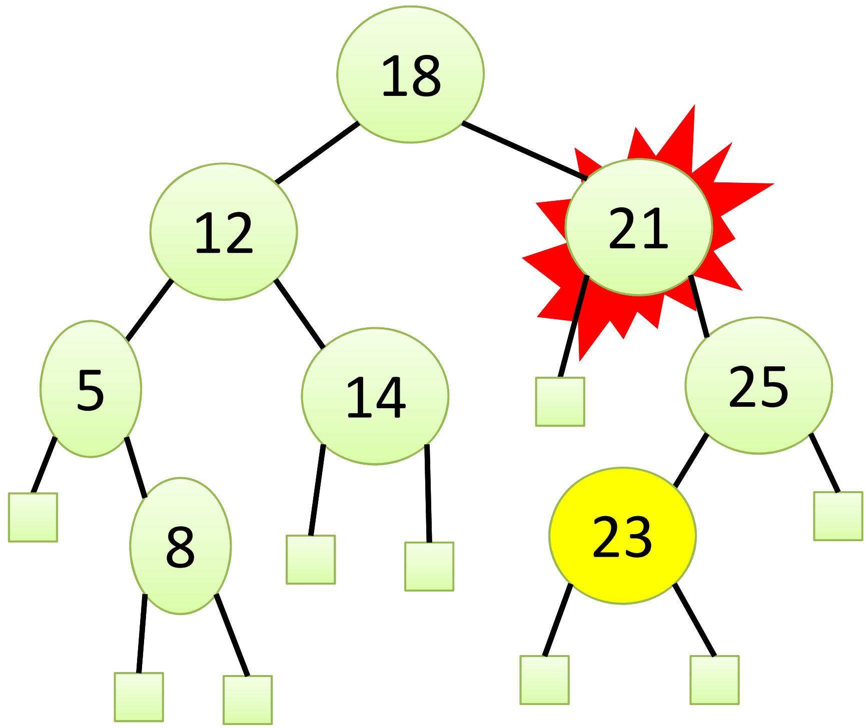
Balance: OK

AVL tree: Insertion of data
Insert x=23

before



after

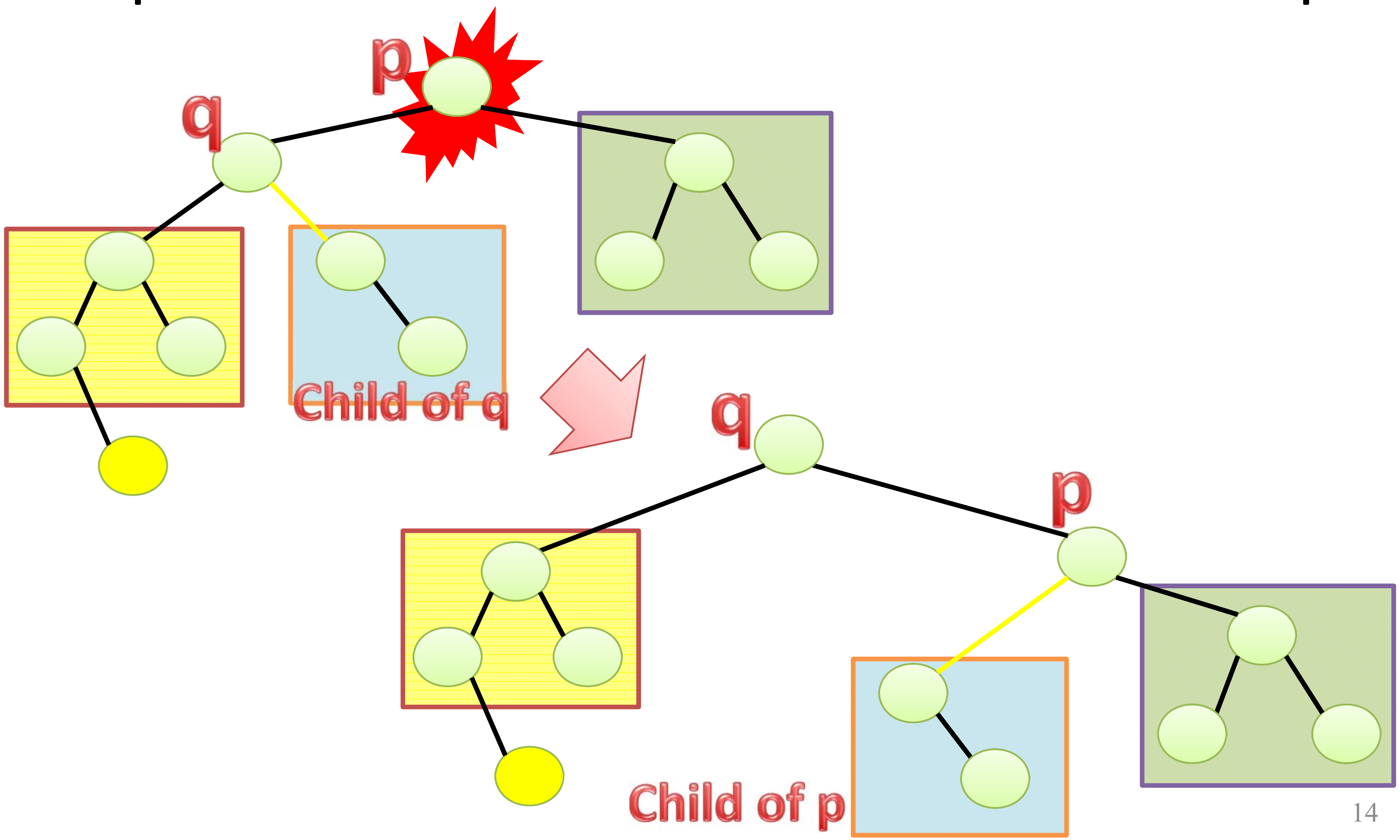


AVL tree: Rebalance by rotations

