Chapter 8: Search Trees (Part 2)

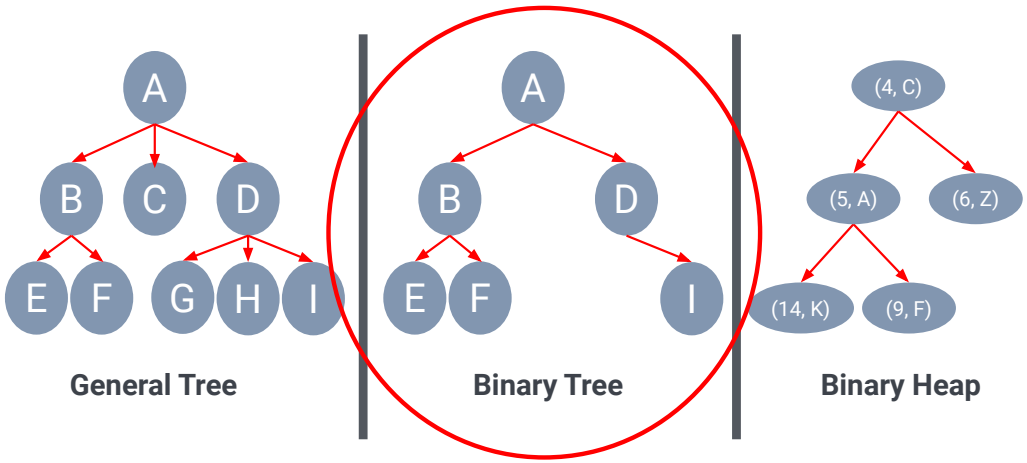
Dr. Sirasit Lochanachit

Outline

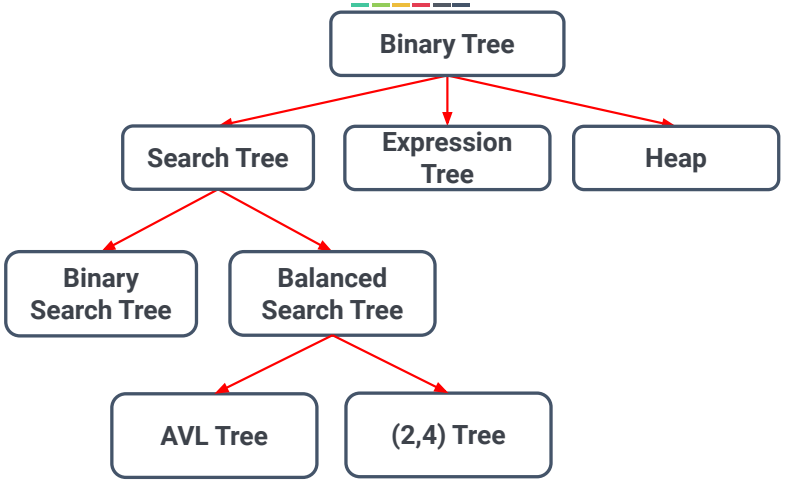
AVL Trees:

- Definition, properties and methods (Insert and rotation)
- Balancing Algorithms and Operation examples

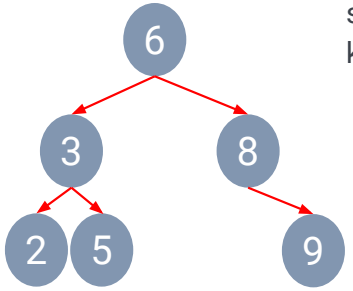
Types of Trees (Revisited)



Types of Binary Trees (Revisited)



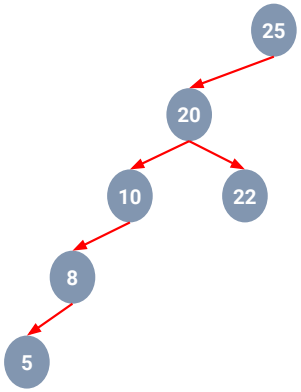
Binary Search Tree (Revisited)



- A **binary search tree (BST)** is a binary tree that stores an ordered sequence of elements or pairs of keys and values and has the following properties [1]:
  - All keys/elements in the ***left subtree*** are **less than** their ***root***.
  - All keys/items in the **right subtree** are **greater than** or **equal to** their ***root***.
  - Each subtree itself is a binary search tree.
- The example uses BST for storing a set of integers.

