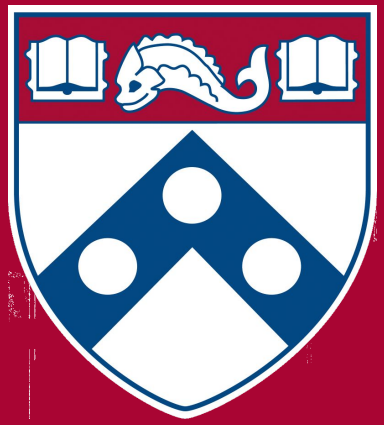
AVL Tree

CIT5940

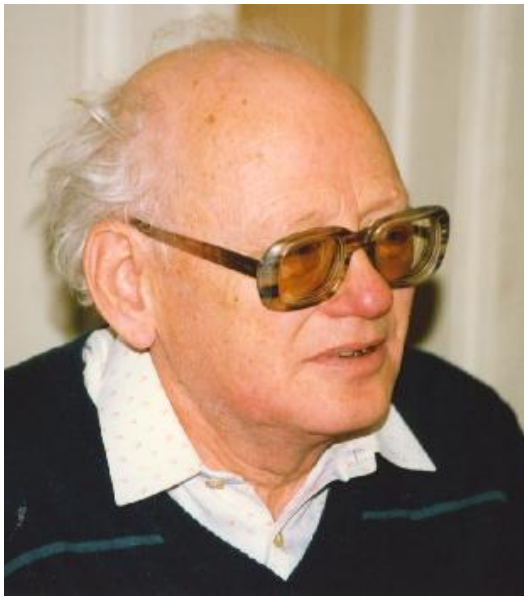


Unbalanced BST

- BST can be easily unbalanced
    - E.g. insert keys in order
- Linear runtime in the worst case
- BST must be balanced to guarantee logarithmic runtime in the worst case
- Balanced criteria:
    - The tree is *height-balanced*
    - The tree has a roughly equal number of nodes in each subtree.

AVL Tree

- Invented by Adelson-Velskii and Landis
- Self-balancing BST
- Height-balanced BST



AVL Tree Property

- *For every node, the heights of its left and right subtrees differ by at most 1*

- If the tree contains n nodes, then it has a depth of at most $O(\log n)$

Implementation

- Each node maintains its height
- Empty BST/Nodes:
  - *height = -1*
- Non-empty nodes:
  - *height = max(height of left child, height of right child) + 1*

Balance Factor

Balance factor = height of left child – height of right child

- Nodes with balance factor {-1, 0, 1} are balanced
- Unbalanced nodes must be “balanced”

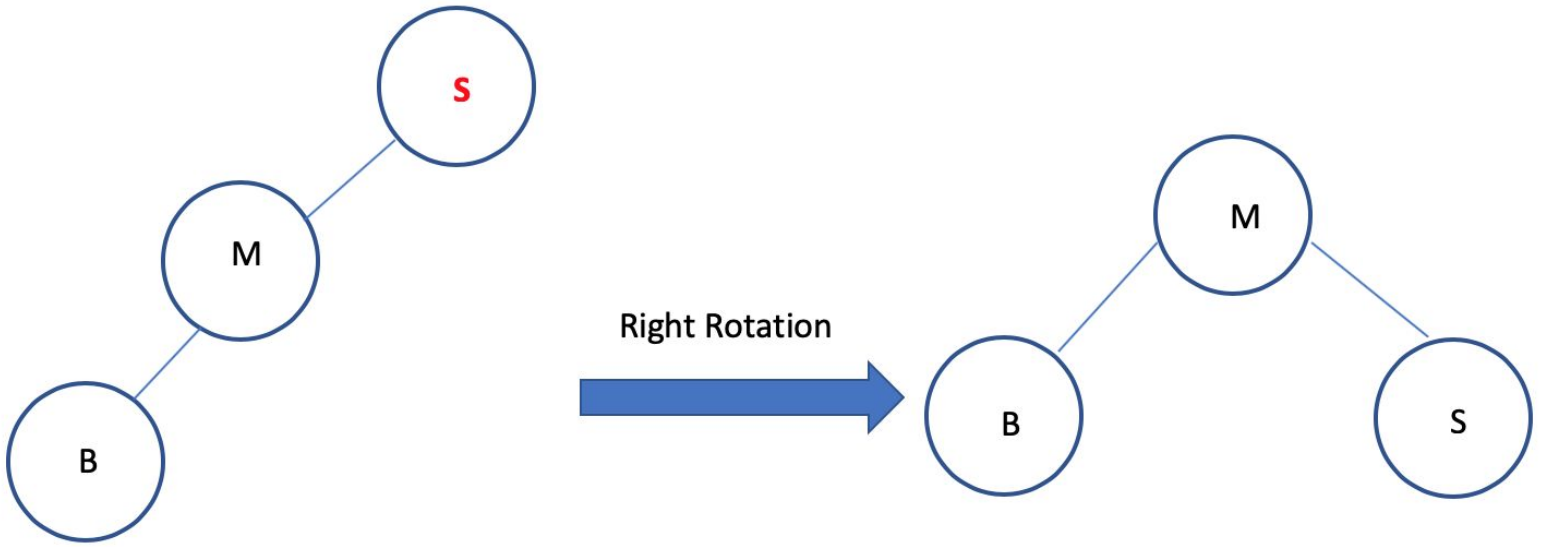
Unbalanced Nodes

- Inserting a new node can cause the BST to be unbalanced
- Given ***S*** the bottommost unbalanced node, there are four cases:

1. *The new node is in the left child of the left child of S*
2. *The new node is in the right child of the left child of S*
3. *The new node is in the left child of the right child of S*
4. *The new node is in the right child of the right child of S*

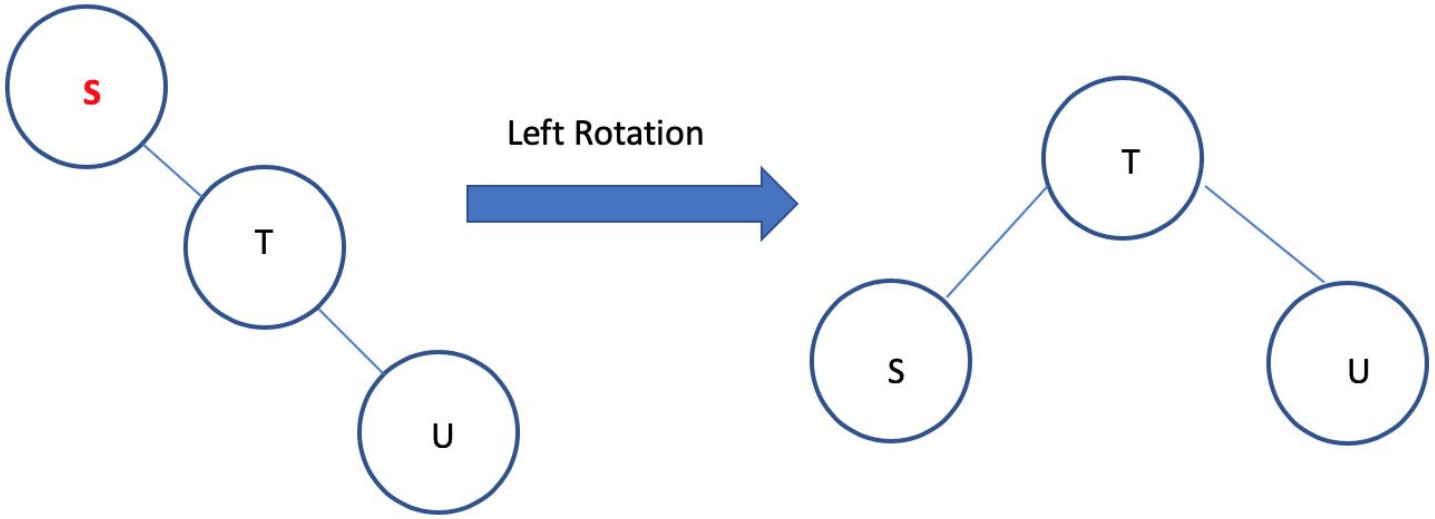
Right Rotation

- The new node is in the left child of the left child of S



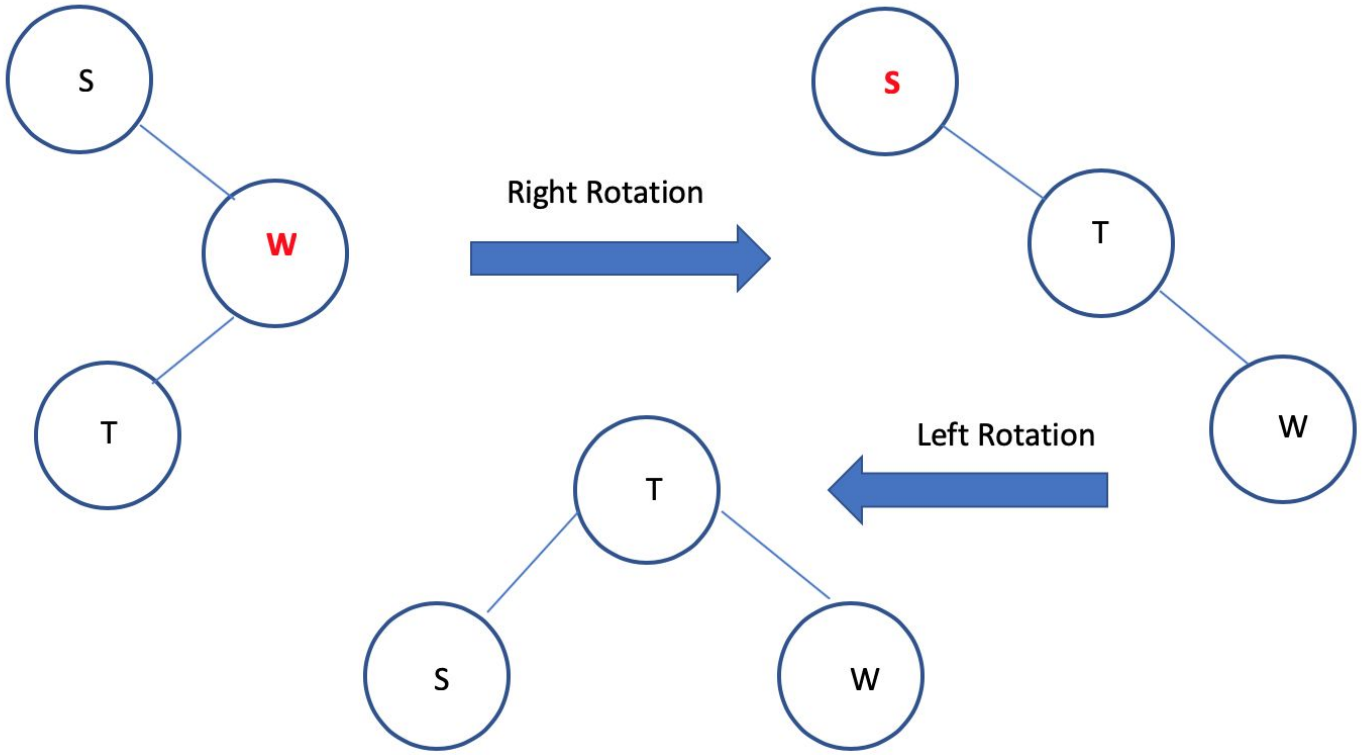
Left Rotation

