AVL Tree

Juan Zhai

juan.zhai@rutgers.edu

Insertion

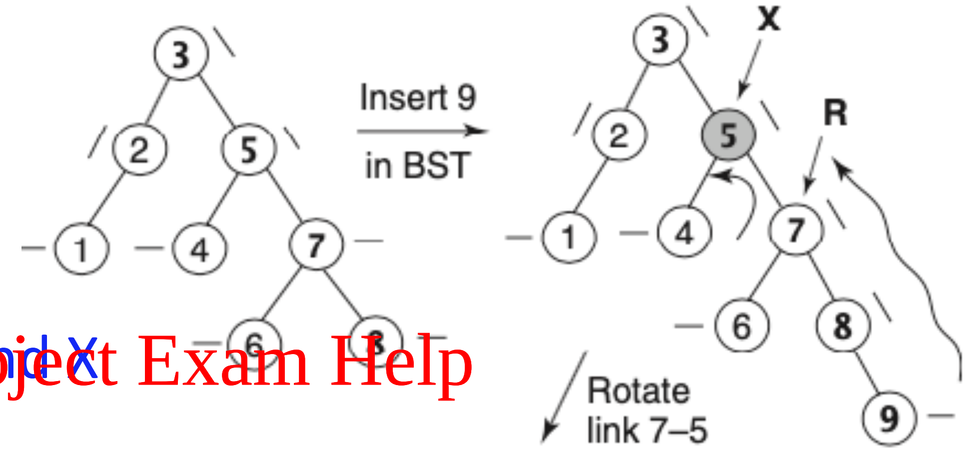
- Insert a value as in a regular BST by searching for its correct position.
- Backtrack from the inserted node up the chain of parents, updating the balance factor of each node along the way. Stop at a node along the way if it is unbalanced and rebalance the node.
  - Never stop and rebalance if no node along the backtracking path is unbalanced.
  - Once stop and rebalance a node, no need to continue backtracking. Insertion can terminate with the guarantee that the resultant tree is an AVL tree.

- X is the unbalanced node
- R is the root of the taller subtree of X after insertion

CASE 1: X and R have the *same* balance factors, both ‘/’ or ‘\’

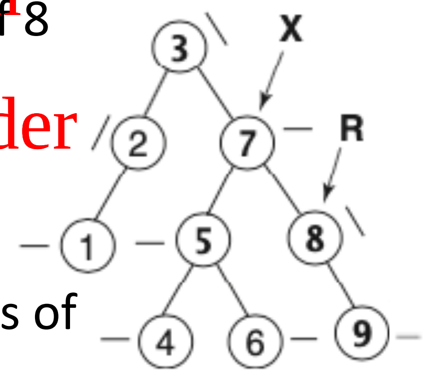
Rebalance:

1. Rotate about the link R-X
2. update balance factors of R and X



1. Insert 9 as right child of 9, set ‘—’
2. Backtrack the parent chain, change balance factor of 8 as ‘\’
3. Then change balance factor of 7 as ‘\’
4. Then stop at 5 because of imbalance
5. *Apply rotation on link 7-5* and update balance factors of 5 and 7
   1) 7 becomes right child of 5’s parent (i.e., 3)
   2) 5 becomes left child of 7
   3) The original left child of 7, namely 6, becomes the right child of 5. why?