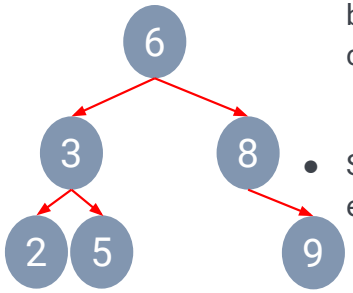
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Binary Search Tree (Revisited)



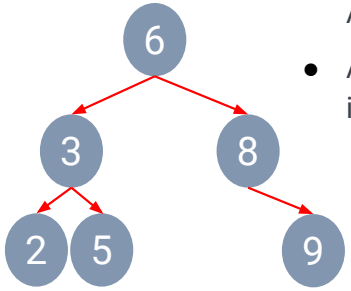
- Running time of inserting node is also proportional to the **height of tree** (i.e. $\log_2 n$ or n) == O(h).
- A **balanced search tree** has the same number of nodes in both left and right subtree.
  - Worst-case performance is $O(\log_2 n)$.
- Inserting keys in sorted order would construct an **imbalanced tree**.
  - Provides poor performance of $O(n)$.
- Other operations' performances are also limited by the height of the tree.

Balanced Binary Search Tree



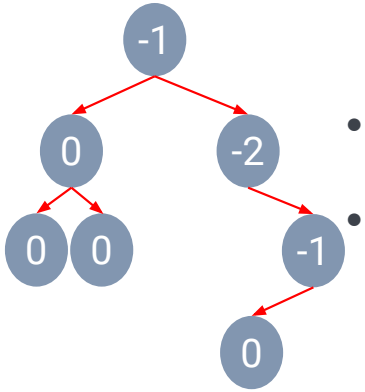
- A **balanced binary search tree (BST)** maintains the balance through a rotation operation which consequently provides a better performance.
  - A child is rotated to be above its parent.
- Several types of binary tree that automatically ensure balance
  - AVL tree
  - Splay tree
  - Red-black tree

AVL Tree



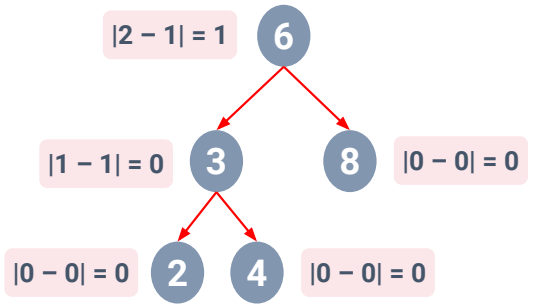
- AVL tree is named after its inventors: G.M. Adelson-Velskii and E.M. Landis.
- AVL tree introduces a **balance factor** for each node in the tree.
  - The height difference between the left and right subtree (H_{left} - H_{right})
  - If a subtree is left heavy, then the factor is > 0.
  - If a subtree is right heavy, then the factor is < 0.
  - If the factor is 0, the tree is perfectly balanced.