- The new node is in the right child of the right child of S



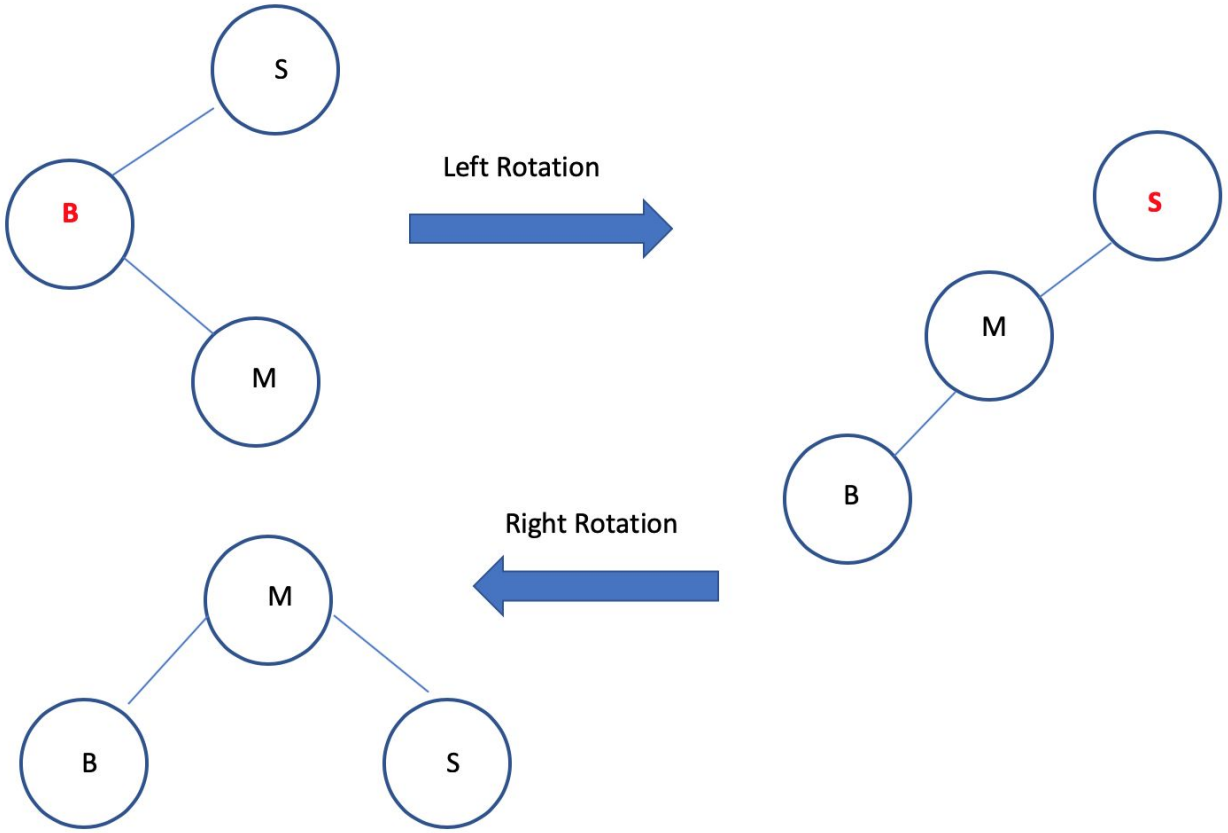
Right-Left Rotation

- The new node is in the left child of the right child of S



Left-Right Rotation

- The new node is in the right child of the left child of S



Runtime Analysis

- Rotations are local operations
- Rotations are constant time($O(1)$) operations
- AVL Tree
  - Search: $O(\log n)$
  - Insertion: $O(\log n)$
  - Deletion: $O(\log n)$