



Balanced Binary Search Trees: An Introduction

Balanced BSTs have $O(\log n)$ height, where n is the number of nodes. They guarantee efficient search, insertion, and deletion. Unbalanced trees lead to $O(n)$ worst-case performance. The goal is to maintain balance through rotations and other techniques.

Why Balance a BST?

Search time in a BST is proportional to the tree's height. An unbalanced tree's height can equal the number of nodes. A balanced tree's height is logarithmic to the number of nodes.

Skewed Tree

All nodes are on one side.

Balanced Tree

Nodes are evenly distributed.

Tree Rotations: The Core Concept

Tree rotations are local restructurings to change the tree shape. The purpose is to reduce height without violating the BST property. Types include left rotation, right rotation, and double rotations.

Left Rotation

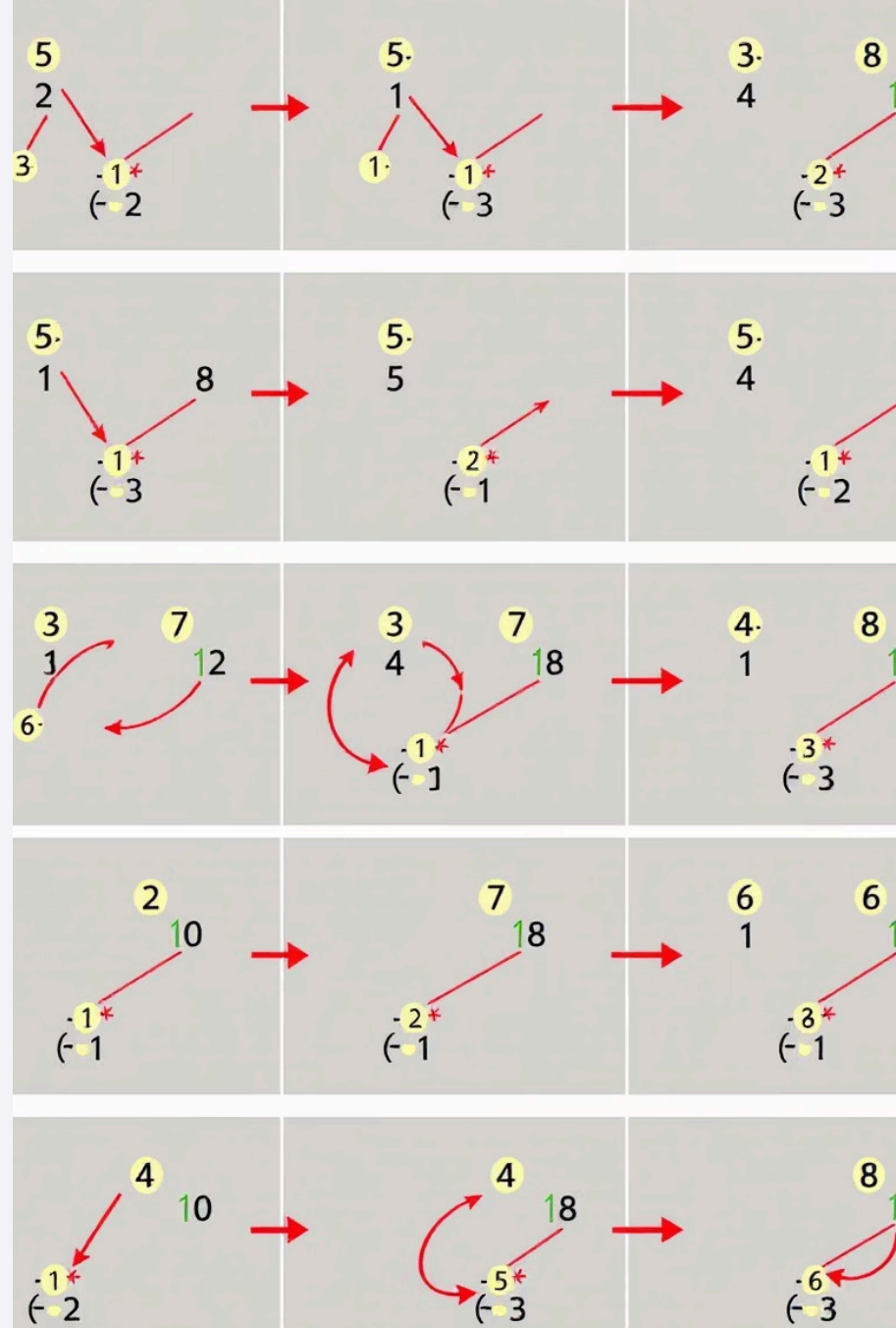
Rotates subtree to the left.

Right Rotation

Rotates subtree to the right.

Double Rotations

Combination of left and right.



Left Rotation Explained

A left rotation happens when the right subtree is too heavy. The right child becomes the new root. The left subtree of the right child moves to the right child's old parent.

1

Pivot

Right child becomes the new root.