Balance Factor in AVL Tree

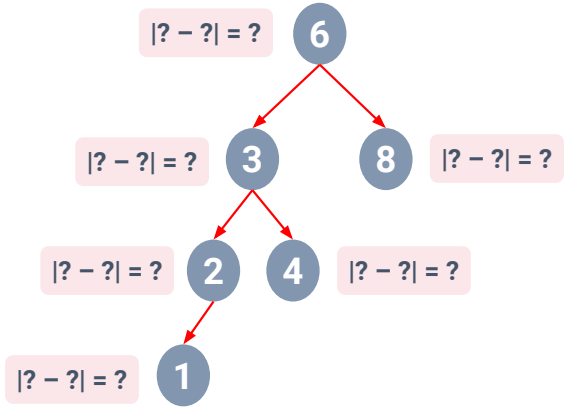


- AVL tree is considered to be balanced when the balance factor is -1, 0, or 1.
  - $|H_{left} - H_{right}| <= 1$
- AVL tree uses **trinode restructuring**, involving reconfigurations of three nodes.
- When new node is inserted into the tree,
  - Balance factor of a new leaf is zero.
  - Balance factor of its parent (and possibly every ancestors) has to be updated (+1 or -1 depends on left or right child).

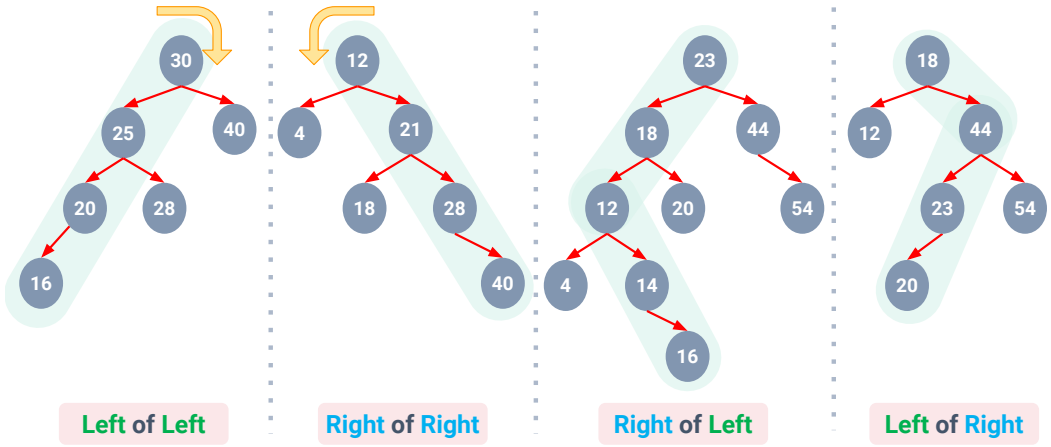
Balance Factor in AVL Tree



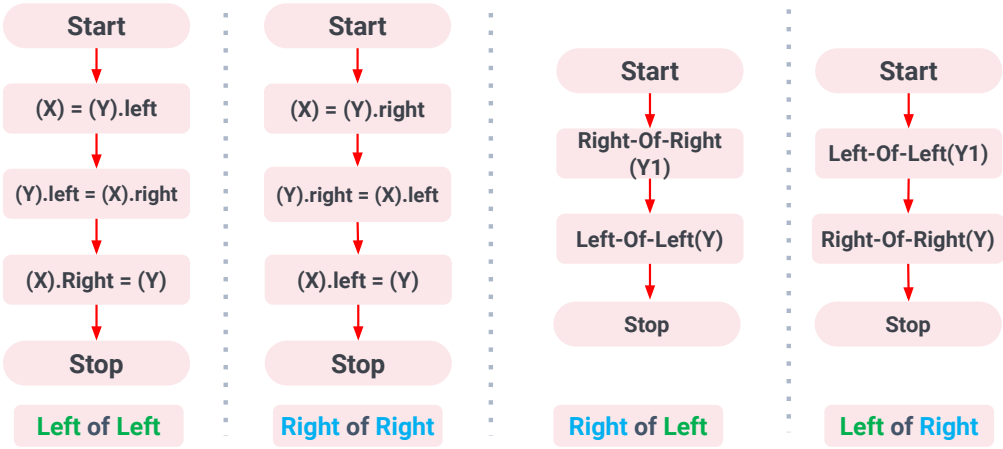
Balance Factor in AVL Tree



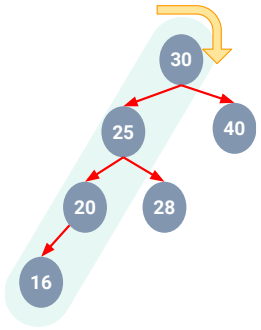
Balancing AVL Tree



Balancing AVL Tree

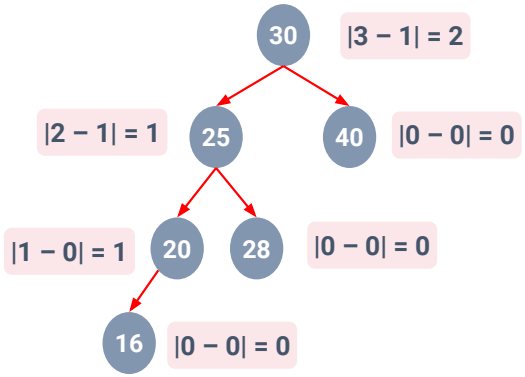


Balancing AVL Tree (LoL)



Left of Left

Balancing AVL Tree (LoL)

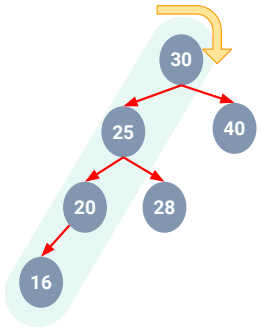