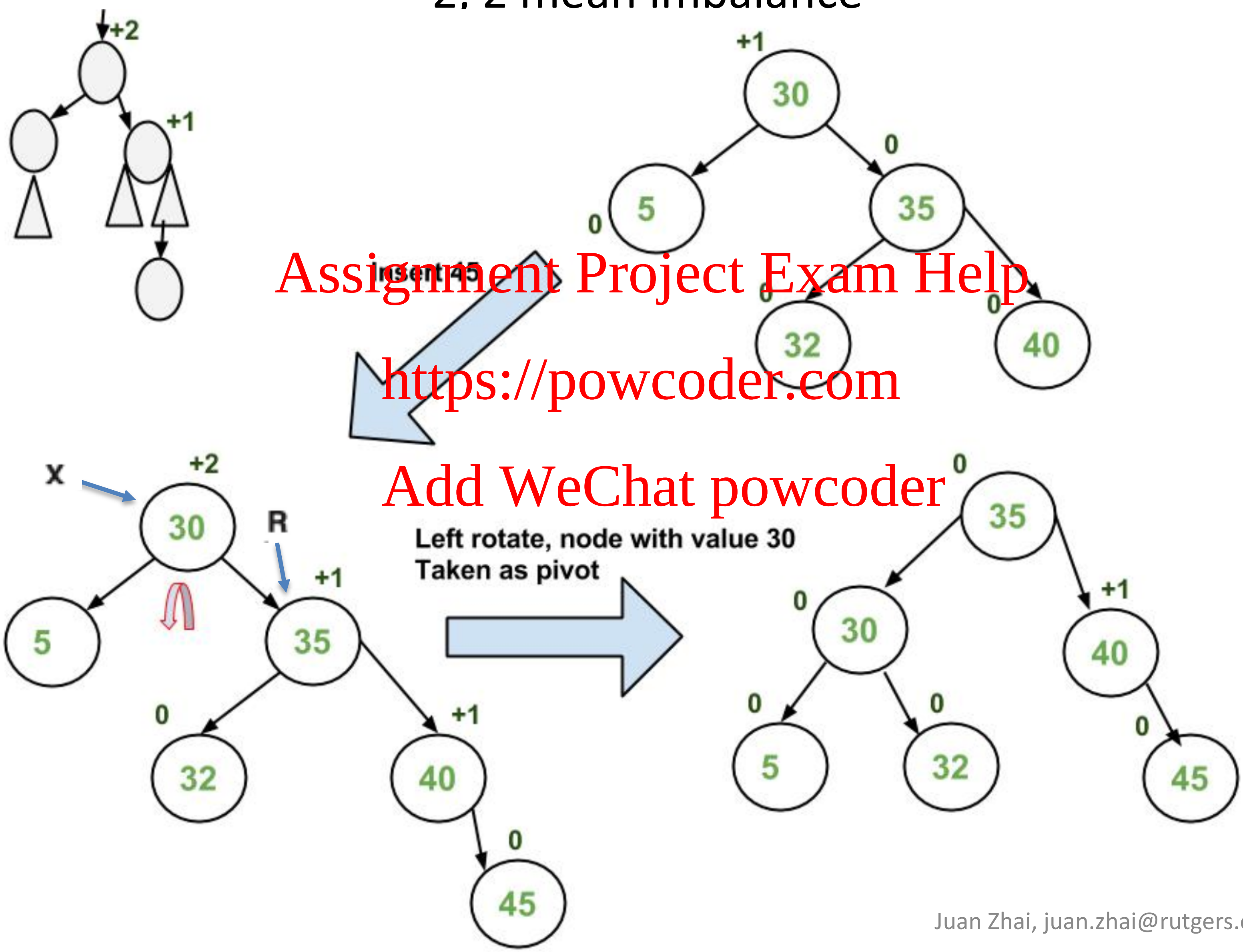
   6 is between 5 and 7, it has to be either the right child of 5 or the left child of 7, but 5 is now the left child of 7, so 6 has to be the right child of 5.



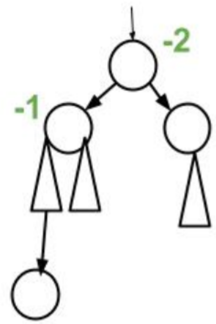
right right case: both X and R are ‘\’
Perform left rotation