2

Adjust

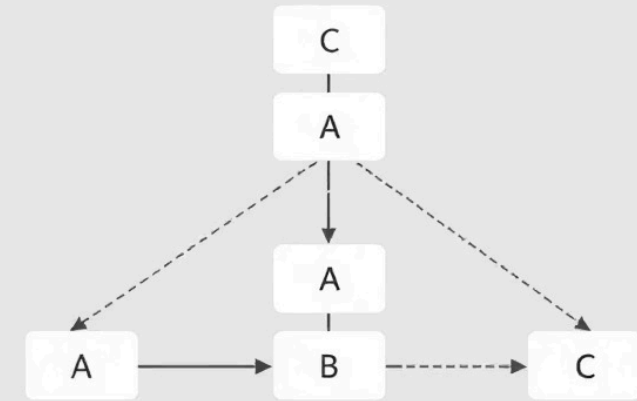
Reassign subtrees to maintain BST order.

3

Maintain

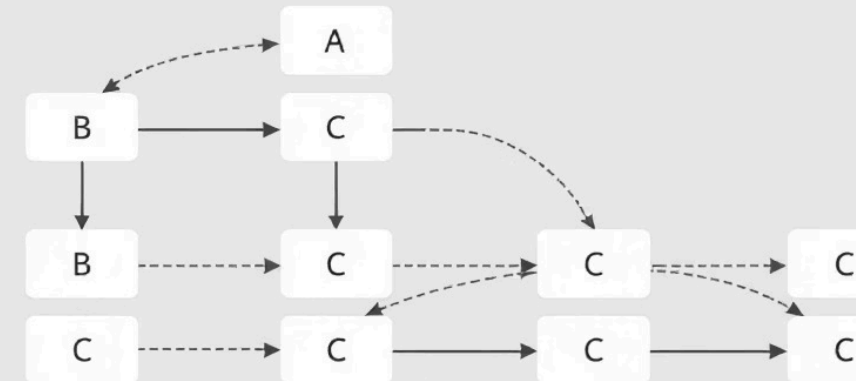
BST ordering is preserved.

Before



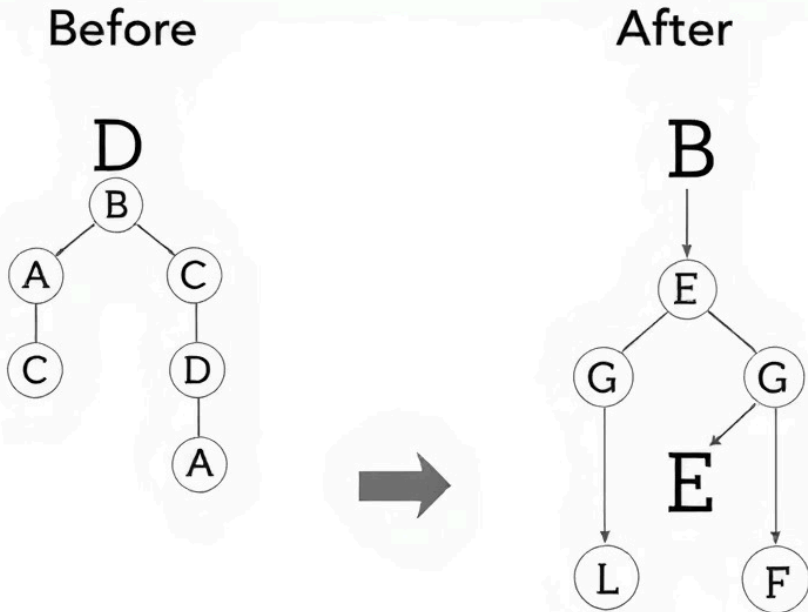
Show : Binary search tree und the nodes indesind the left rotation

After ufurg a left rotation



After:: Becary ludring the movemeot conec durg nodes left rotation

Right Search Tree on a Binary Search Tree



Right Rotation Explained

A right rotation occurs when the left subtree is too heavy. The left child becomes the new root. The right subtree of the left child moves to the left child's old parent.

1

Pivot

Left child becomes the new root.

2

Adjust

Reassign subtrees to maintain BST order.

3

Maintain

BST ordering is preserved.

Double Rotations: Left-Right and Right-Left

Single rotations are sometimes insufficient. Double rotations fix zig-zag imbalances. Left-Right rotation: left rotation on left child, then right rotation on node. Right-Left rotation: right rotation on right child, then left rotation on node.



Height Balancing Algorithms

AVL Trees maintain a balance factor using rotations. Red-Black Trees color nodes to ensure logarithmic height. B-Trees are optimized for disk-based storage. Different algorithms trade off space, complexity and speed.

AVL Trees


Balance factor of -1, 0, or 1.

Red-Black Trees

Colored nodes for logarithmic height.

B-Trees

Optimized for disk-based storage.



Conclusion: Balanced BSTs in Practice

Height balancing is crucial for BST performance. Rotations are fundamental to height balancing. Algorithms like AVL and Red-Black trees guarantee efficiency. Balanced BSTs are essential in many applications.

Performance

Guaranteed $O(\log n)$ operations.

Efficiency

Optimized search, insert, delete.

Applications

Databases, indexing, search engines.