11. Trees

Dr. Swati Agarwal

Agenda

Balanced Binary Trees

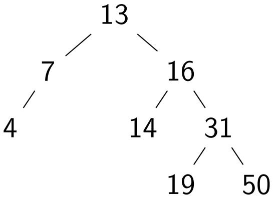
1. Balanced Binary Trees

- We have seen different trees with worst case complexity $\mathcal{O}(n)$ where n is the number of nodes in the tree. (happens in the case of Skew Trees).
- We try to reduce the complexity by imposing restrictions on the height of the tree and preserve the inductive structure of a search tree after insertion/deletion operation.
- A balanced tree is one in which, at each node, the sizes of the left and right subtrees differ by at most one.
- **It is difficult to maintain this definition without incurring a heavy re-balancing cost.**

1.1 Height Balanced Binary Trees

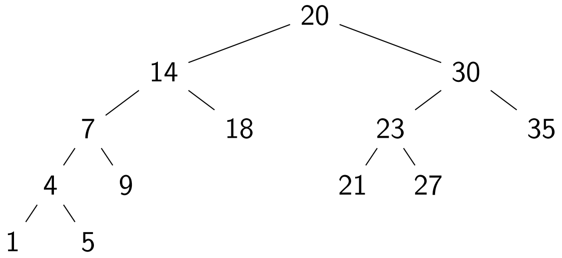
- A more flexible definition will be that the heights of the left and right subtrees differ by at most one.
- The balance factor value can be $\{-1, 0, 1\}$
- Such trees are called Height-Balanced trees or AVL trees represented with $HB(k)$ where k represents the Balance Factor.

1.2 Height vs. Size Balanced Binary Trees



- For node 13, the size-based balance factor is 3 since the number of nodes in left and right sub-trees are 2 and 5 respectively.
- While, the height-based balance factor is 1 since the height of left and right sub-trees are 2 and 3 respectively.
- Given tree is height balanced but not size balanced.

1.3 Calculate Balance Factor in AVL Tree



2. Height of Binary Balanced Trees

- To use height-balanced trees, we have to ensure that height of the tree is logarithmic in the number of nodes in a tree.
- Rephrase the problem and find that for a given height h, what is the size $S(h)$ of the **smallest** height-balanced tree with height h?

```
h   S(h)      T(h)

0    1         *

1    2          *
               /
              *

2    4          *
               / \
              *   *
             /
            *
```

```
h   S(h)      T(h)

3    7          *
               / \
              *   *
             / \   \
            *   *   *
           /
          *
```

2. Height of Balanced Binary Tree

$S(1) = 1$
$S(2) = 2$
$S(3) = 4$
$S(4) = 7$
...

$S(h) = S(h-2) + S(h-1) + 1, \quad h > 2$
is very similar to that for the Fibonacci numbers
$F(k) = F(k-1) + F(k-2), \quad k > 2.$

- In the case of Fibonacci numbers, we can show that $F(k)$ is exponentially large with respect to k.
- Analogously, we can show that $S(h)$ grows exponentially with respect to h. This means that even the "most skewed" height-balanced tree with n nodes has height logarithmic in n.

2.1 Number of Nodes in AVL Tree

- Minimum number of nodes in AVL Tree:
  $S(h) = S(h-2) + S(h-1) + 1, \quad h > 2$
  By solving the above recurrence equation:
  $S(h) = \mathcal{O}(1.618^h) \Rightarrow h = 1.44 \log n = \mathcal{O}(\log n)$

- Maximum number of nodes in AVL Tree:
  $S(h) = S(h-1) + S(h-1) + 1, \quad h > 2$
  The above expression defines the case of full binary tree. By solving the above recurrence equation:
  $S(h) = \mathcal{O}(2^h) \Rightarrow h = \log n = \mathcal{O}(\log n)$

3. Rotations

- AVL Tree is a self-balancing binary search tree.
- When the tree structure changes (insertion or deletion operations), the tree needs be modified to maintain the AVL properties.
- This balancing can be done via rotations.
- Since during insertion or deletion operations, only one node is modified, this can only increase/decrease the height of some sub-tree by 1.
- If the AVL tree property is violated at node X that means that height of $left(X)$ and $right(X)$ subtrees differ by exactly 2.
- Rotations is the technique to restore the AVL tree property.