A tree is **left-heavy** with a balance factor of 2 at the root.

Require a **right rotation**.

Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

Right Rotation

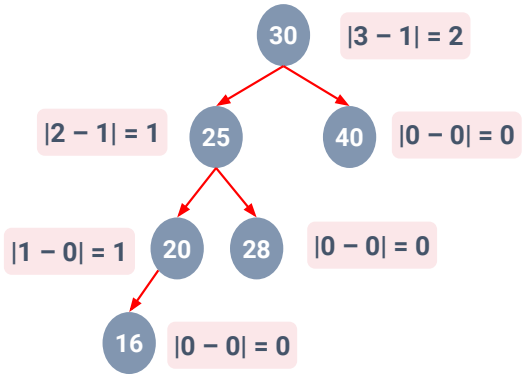


Left of Left

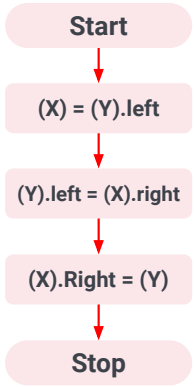
To perform a right rotation (at node 30), do **4 steps** below:

- Promote the left child (25) to be the root of the subtree.
- Move the old root (30) to be the right child of the new root.
- If the new root (25) already had a right child (28),
  - The right child (28) become the left child of the new right child (30).
- Update parents pointers of old root node (if exist).

Balancing AVL Tree (LoL)

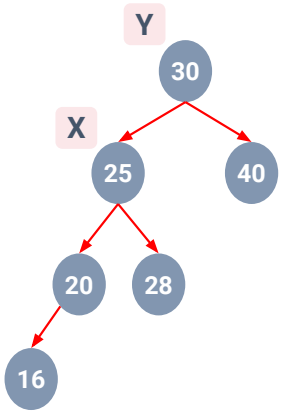


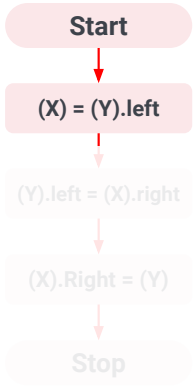
Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$



where (Y) is a node which is a rotated node

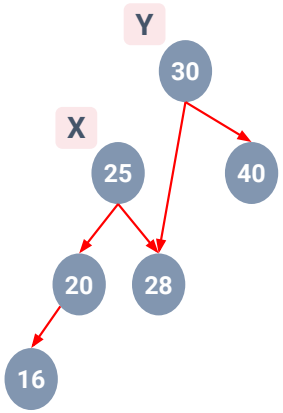
Balancing AVL Tree (LoL)