- "Rotate" tree vertices to make the difference up to 1:
  - Rotation LL
  - Rotation RR
  - Double rotation LR
  - Double rotation RL

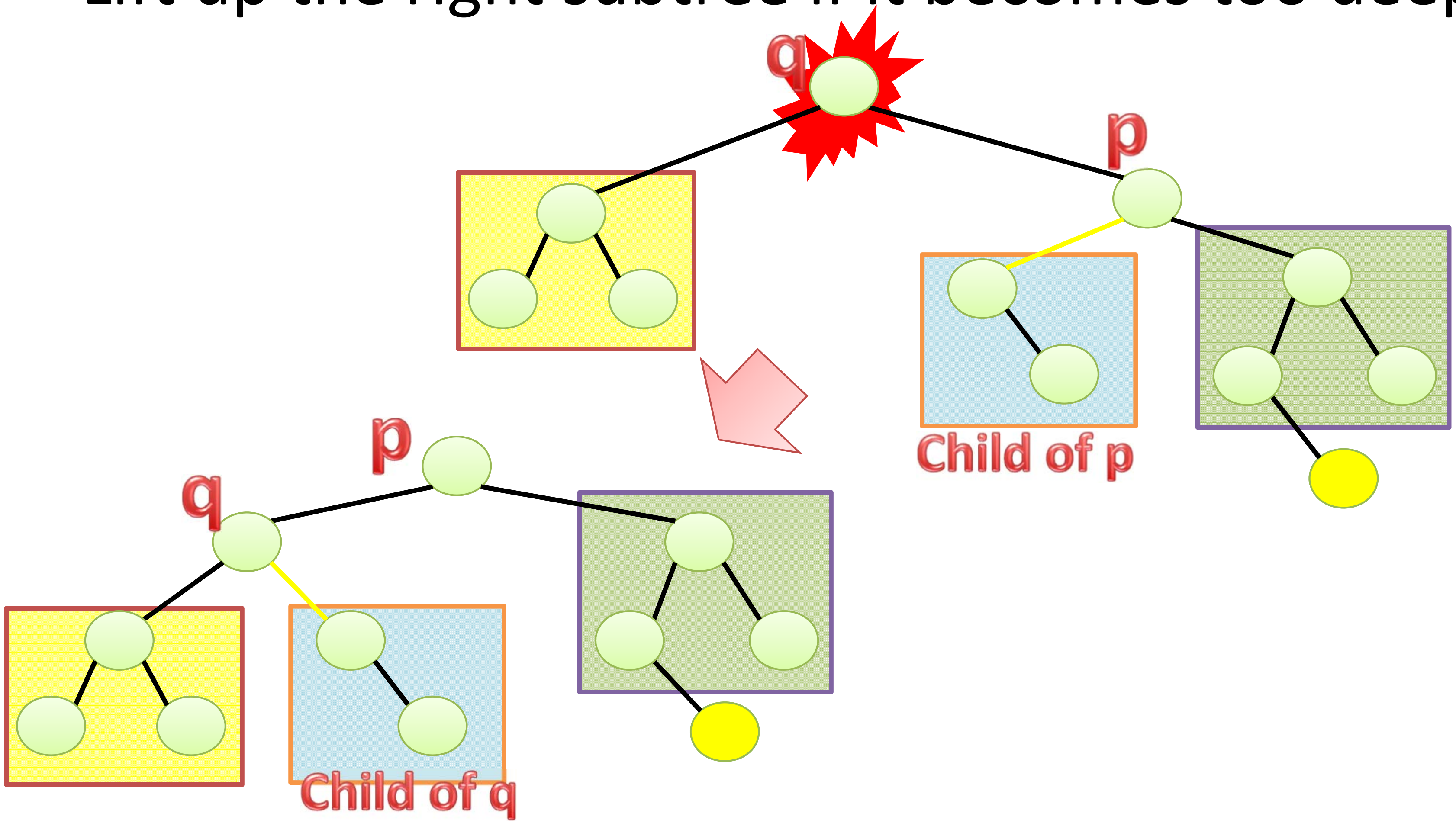
AVL tree: Rebalance by rotation: Rotation LL

