Lecture 15: Red Black Trees 1

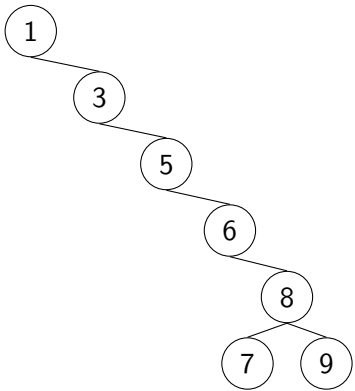
COSC242: Algorithms and Data Structures

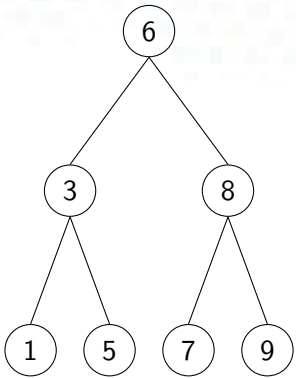
Brendan McCane

Department of Computer Science, University of Otago

The importance of balance

Search, insert, and delete are much more efficient for some BSTs than others. Contrast a BST created by inserting 1,3,5,6,8,7,9 with a second created by inserting 6,3,8,1,5,7,9:



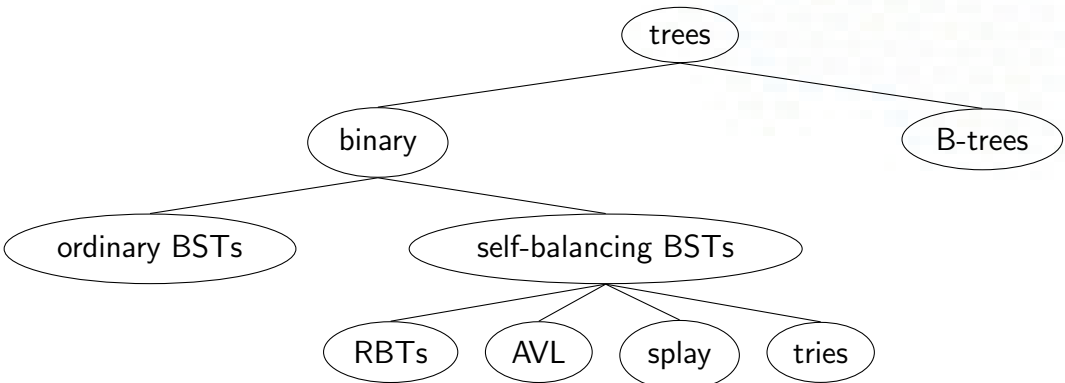


The importance of balance

Tall trees are less efficient than short trees. Note that both of these trees have the same keys, so for any given set of keys we would prefer the shortest tree containing those keys. But building optimal BSTs requires rebuilding the whole tree each time a key is inserted or deleted, and this is very expensive.

Instead, there are several good approximate algorithms that *rebalance* BSTs dynamically. We will look at one such algorithm (red-black trees).

A taxonomy of trees



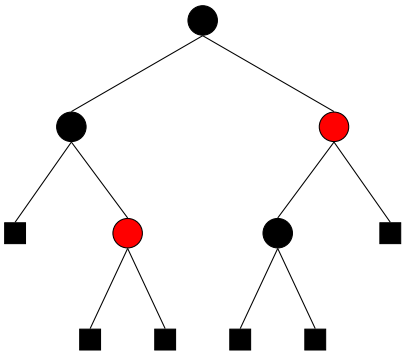
Red-black trees

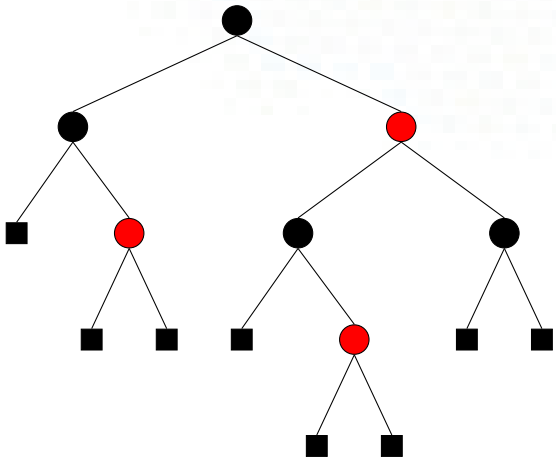
An RBT is a BST with the following properties:

1. Every node is either red or black
2. The root is black
3. All dummy leaves are black (think of dummy leaves as NULL subtrees)
4. If a node is red, then the roots of both its subtrees are black
5. For each node, all paths from the node to the leaves contain the same number of black nodes (including the dummy leaves).

RBTs

Are these RBTs? Rectangles are dummy leaves (NULL pointers).

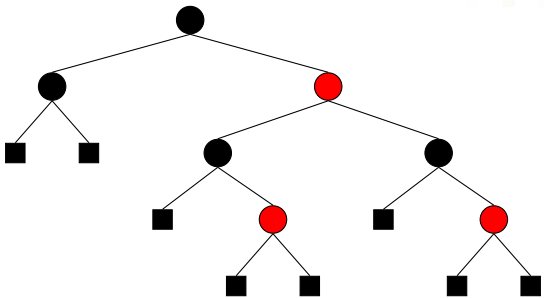




Black-height

Definition (Black-height)

The black-height of a node is the number of black nodes on any path from that node to a leaf, not counting the node itself, but counting dummy leaves.



