Height-balanced trees

AVL trees

Tyler Moore

CS 2123, The University of Tulsa

The problem with binary search trees

- Search time average case: $\lg(n)$
- Search time worst case: n
  - Can you construct such a tree?
- Solution: height-balanced binary trees

Height-balanced trees

Definition

Height of a binary tree is the length of its longest path from root to leaf

Definition

Height-balanced k-tree aka HB[k] tree: binary tree where all left and right subtrees differ by at most k in height

Definition

AVL tree: HB[1] tree (named for Adelson-Vel'skii and Landis)

Note: AVL trees behave like binary trees for lookup, but vary for insertion and deletion

Performance of lookups

TABLE 3.3

Comparisons used in a search	Completely balanced tree of n nodes	AVL tree of n nodes
Worst possible number	$\lg(n+1)$	$1.44 \lg(n+2)$
Average number	$\lg(n+1)-2$	$\lg n + 0.25$†

† Based on empirical studies.

Height-balanced trees

Definition

Balance Factor aka BF(node) = Height(left subtree) - Height(right subtree)

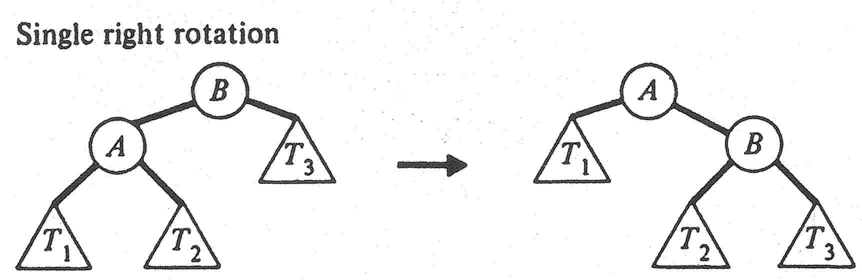
- BF(node) = 1 $\Longrightarrow$ Left-heavy tree
- BF(node) = -1 $\Longrightarrow$ Right-heavy tree
- BF(node) = 0 $\Longrightarrow$ Balanced Tree

Spot the AVL tree

AVL insert rules

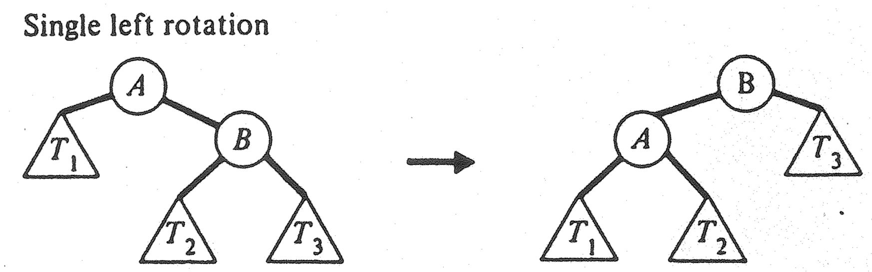
1. Find position to insert node as in a BST. Identify the deepest level node along the path that has BF 1 or -1 prior to insertion. Label this node the pivot.
2. From the pivot node down, recompute balance factors.
3. Check whether any node's balance factor switched from 1 to 2 or -1 to -2.
4. If balance factor did change to -2 or 2, then a rebalancing at the pivot is needed.

Insertion Case 1



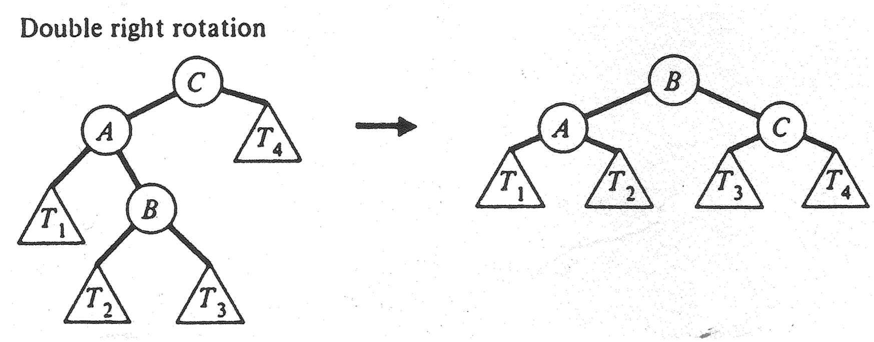
$T_1 < A < T_2 < B(\text{root}) < T_3 \quad \rightarrow \quad T_1 < A(\text{root}) < T_2 < B < T_3$

Insertion Case 2



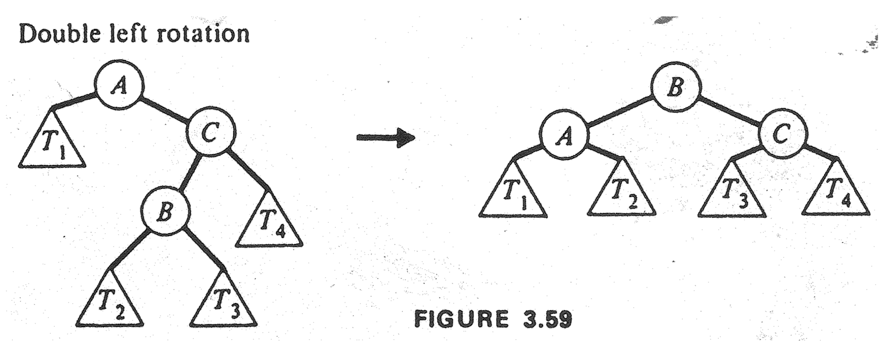
$T_1 < A(\text{root}) < T_2 < B < T_3 \quad \rightarrow \quad T_1 < A < T_2 < B(\text{root}) < T_3$

Insertion Case 3



$T_1 < A < T_2 < B < T_3 < C(\text{root}) < T_4 \rightarrow$
$T_1 < A < T_2 < B(\text{root}) < T_3 < T_4$

Insertion Case 4



$T_1 < A(\text{root}) < T_2 < B < T_3 < C < T_4 \rightarrow$
$T_1 < A < T_2 < B(\text{root}) < T_3 < T_4$

AVL example

Insert in sequence 20,10,40,50,90,30,60,70,5,4,80

In-class exercises

Analysis of AVL trees

- How often do we need to rotate?
- Cost of an insert that requires rotation
- Worst-case cost of a search

Analysis of AVL trees

- What is the additional cost of an AVL rotation
  1. Locate pivot (additional 1 unit cost per level): $O(lg(n))$
  2. Cost of rotation: $O(1)$
- Conclusion: does not affect the order of the search cost, remains $O(lg(n))$ worst case

Concluding thoughts on AVL trees

1. Insertions require at most one rotation and therefore do not affect lookup costs, but deletions require up to $\lg(n)$ rotations
2. On average, 0.465 rotations required per insertion visiting 2.78 nodes to restore balance
3. 52% of the time: no rebalancing, single rotation 23.3% , 23.2% double rotation
4. AVL preferred over other balanced binary trees if only insertion and lookup operations required; if deletion required should consider other options