COL 106 Lecture 21

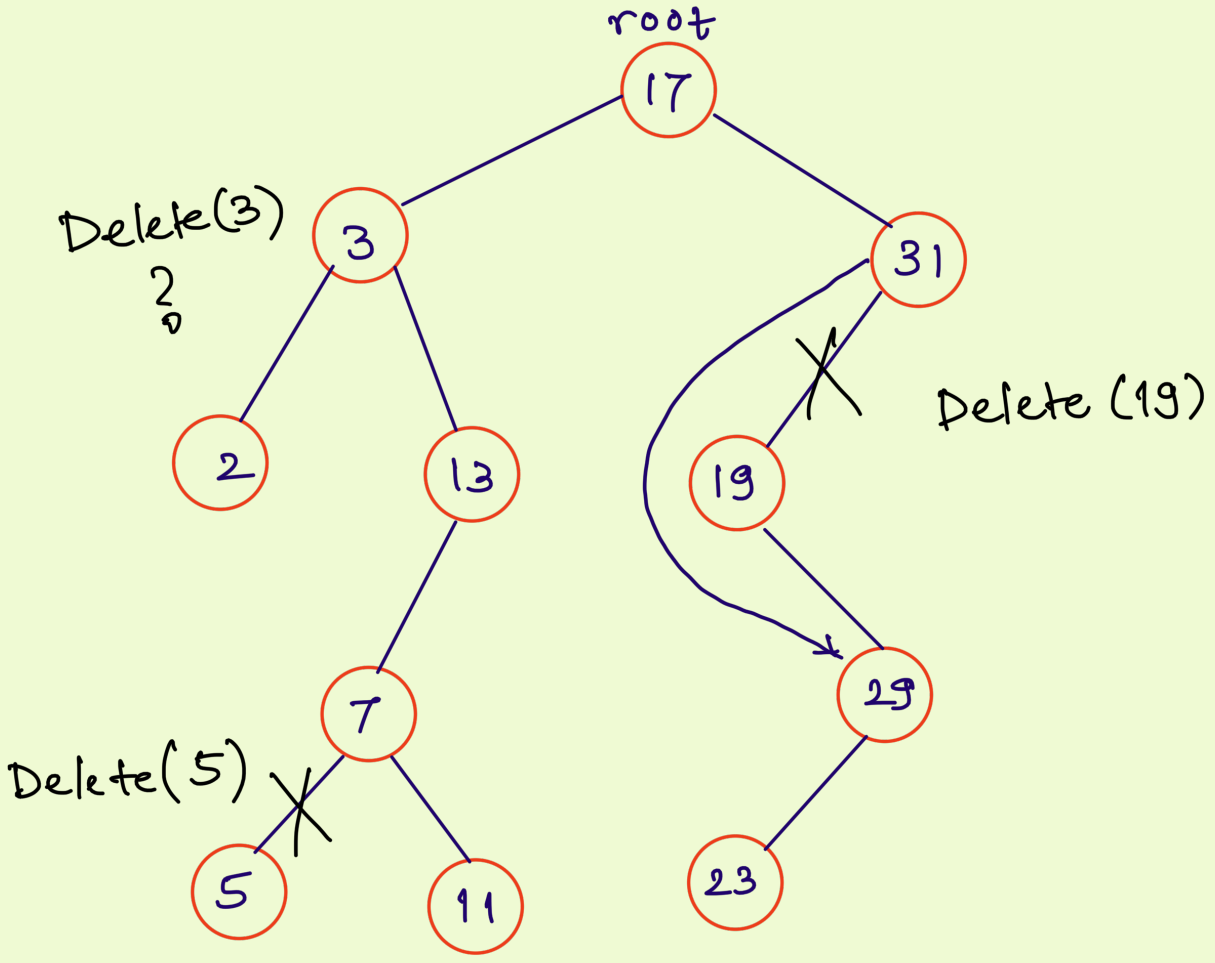
Topic : AVL Trees : Deletion

Modifier operations on AVL trees

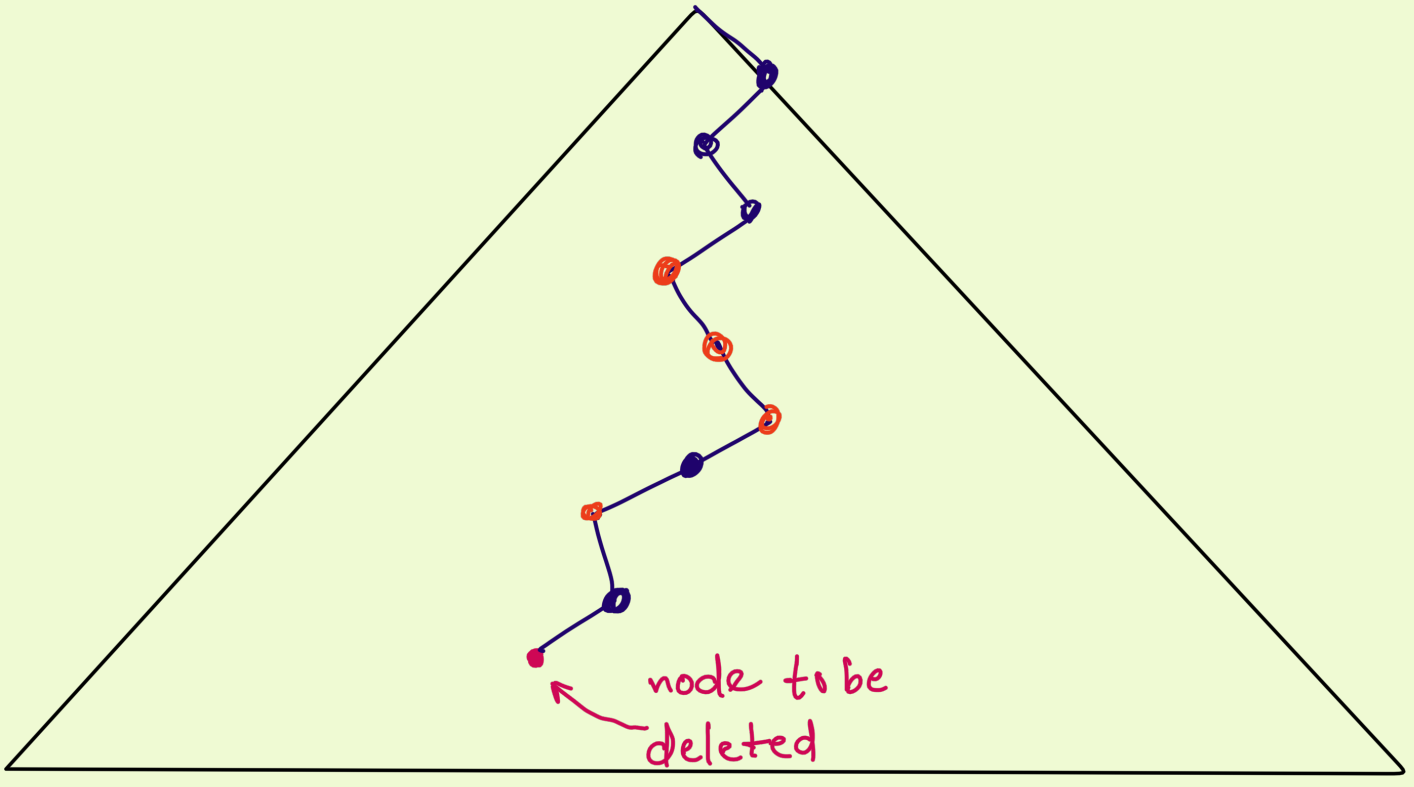
- ☑ Add element (Insertion)
  (last lecture)
- ☐ Remove element (Deletion)
  (today)

1. Insert / Delete like in a Binary Search tree
2. Fix imbalances
   ↳ How?

Recall: Deletion in a Binary Search Tree



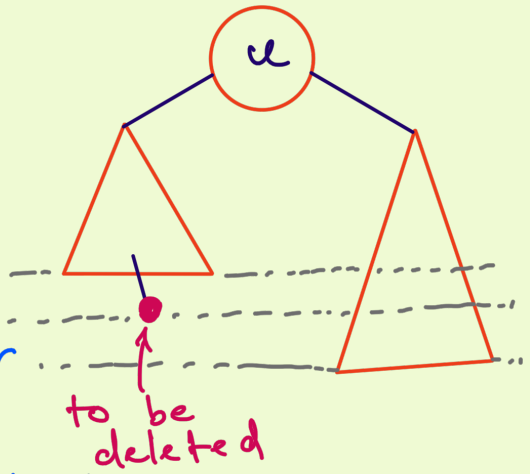
Imbalances after Deletion



Claim: Only ancestors of node to be deleted can become unbalanced.