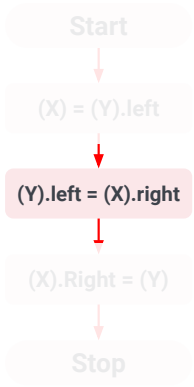




where (Y) is a node which is a rotated node

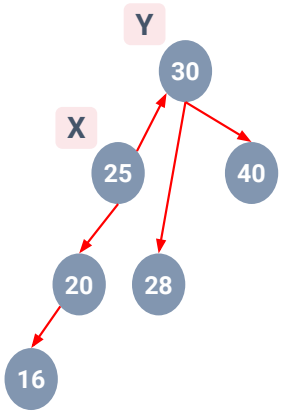
Balancing AVL Tree (LoL)

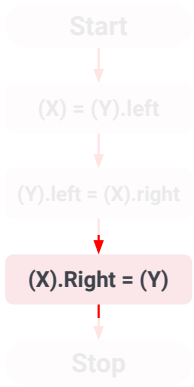




where (Y) is a node which is a rotated node

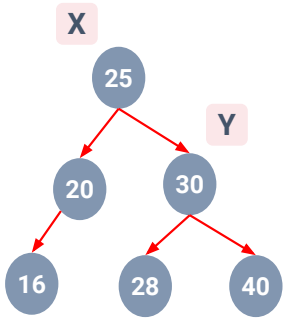
Balancing AVL Tree (LoL)

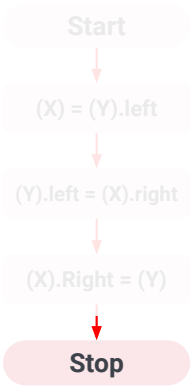




where (Y) is a node which is a rotated node

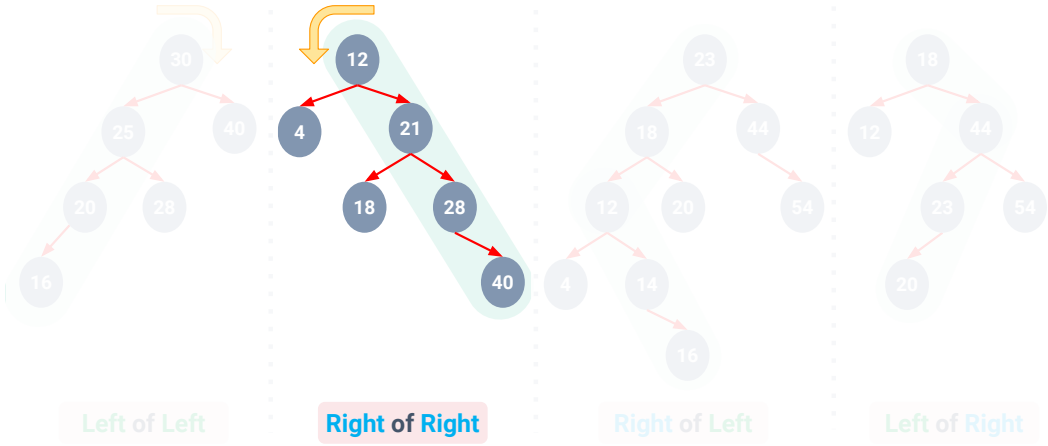
Balancing AVL Tree (LoL)



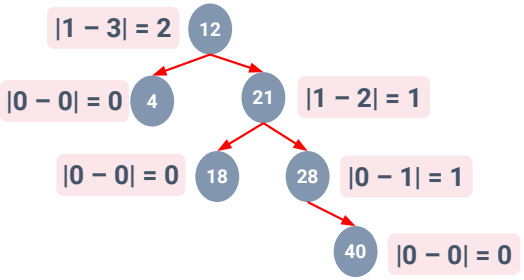


where (Y) is a node which is a rotated node

Balancing AVL Tree



Balancing AVL Tree (RoR)

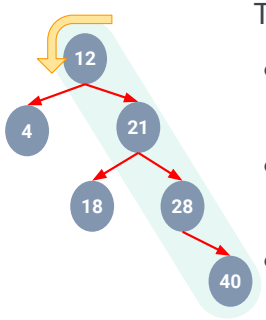


A tree is **right-heavy** with a balance factor of 2 at the root.