- Lift up the left subtree if it becomes too deep



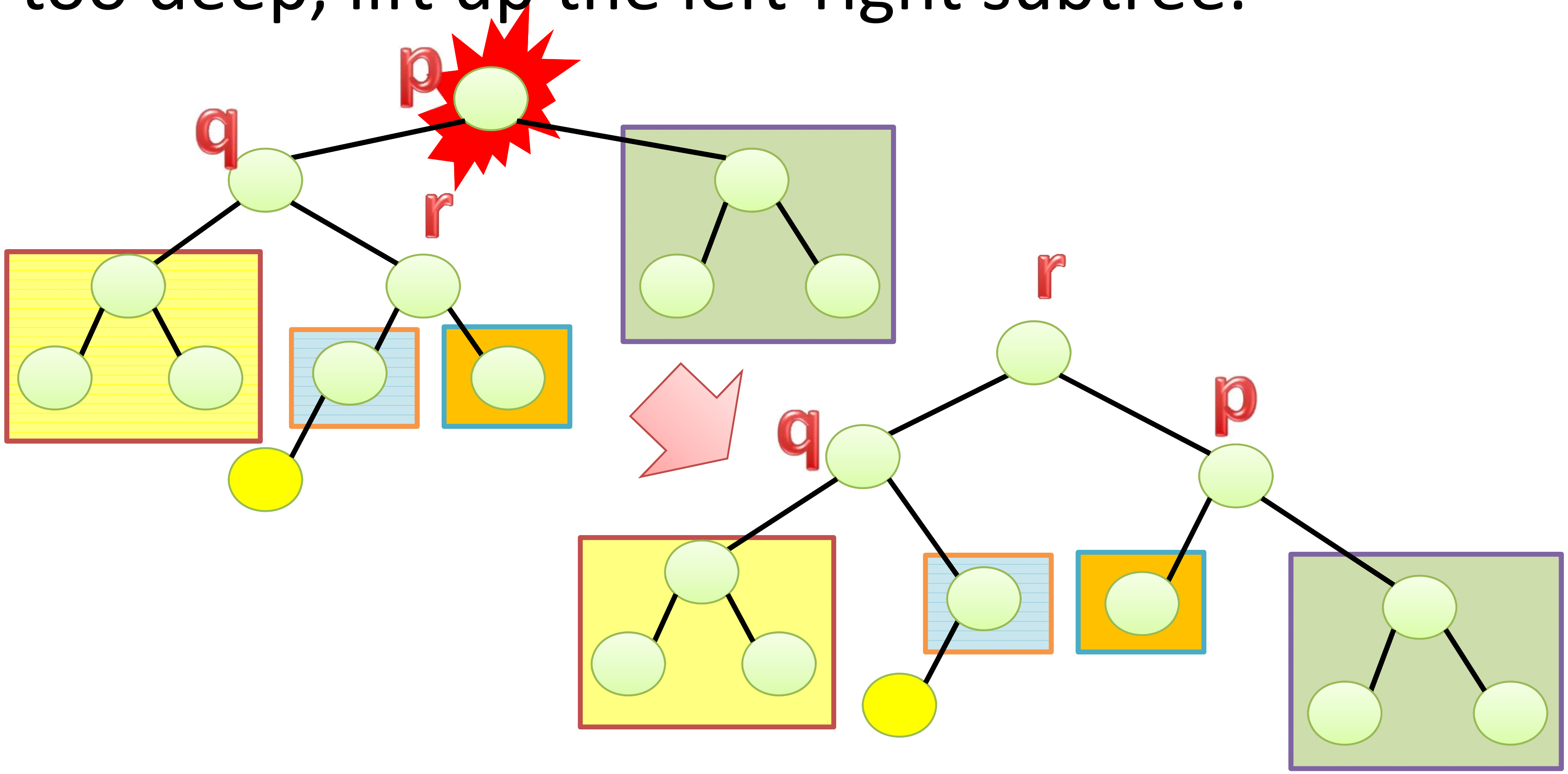
AVL tree: Rebalance by rotation: Rotation RR