Types of Violations

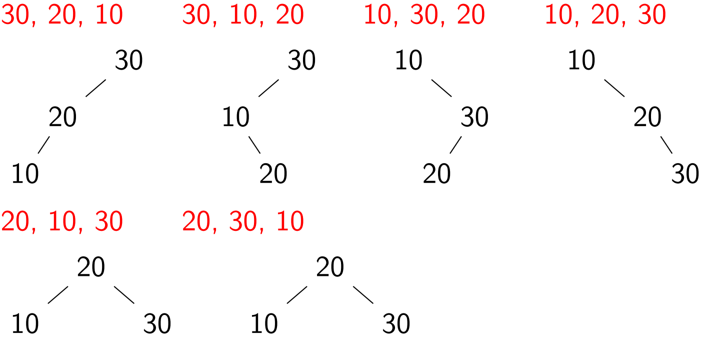
An insertion operation into the

- **left subtree of left child** of X. LL Rotation
- **right subtree of left child** of X. LR Rotation
- **left subtree of right child** of X. RL Rotation
- **right subtree of right child** of X. RR Rotation

After inserting the element in an AVL Tree:

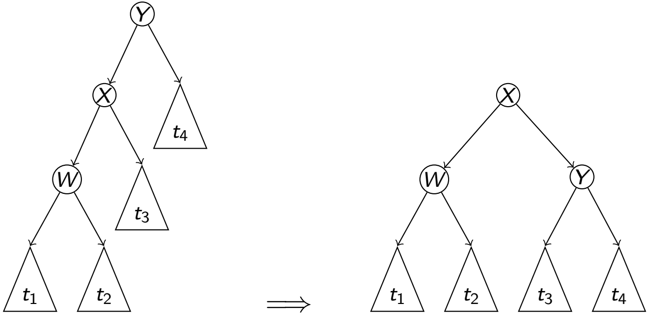
- Walk up to the root node
- find first node that is violated
- maintain the AVL property for that node using rotations
- repeat.

Example: same set keys

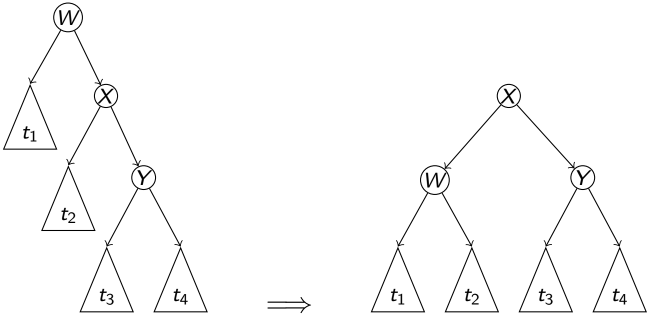


For left and right skewed trees, single rotations are enough. However, for zigzag trees, single rotation converts the sub-tree into it's mirror image depicting the no use of rotations. For zig-zag shaped trees, we need double rotations.

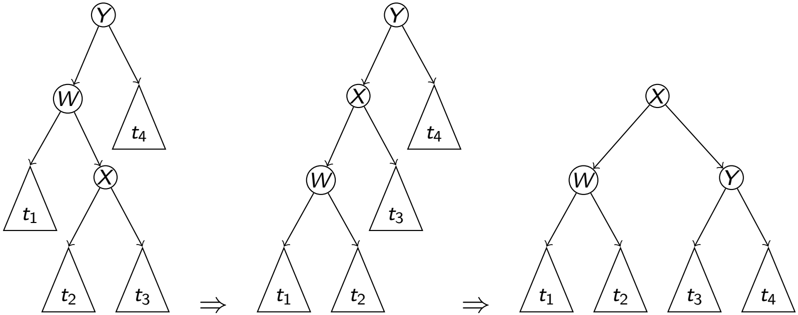
Single Rotation- LL



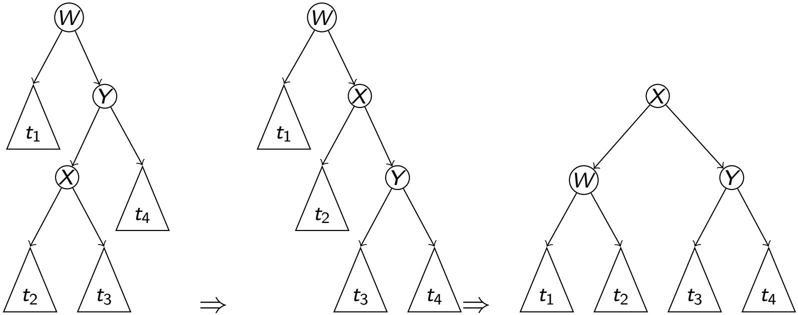
Single Rotation- RR



Double Rotation: LR



Double Rotation: RL



Practice Question

Input sequence: 30, 20, 10, 40, 50, 60, 25, 35, 28, 38

The above sequence will have all four rotations.

Deletion in AVL Tree

- Apply BST_Delete to find and delete the element
  - Leaf Node
  - Node with one child/sub-tree
  - Node with both children/sub-trees
- Maintain the AVL properties using rotations.