Require a **left rotation**.

Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

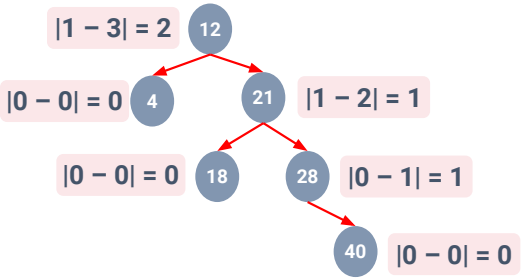
Left Rotation



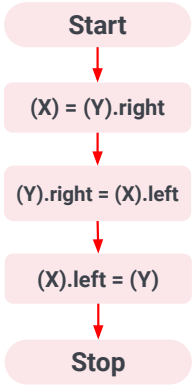
To perform a left rotation (at node 12), do **4 steps** below:

- Promote the right child (21) to be the root of the subtree.
- Move the old root (12) to be the left child of the new root.
- If the new root (21) already had a left child (18),
  - The left child (18) become the right child of the new left child (12).
- Update parents pointers of old root node (if exist).

Balancing AVL Tree (RoR)

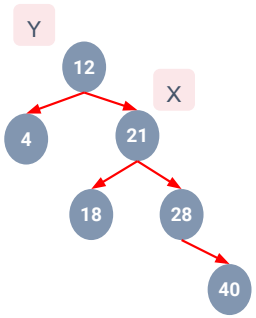


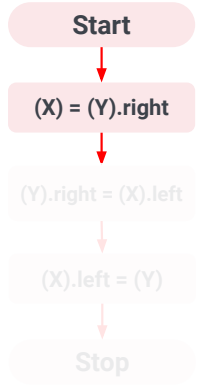
Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$



where (Y) is a node which is a rotated node

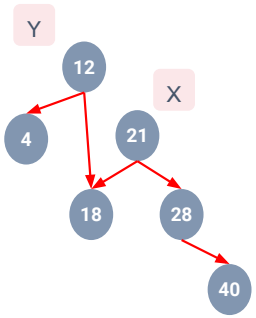
Balancing AVL Tree (RoR)

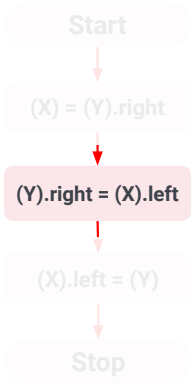




where (Y) is a node which is a rotated node

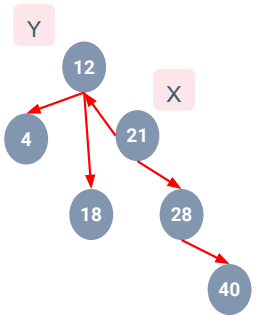
Balancing AVL Tree (RoR)

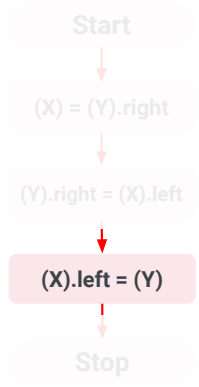




where (Y) is a node which is a rotated node

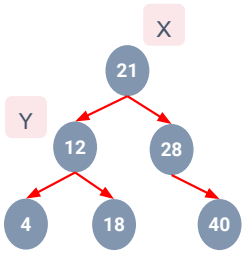
Balancing AVL Tree (RoR)

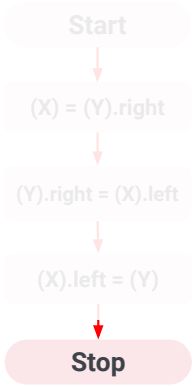




where (Y) is a node which is a rotated node

Balancing AVL Tree (RoR)