- Lift up the right subtree if it becomes too deep



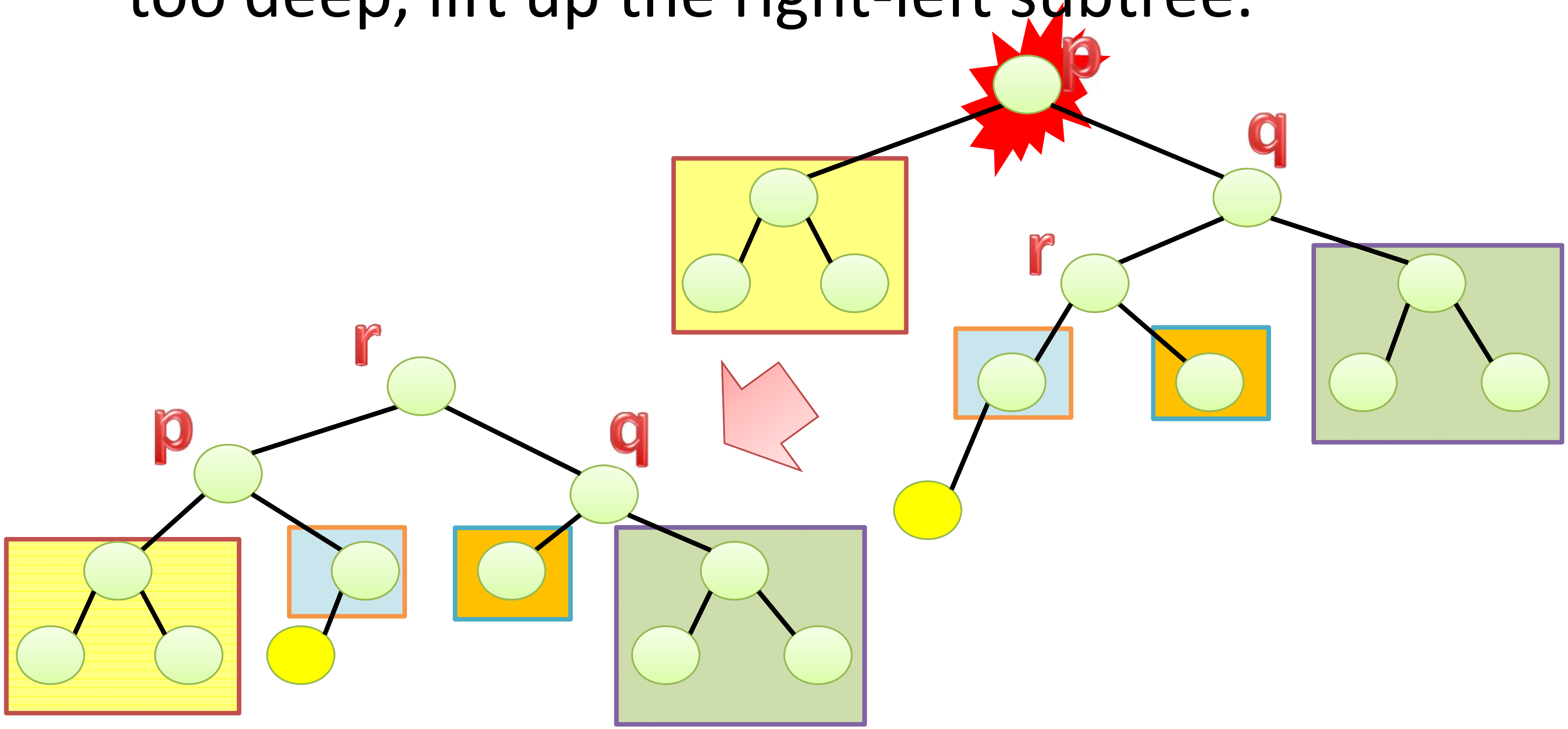
AVL tree: Rebalance by rotation: Double rotation LR

