Extensions to BSTs

The binary search tree (BST) was invented in about 1960[1]; it is a binary tree that satisfies:

1. Each node contains a value.
2. For each node, all the values in its left subtree are less than its value.
3. For each node, all the values in its right subtree are greater than its value.

The big problem with the BST is that if it is not balanced, the search time can be O(`n`) for a tree of size `n`, and yet, it is difficult and time consuming to keep BSTs balanced. Therefore, computer scientists looked for extensions to the basic BST that would allow the BST to be efficiently balanced when insertions and deletions were performed. Here, we give a broad overview of some of these extensions, so you (1) have an idea about how the BST was extended and (2) get an idea about how computer scientists worked to improve data structures. No code is given.

Throughout, we consider BSTs (and their extensions) of size `n`, i.e. with `n` nodes.

AVL trees[2]

AVL trees were invented in 1962 by Russians Adelson-Velsky and Landis. The tree is named after the authors: AV for Adelson-Velsky and L for Landis. In an AVL tree, *for any node, the heights of its children differ by at most one*. If an insertion or deletion causes this property to be falsified, the tree is rebalanced to restore. How is this property maintained?

The *balance factor* for any node `m` is defined to be:

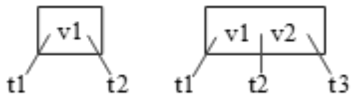
(height of `m`'s right subtree) – (height of `m`'s left subtree)

Therefore, the balance factor of each node is maintained in the range -1..1. The balance factor of node `m` is kept in node `m` itself. For this reason, the space requirement of AVL trees is O(`n`). When an insertion or deletion causes the balance factor to get out of the range -1..1, rebalancing occurs.

Using the balance factor, it can be shown that search, insertion, and deletion are all O(log `n`) operations. However, the coding of balancing and insertion or deletion is quite involved and messy, so other data structures were invented.

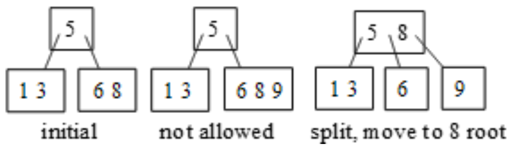
2-3 trees[3]

To help in balancing an AVL tree, each node has extra information. In 1970, John Hopcroft invented the 2-3 tree with a different idea: allow more values and subtrees in a node. Thus, it is no longer a binary tree.



As shown to the right, each node has 2 or 3 subtrees, `t1`, `t2`, and `t3`. Interspersed with 1 or 2 values, `v1` and `v2`. Here are more properties of a 2-3 tree.

- In a leaf, all subtrees are empty. In an inner node, no subtree is empty.
- In a node: (values in `t1`) < `v1` < (values in `t2`) < `v2` < (values in `t3`).
  Thus, an inorder traversal of a 2-3 tree produces the values in ascending order.
- All leaves are at the same level. The empty tree is a 2-3 tree.



Inserting a value into a 2-3 tree is fairly easy. Consider inserting 9 into the first tree to the right. It would have to be inserted as shown in the second tree, but that's not possible. Therefore, split that node as shown in the third tree, moving the middle value to the parent.

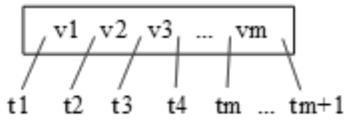
If initially the node with 5 had two values, that node would also be split and a value pushed to its parent. If there is no parent, a parent would be created.

[1] See https://en.wikipedia.org/wiki/Binary_search_tree
[2] See https://en.wikipedia.org/wiki/AVL_tree
[3] See https://en.wikipedia.org/wiki/2–3_tree

B-trees[4]

B-trees were invented by Bayer and McCreight in 1971. The basic idea is the same as the 2-3 tree but more general. We don't think that Bayer and McCreight knew about Hopcroft's 2-3 trees at the time. There are many variations of B-trees now, and we discuss one of them.

In a fixed implementation, nodes can contain from `d` to `D` subtrees, with values interspersed as shown. Thus, we speak of a `d-D` tree. With `d` = 2 and `D` = 3, we have Hopcroft's 2-3 tree. The node shown has `m` values. Here are properties of the tree:



- In a leaf, all subtrees are empty. In an internal node, no subtree is empty.
- All leaves are at the same level.
- As you might imagine, for each `k`, (values in subtree t_k) < v_k < (values in subtree t_{k+1}).

In a B-tree with 2 < `D`, creation of nodes upon insertion of a value doesn't happen as often as with a 2-3 tree, but space is typically wasted. Having `d` be `D`/2 reduces waste because then each node is at least half full.

The B-tree is used in databases and file systems, for it is well suited for systems that manipulate large blocks of data.

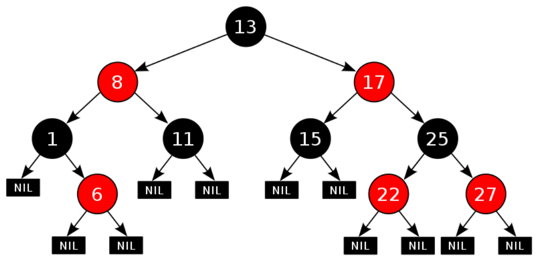
Red-black trees[5]

In 1978, Guibas and Sedgewick developed the red-black tree based on Bayer's "symmetric binary B-tree" tree (1972). They found a way to maintain a BST in close-to-balanced form with the addition of one extra bit per node, which is viewed as the color of a node —red or black. The balancing is not perfect, but it is enough to be able to prove O(log n) time for searching, insertion, and deletion. Later work in the 1990's by several people helped reduce the complexity of insertion and deletion. Even later, in 2008, Sedgwick found a way to improve it even further.[6] It took 36 years from start to finish —and perhaps you will find a way to improve it further.

Here, we just state the properties of a binary tree to be a red-black tree. You can find full explanations of insertion and deletion of values in the references in the footnotes.

A red-black tree is a BST that satisfies the following additional properties. You can verify that the tree shown, taken from footnote 5, is a red-black tree.

- Leaves do not contain data —only inner nodes.
- Each node is red or black.
- The root and all leaves are black.
- The children of a red node are black.
- All (downward) paths from a given node to any leaf contain the same number of black nodes.



Other kinds of binary search trees

Search the literature (or the web) and you will find several other variations on the binary-search tree theme, each with a different strategy or purpose. For example, look for *splay tree*, *treap* (tree heap), *tango trees*, *T-trees*, *randomized BST*, and *optimal binary search tree*.

---

[4] See https://en.wikipedia.org/wiki/B-tree.

[5] See https://en.wikipedia.org/wiki/Red–black_tree.

[6] Sedgwick: *Left-leaning red-black trees*: http://www.cs.princeton.edu/~rs/talks/LLRB/LLRB.pdf