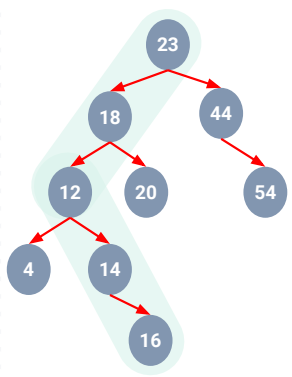


where (Y) is a node which is a rotated node

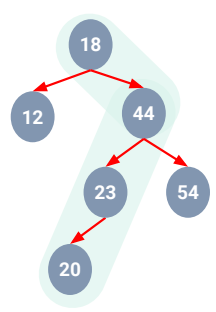
Balancing AVL Tree

Left of Left

Right of Right

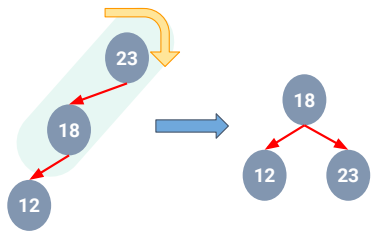


Right of Left



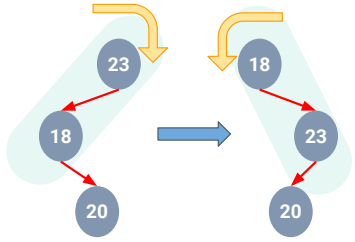
Left of Right

Right Rotation



Left Heavy Balanced

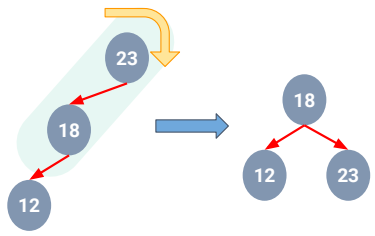
Right Rotation



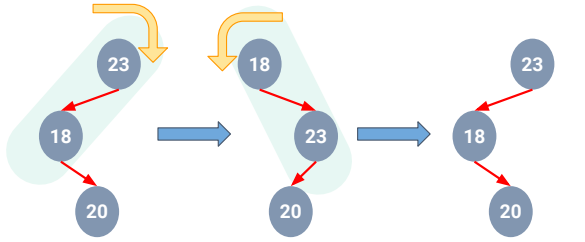
Left Heavy Right Heavy

Right Rotation

Right Rotation



Left Heavy Balanced



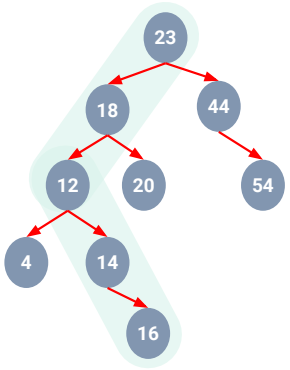
Left Heavy Right Heavy Left Heavy

Right Rotation Left Rotation

Left then Right Rotation

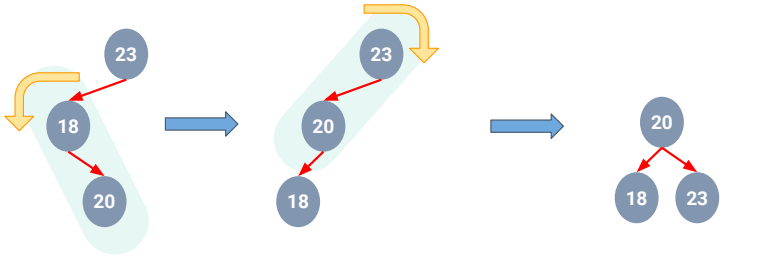
To solve this problem, there are additional Rules:

- If a subtree needs a **right rotation**,
  - Check the balance factor of the left child.
  - If the left child is right-heavy, then do **left rotation on the left child**.
  - Then do right rotation on the subtree.



