Balanced tree notes

Prof Bill, Mar 2020

The goal of balanced trees: **O(log n) worst-case performance**. They do this by eliminating the destructive case where a BST turns into a linked list.

Some “textbook” links:

- ➢ Morin, 9 Red-black trees, opendatastructures.org/ods-java/9_Red_Black_Trees.html
- ➢ Sedgewick algos 3.3 Balanced search trees, algs4.cs.princeton.edu/33balanced

Sections:

1. Why does balanced = log N?
2. B-trees, 2-3-4 trees
3. Red-black trees
4. AVL trees
5. Scapegoat trees

In general, we will learn the rules of each structure, and the insert and search operations. We’ll punt on delete. Rationale: Time is limited. If you understand insert and search, then you can figure out delete if you need it.

These notes are just a quick summary of each structure. I’ll have separate, detailed Prof Bill notes™ of most/all of these guys.

thanks...yow, bill

1. Why does balanced = log N?

Important to understand why this “balanced stuff” works, **O(log n) worst case**.

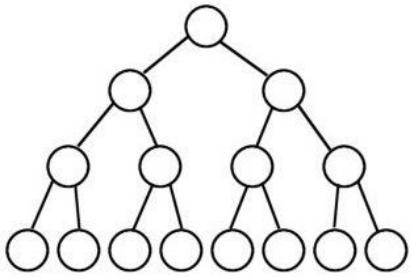
- Insert and search ops never scour the whole tree, only 1 or 2 branches.
- In a balanced tree, those branches are guaranteed to have log(n) nodes.

Counter-example: Printing the tree (balanced or not) is O(n), each node is visited.

Terms:

- ➢ **full binary tree** - every node that isn’t a leaf has two children
- ➢ **complete binary tree** - every level except the last is full and all nodes are as far left in the tree as possible

Key: Balanced tree algorithms are O(log n) because their trees are full/complete!

Example: Height = 4; num nodes = 15 In general, for a complete/full tree: Height = log(num nodes); 4 = log(15) num nodes = 2^(height); 15 = 2^4	

Btw, let’s do the maths (for Joe K):

```
# S is num nodes, n is height
# count nodes at each level
S = 2^0 + 2^1 + 2^2 + … + 2^(n-1)
2*S = 2^1 + 2^2 + 2^3 + … + 2^(n-1) + 2^(n)

subtract: 2S - S = S
S = -2^0 + 2^(n)
S = 2^n - 1
```

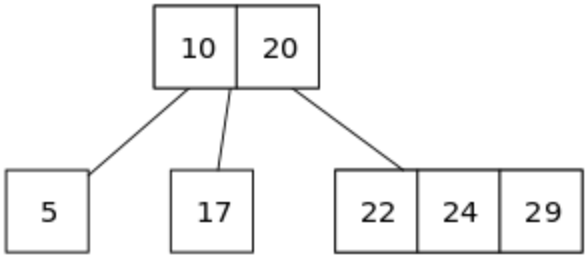
2. B-trees, 2-3-4 trees

I like Wikipedia here, actually:

- ➔ B-tree, en.wikipedia.org/wiki/B-tree
- ➔ 2-3-4 tree, en.wikipedia.org/wiki/2%E2%80%933%E2%80%934_tree

B-trees are the general case...nodes with N children.

We will study 2-3-4 trees. These have nodes with 2, 3, or 4 children.



Source: https://en.wikipedia.org/wiki/2%E2%80%933%E2%80%934_tree

Insert new nodes as a leaf, then split nodes that get too large.

Search is a little weird, but straightforward.

3. Red-black trees

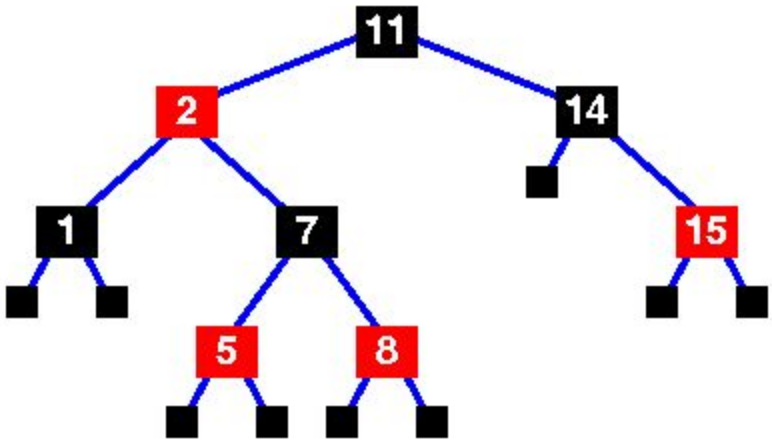
Again. Kudos to Wikipedia, en.wikipedia.org/wiki/Red%E2%80%93black_tree

The red-black rules are:

- ❏ Each node is either red or black.
- ❏ The root is black.
- ❏ All leaves (NIL) are black.
- ❏ If a node is red, then both its children are black.
- ❏ Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes.

Insert a new node as a leaf, then re-color or rotate to maintain our red-black properties.

Search is just like BST...but with the worst-case performance of O(log n).



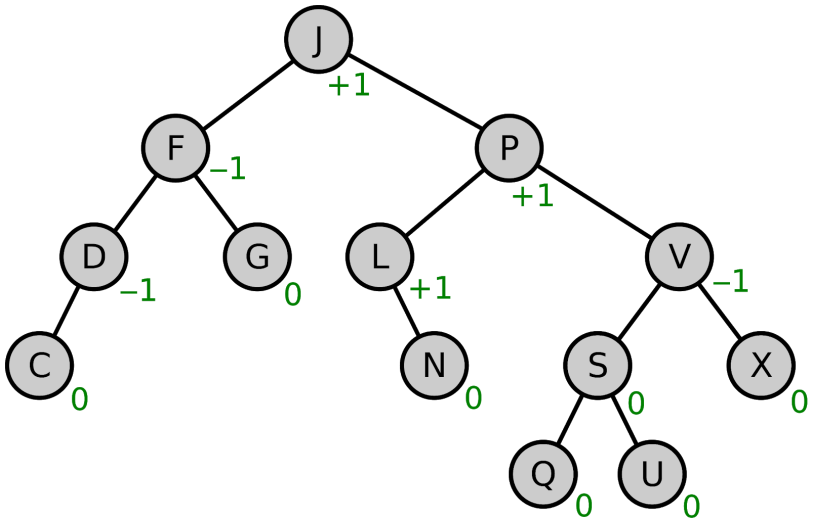
Source: www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

4. AVL trees

Wikipedia: en.wikipedia.org/wiki/AVL_tree

Quickly:

- An AVL tree is a **self-balancing** binary search tree (BST). It was the first such data structure to be invented.
- The AVL tree is named after its two Soviet inventors, Georgy Adelson-Velsky and Evgenii Landis, who published it in their **1962 paper** "An algorithm for the organization of information".
- The heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.
- Search, insertion, and deletion all take **O(log n) time** in both the average and worst cases!



/* AVL is pretty unpleasant to code */

5. Scapegoat trees

Read: Morin 8 Scapegoat trees, opendatastructures.org/ods-java/8_Scapegoat_Trees.html

Quick facts: 1) There is no animation for scapegoat trees. 2) I've never used them.

Here's a better explanation than Morin (sigh), brilliant.org/wiki/scapegoat-tree.

- Unlike the red-black tree and the AVL tree, the scapegoat tree is an unencumbered data structure. (no extra info per node, like color)