- When right subtree of left subtree becomes too deep, lift up the left-right subtree.



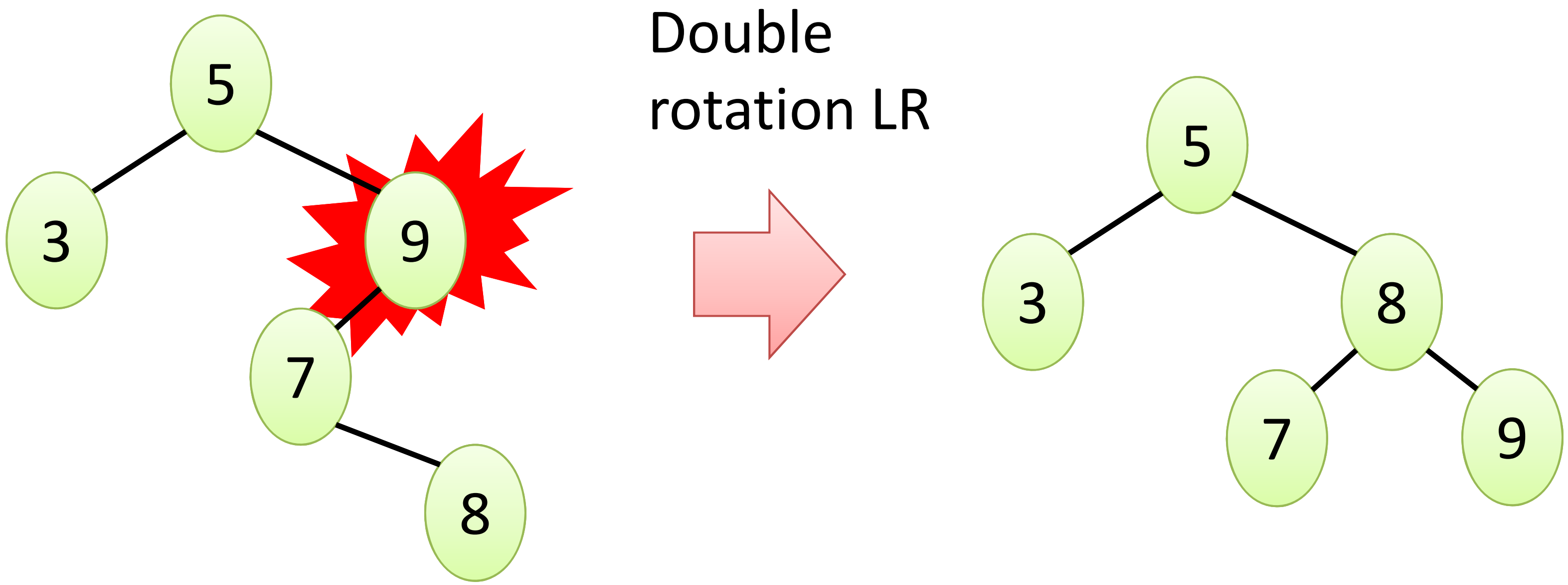
AVL tree: Rebalance by rotation: Double rotation RL

- When left subtree of right subtree becomes too deep, lift up the right-left subtree.



