CSC 212
Data Structures and Abstractions
Spring 2016

Lecture 13: Balanced Trees

Balanced Trees

Challenge:
- provide guaranteed fast tree operations

Simplest idea: require left and right subtrees to have the same height
- too rigid — hard to implement

AVL Tree

BST with a balance condition
- ensures height O(log n)

AVL Tree (Adelson-Velskii and Landis)
- maintains balance using rotations

Balance Condition

An AVL tree is a BST wherein for every node, the height of the left and right subtrees can differ by at most 1
- height information is kept for each node
- height of an empty tree is defined to be −1

Balance of a Node

balance(x) = height(x.left) - height(x.right)

An AVL tree is a BST wherein for every node x, \(|balance(x)| \leq 1\)

AVL Tree?

Minimum #nodes for an AVL with h=9
Analysis

N(h) — minimum number of nodes in an AVL tree of height h

Base cases: N(0) = 1 and N(1) = 2

For h > 1: N(h) = 1 + N(h-1) + N(h-2)

Analysis

N(h) = N(h - 1) + N(h - 2) + 1
N(h) ≥ φ^h, where φ ≈ 1.62 (cf. Fibonacci)

n ≥ N(h), (n nodes in an AVL tree of height h)

n ≥ φ^h, therefore log_φ n ≥ h

h ≤ 1.44 log n = O(log n)

Implications

Search operations same as BST's search cost O(log n)

Insert and Remove operations need to detect and fix imbalances

Remove 60? Insert 73?

Insertion

Use BST's insertion

Balance may become 2 or -2 for some node(s)

need to check nodes along the path to the root

update heights and calculate balance

How to fix imbalances?

adjust the tree by applying rotations

Four Cases

Let us call the node that must be rebalanced x

Case 1: insertion into left subtree of x's left child
Case 2: insertion into right subtree of x's left child
Case 3: insertion into left subtree of x's right child
Case 4: insertion into right subtree of x's right child

Case 1: Single Rotation (R)

Case 4: Single Rotation (L)