Number of nodes of an RBT

Lemma

Let $n(x)$ be the number of real nodes of an RBT rooted at node x, then:

$$n(x) \geq 2^{b_h(x)} - 1.$$

Proof To prove this we will use induction.

Atom If $h(x) = 0$ then x is a dummy leaf so $n(x) = 0$ and:

$$2^{b_h(x)} - 1 = 2^0 - 1 = 0$$

as required.

Number of nodes continued

Induction Assume the claim holds for every node of height $h(x) = k$ (induction hypothesis). We need to show the claim is also true for nodes of height $h(x) = k + 1$.

Suppose $h(x) = k + 1$. Then $h(x) > 0$ so x is a real node with at least one child of height k. If a child y is red then $b_h(y) = b_h(x)$, otherwise $b_h(y) = b_h(x) - 1$. So $b_h(y) \geq b_h(x) - 1$.

By the induction hypothesis, each subtree of x has at least $2^{b_h(x)-1} - 1$ real nodes. So:

$$\begin{aligned} n(x) &= 1 + n(y_1) + n(y_2) \\ &\geq 1 + (2^{b_h(y_1)} - 1) + (2^{b_h(y_2)} - 1) \\ &\geq 1 + (2^{b_h(x)-1} - 1) + (2^{b_h(x)-1} - 1) \\ &\geq 2(2^{b_h(x)-1}) - 1 \\ &\geq 2^{b_h(x)} - 1. \end{aligned}$$

□

Height of RBTs

Theorem

An RBT with n 'real' nodes has height:

$$h \leq 2\log_2(n+1)$$

Height of RBTs

Proof:

Observation: A node, x, of height h has black-height $b_h(x) \geq h(x)/2$.

From the previous lemma we have:

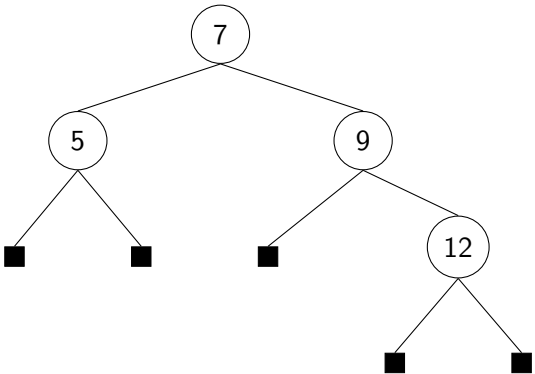
$$\begin{aligned} n(x) &\geq 2^{b_h(x)} - 1 \\ &\geq 2^{h(x)/2} - 1. \end{aligned}$$

Thus (add 1 to both sides and take logs):

$$\begin{aligned} \log\left(n(x)+1\right) &\geq \log\left(2^{h(x)/2}\right) \\ &\geq h(x)/2 \\ h(x) &\leq 2\log(n(x)+1) \end{aligned}$$

□

Inserting into an RBT

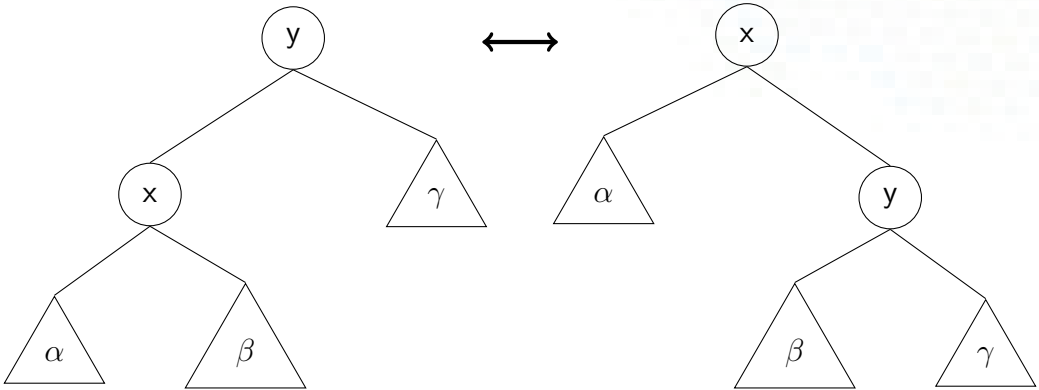


- Insert 8: can we colour the tree to make it an RBT?
- Insert 11
- Insert 10.

We need to be able to efficiently rebuild a tree to maintain its RBT qualities.

Rotations

The basis for efficient rebuilding is tree rotation (AVL trees also use rotations):



Rotation algorithm

For right rotation around y: x keeps its left subtree but its right subtree becomes the left subtree of y, y keeps its right subtree, and the parents of x and y change.

```
1:  procedure RBT_ROTATE_RIGHT(x)
2:      y = x→parent
3:      y→left = x→right
4:      x→right = y
5:      x→parent = y→parent
6:      if y→parent == Nil then
7:          root = x
8:      else
9:          y→parent→[left or right] = x
10:     end if
11:     y→parent = x
12: end procedure
```


Relevant parts of the textbook

Red-Black trees are the topic of Chapter 13 of the textbook.

Today's lecture covered sections 13.1 and 13.2.