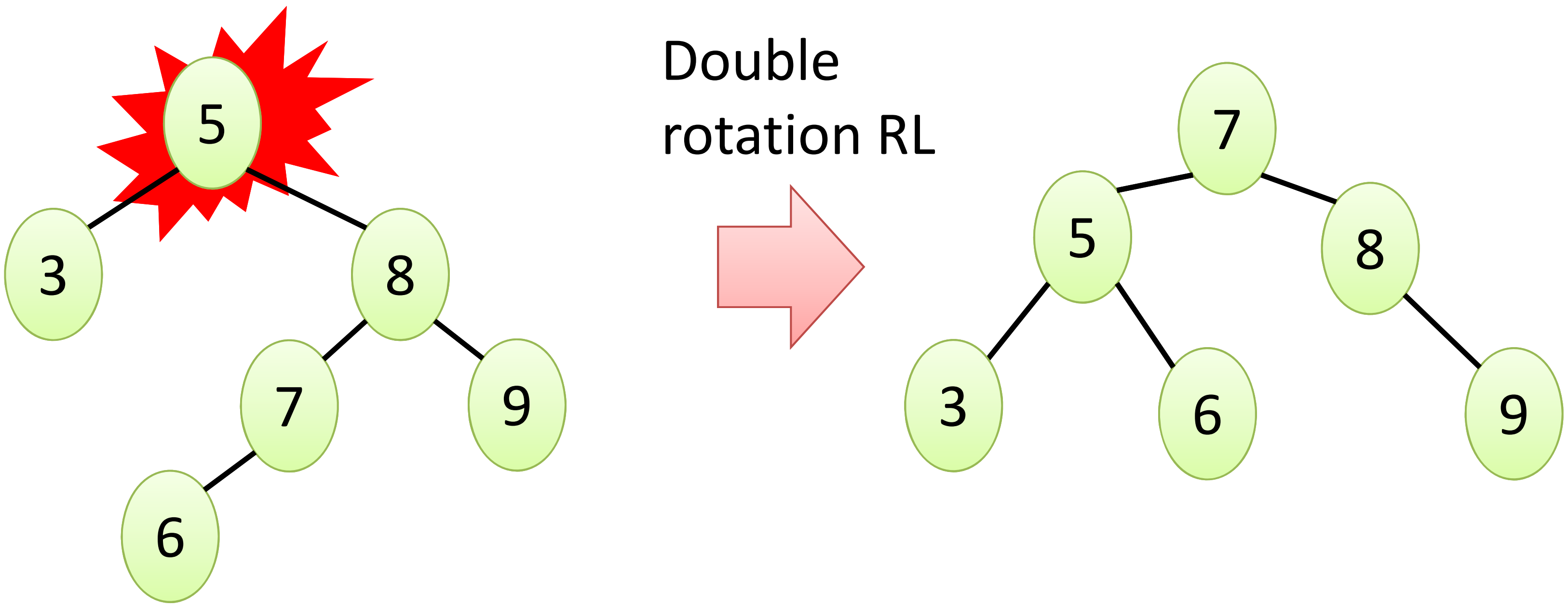
AVL tree: Example

- Insertion of 8



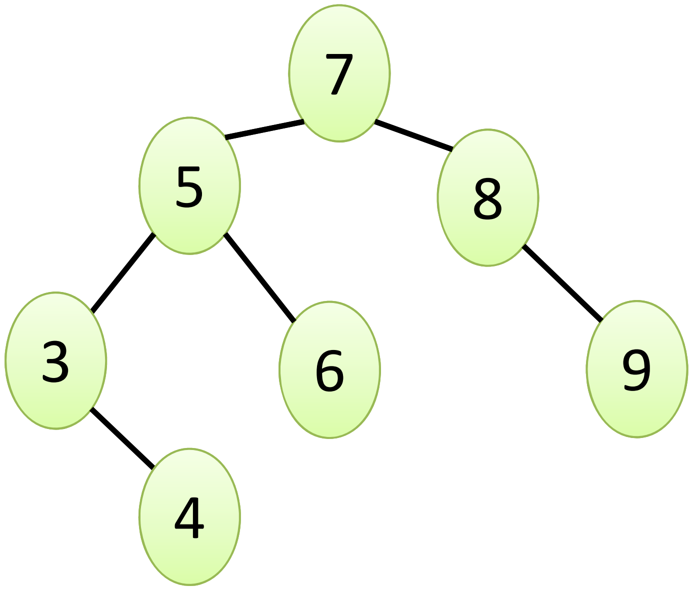
AVL tree: Example

- Insertion of 6



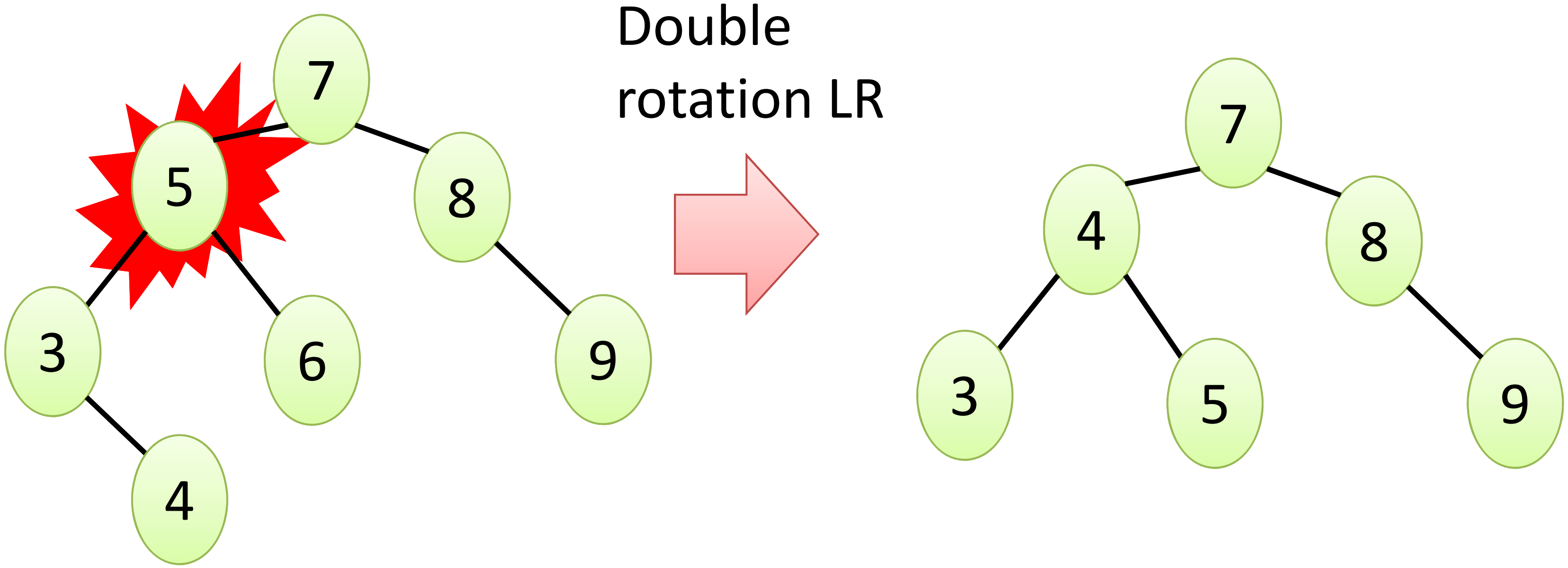
AVL tree: Example

- Insertion of 4