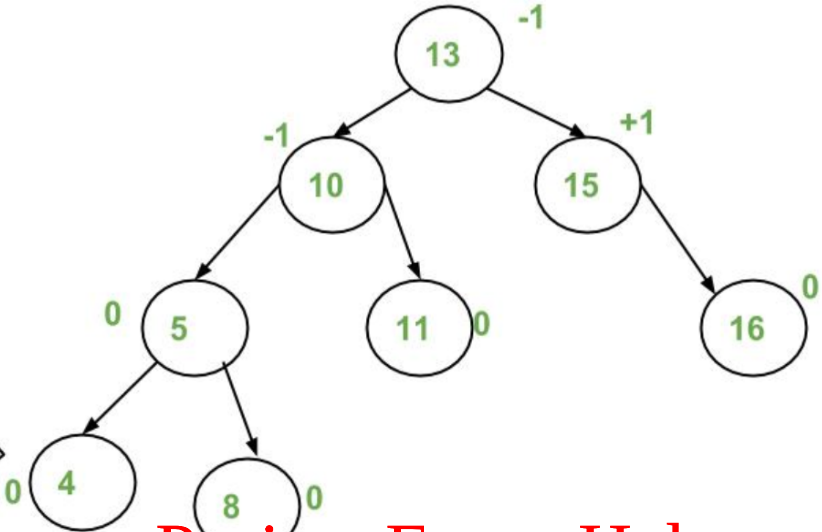
right right case:
Left Rotation

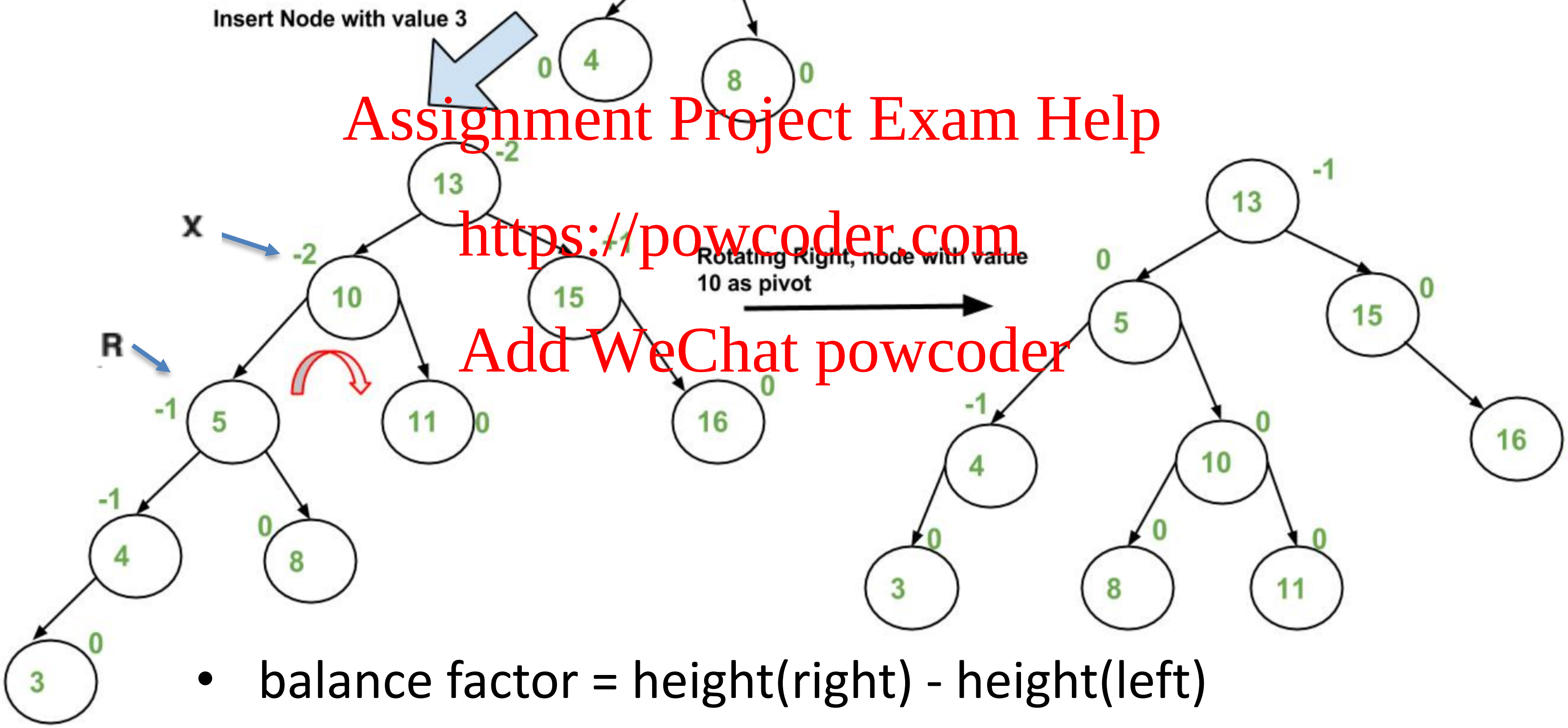
- balance factor = height(right) - height(left)
- 0, -1, 1 mean balance
- -2, 2 mean imbalance



left left case:
right rotation







- balance factor = height(right) - height(left)
- 0, -1, 1 mean balance
- -2, 2 mean imbalance