Right of Left

Left then Right Rotation



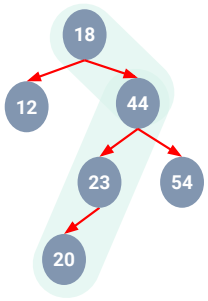
Left Rotation

Right Rotation

Right then Left Rotation

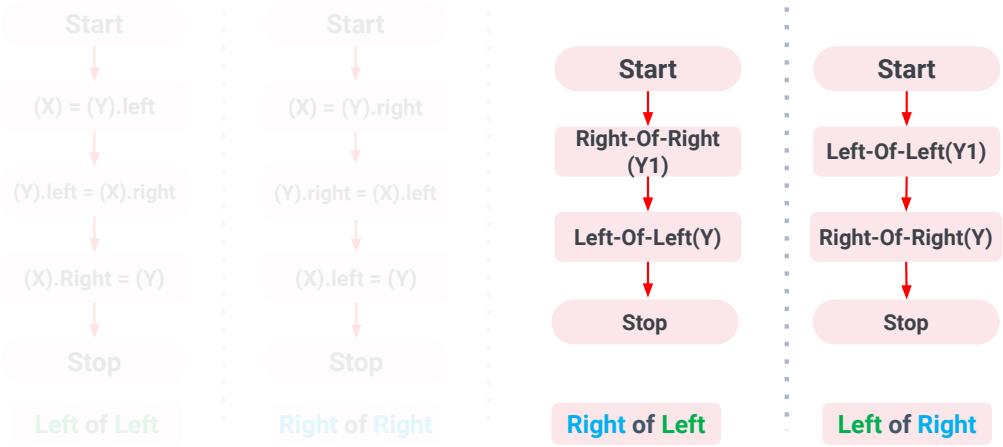
Additional Rules:

- If a subtree needs a **left rotation**,
  - Check the balance factor of the right child.
  - If the right child is left-heavy, then do **right rotation on the right child**.
  - Then do left rotation on the subtree.



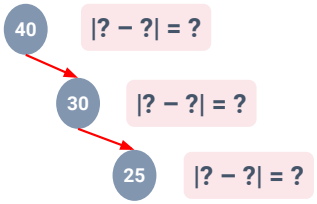
Left of Right

Balancing AVL Tree



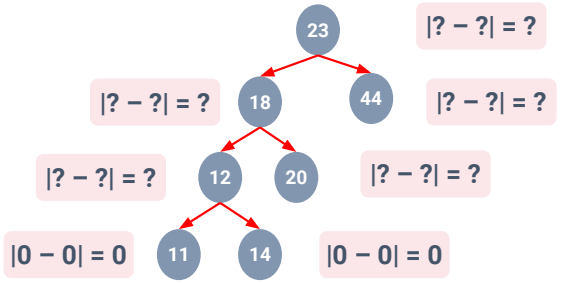
Balancing AVL Tree Exercise 1

Exercise 1: Rebalance the given AVL tree.

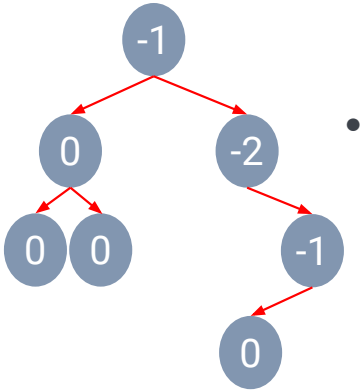


Balancing AVL Tree Exercise 2

Exercise 2: Rebalance the given AVL tree.



Balance Factor in AVL Tree



- AVL tree is considered to be balanced when the balance factor is -1, 0, or 1.
  - $|H_{left} - H_{right}| <= 1$
- AVL tree ensure that accessing the node costs only $O(\log_2 n)$ time.

Individual Assignment

- Assignment#6: BST and AVL trees
- Due 09.00 am, Tuesday 06/10/2020.
- Submission
  - Email: sirasit@it.kmitl.ac.th
  - Paper: in classroom next week
- Can be either written by hand or typing.
- Make sure to submit on time!!
  - Late submission has penalty on the score.
- If unable to submit on time for reasonable reasons, let me know asap.