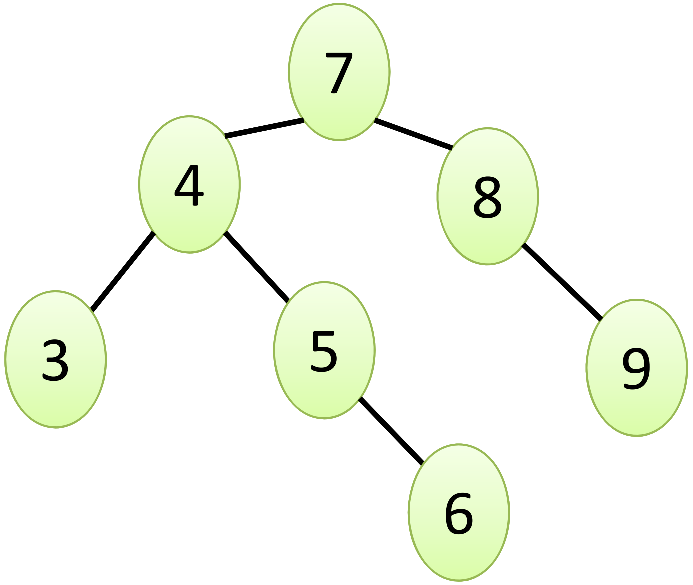
AVL tree: Example

- Deletion of 6



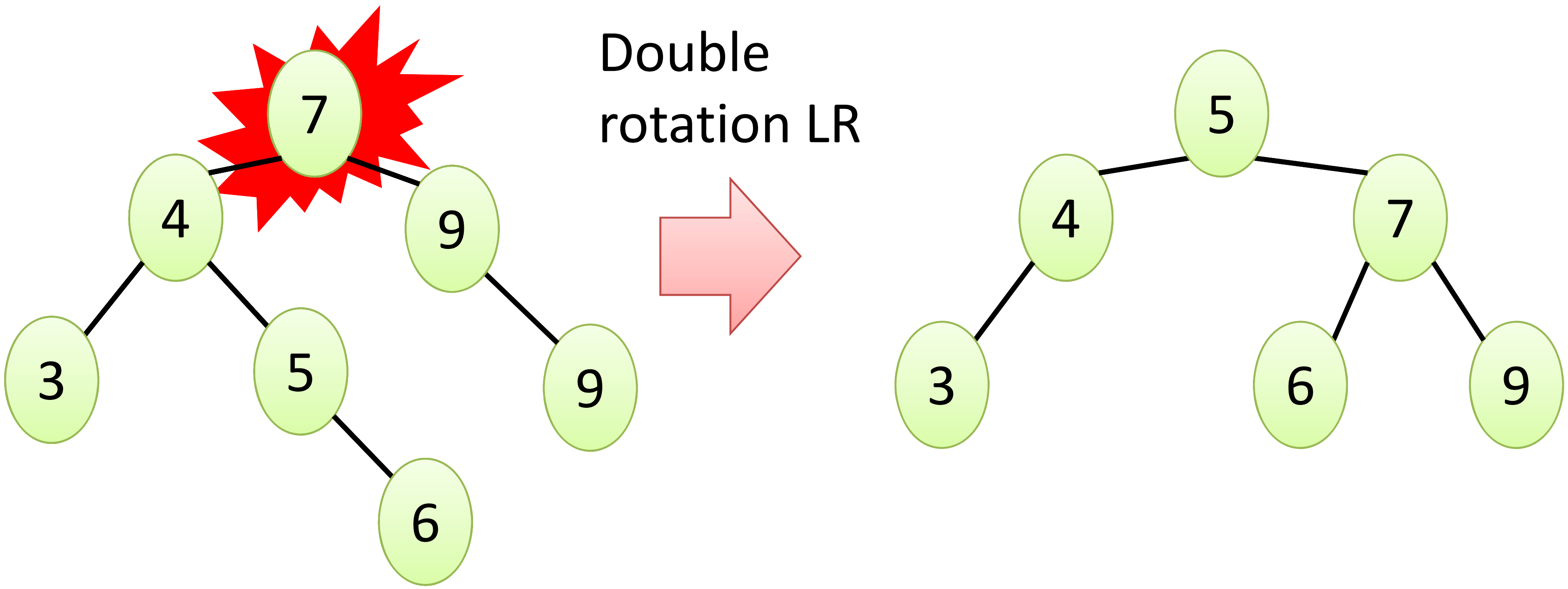
AVL tree: Example

- Insertion of 6



AVL tree: Example

- Deletion of 8



Time complexity of balanced binary search tree

- Search: $O(\log n)$ time
- Insertion/Deletion: $O(\log n)$ time
  – $O(\log n)$ rotations
  – Each rotation takes constant time

- In total, on a balanced binary search tree, every operation can be done in $O(\log n)$ time.
  (n is the number of data in the tree)