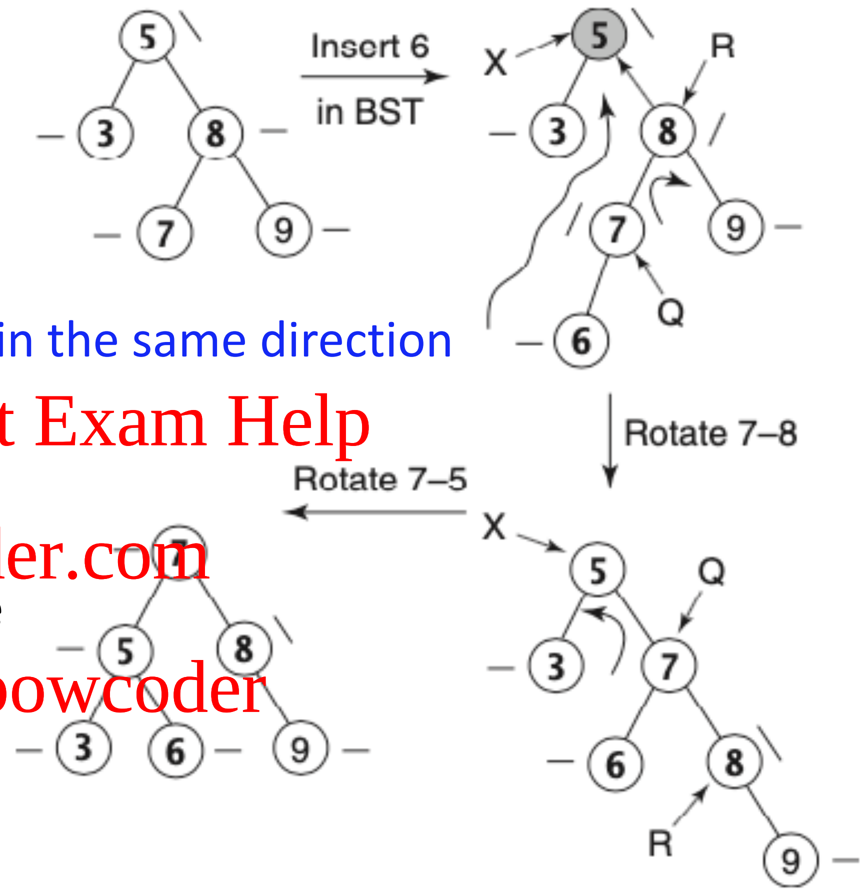
- X is the unbalanced node
- R is the root of the taller subtree of X after insertion
- Q is the root of the taller subtree of R

CASE 2: X and R have the *opposite* balance factors, one is '/' and the other one is '\'

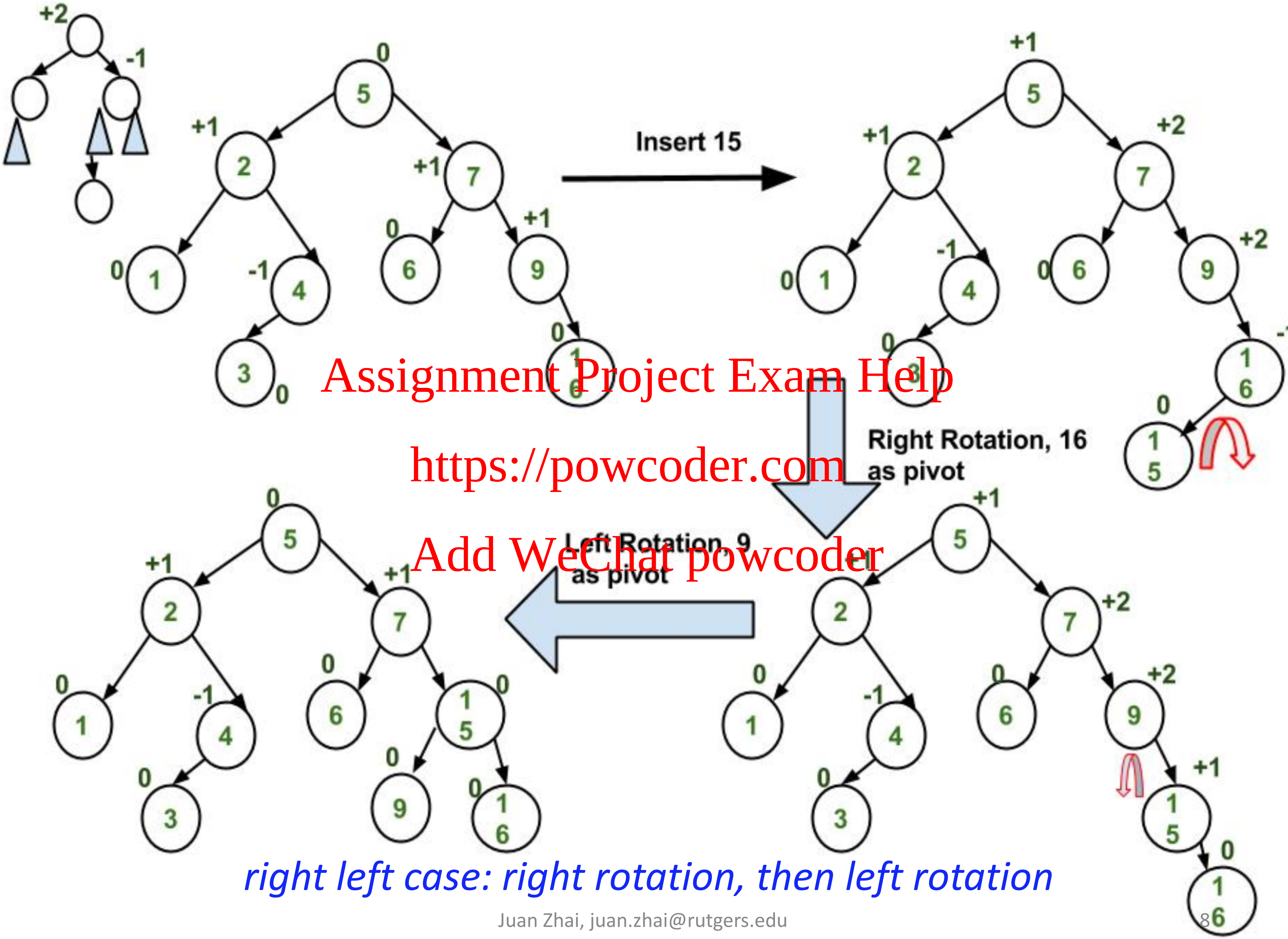
Rebalance:

1. Rotate link Q-R, which aligns X, Q and R in the same direction
2. Rotate link Q-X
3. Update balance factors of X, Q and R

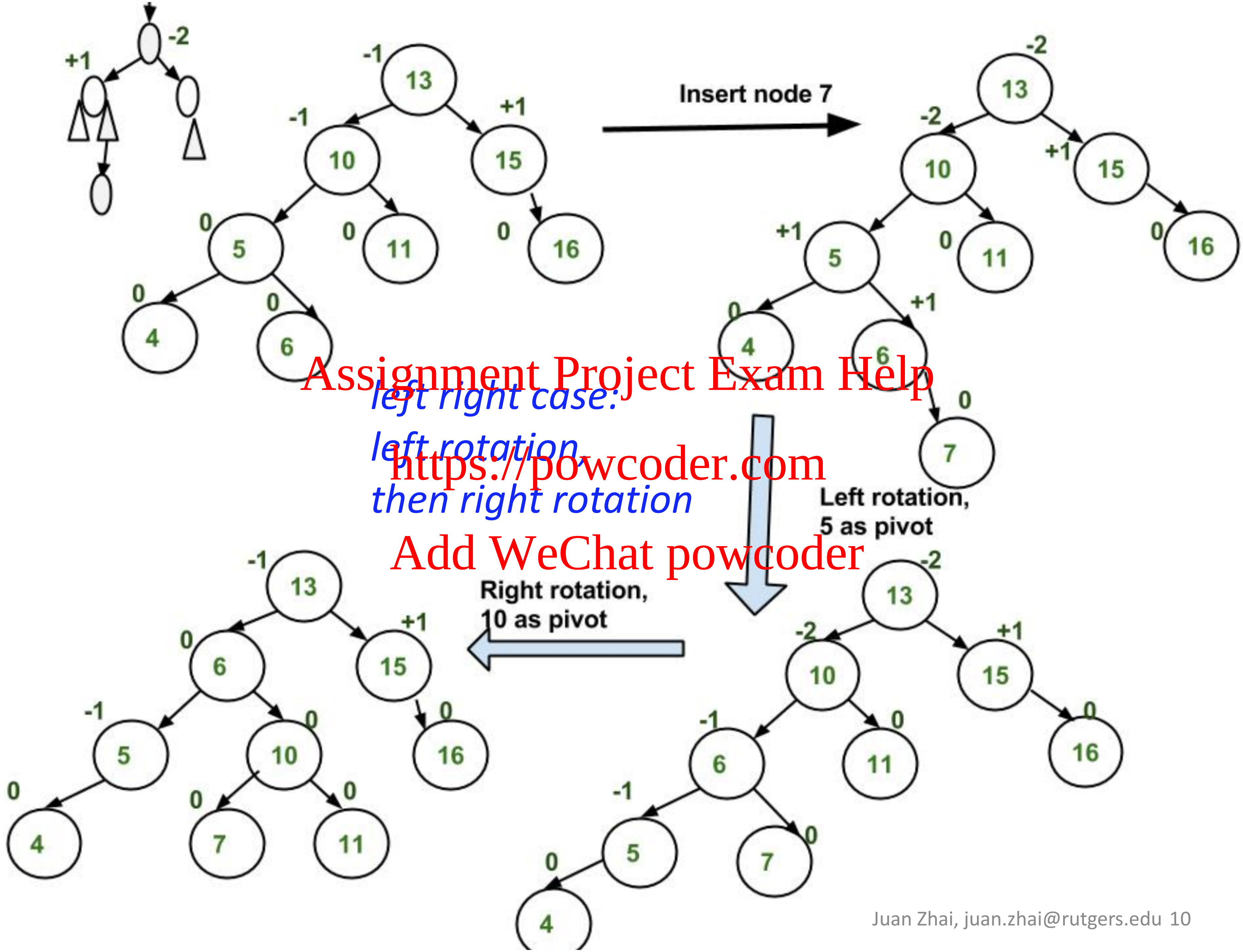
1. Insert 6 as left child of 7, set '—'
2. Backtrack the parent chain, change balance factor of 7 as '/'
3. Then change balance factor of 8 as '/'
4. Then stop at 5 because of imbalance, 5 has the balance factor that is opposite of its child in the taller subtree 8
5. *Apply rotation on link 7-8*, which aligns 5(X),7(Q),8(R) in the same direction
6. *Apply rotation on link 7-5*
7. Update balance factors of 5, 7, 8