Claim: At most one node becomes unbalanced after a BST-deletion in an AVL tree.

Proof Suppose u gets unbalanced after BST-delete. Before deletion, subtrees of u had unequal height and deletion happened in u's shorter subtree



$\therefore$ ht of subtree rooted at u doesn't change due to deletion.

$\Rightarrow \forall$ v ancestor of u : ht of subtree rooted at v doesn't change due to deletion.

$\Rightarrow \forall$ v ancestor of u other than u: v remains balanced after deletion.

AVL Tree Deletion Algorithm (?)

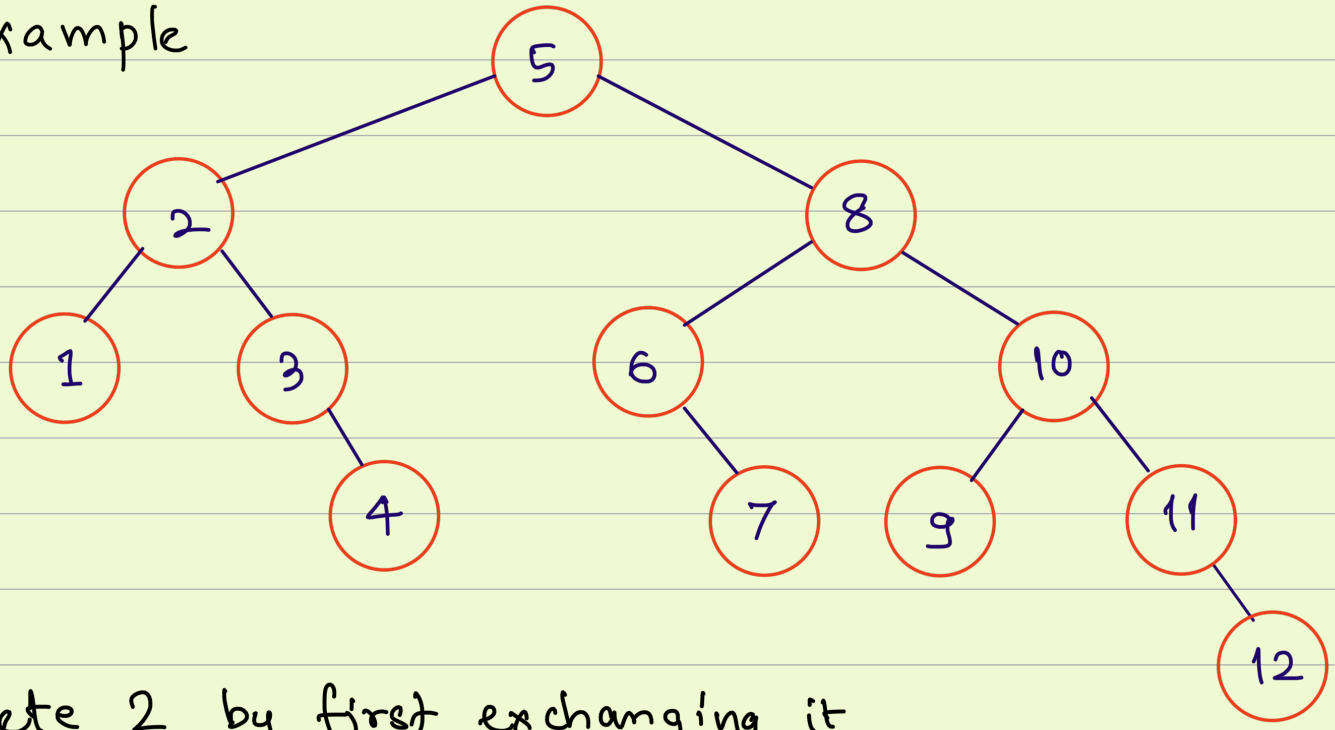
AVLDelete(x)

1. BSTDelete (x).
2. $p \leftarrow$ Parent of the node deleted in step 1.
3. For $q = p$ to root:
   If q unbalanced:
   Rebalance q
   Return.

Incorrect!

Rebalancing q can make its parent unbalanced.

Example



Delete 2 by first exchanging it with 1, its predecessor.
Fix the resulting cascading imbalances.

AVL Tree Deletion Algorithm.

AVLDelete(x)

1. BSTDelete (x).

2. p ← Parent of the node deleted in step 1.

3. For q = p to root:
   If q unbalanced:
      Rebalance q
   Update heights

} O(1)

Running time: O(logn)