right left case



right left case: right rotation, then left rotation



left right case:
left rotation,
then right rotation

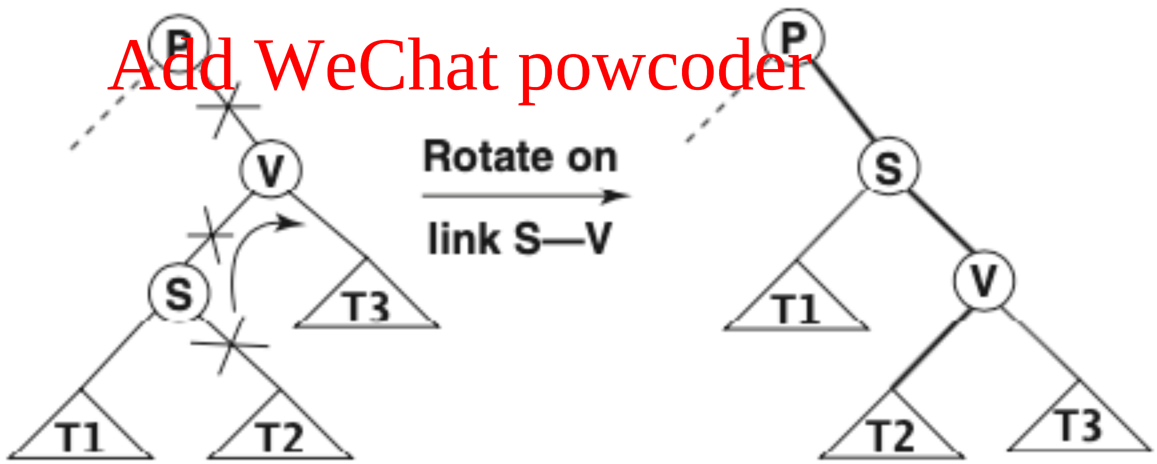
Deletion

- Uses rotation, has nothing new.
- Not introduced in the course.

Running time for rotation

- Rotate on link S-V, T1, T2 and T3 are AVL trees, any of them could be empty
  - Break three links: P → V, S → V, S → T2
  - Make three links: P → S, S → V, V → T2
  - In code: several assignments

  → O(1)



Running time

- AVL tree with n nodes, then the height is O(log *n*)
- Search: O(log *n*) when the data is in the leaf node or not in the tree
- Insertion: O(log *n*)
  - Search: O(log *n*)
  - Rebalance:
    - Search back to the root which takes O(log *n*)
    - Rotation: O(1), update some links

java.util.TreeMap

- Provides guaranteed log(n) time cost for the operations like containsKey, get, put and remove.