

Red-Black Trees - the Pudding¹

Robin Dawes

February 18, 2021

¹ If you understand why this unit has this title, congratulations! You are Trivia Royalty!

FINALLY presents a convincing argument that Red-Black Trees provide $O(\log n)$ algorithms for search, insert and delete, placing these operations securely in $\Theta(\log n)$. This is a profoundly important result that gives us a basis for comparison with other data structures.

Proof that the Height of a RB Tree is in $O(\log n)$

WE HAVE SEEN that the insert operation on a RB Tree takes an amount of time proportional to the number of the levels of the tree (since the additional operations required to do any rebalancing require $O(1)$ time at each level of the tree).

We have not yet shown that the number of levels in a RB tree is in $O(\log n)$. We need to do this so that we can truthfully claim that search, insert (and by extension, delete) are $O(\log n)$ operations on RB Trees.

We will show that the height of a RB Tree containing n values is not more than $2 * \log_2(n + 1)$. This can be proved algebraically or structurally. I prefer the structural proof so we will start with that:

THEOREM: A RB Tree containing n values has height $\leq 2 * \log_2(n + 1)$

PROOF:

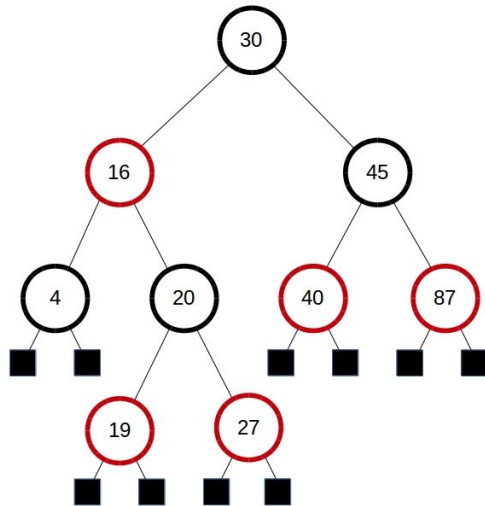
Let T be a RB tree containing n values. Now compress T by absorbing each Red vertex into its parent (discarding the values). This creates a tree (call it T') which is no longer a binary tree and which is pretty useless for practical purposes, but which is incredibly useful for proving the desired result about the height of T .

This violent act is purely a mental exercise. We do not need to design an algorithm for this compression. We need only agree that it can be done.

We know two things about T' :

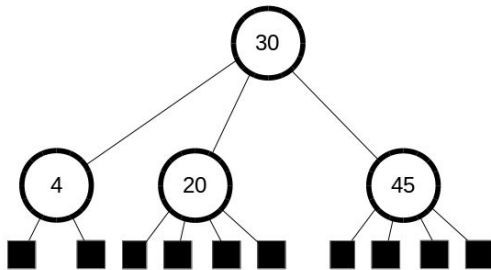
- all the leaves are at the same level (since the number of Black vertices on every path down from the root of T to the leaves of T is the same, and we have just removed all the Red vertices).
- every internal vertex of T' has 2, 3, or 4 children.

Here is a small example of T and T' :



The RB Tree T

As an exercise, see if you can come up with a sequences of insertions that would create exactly this RB tree.



The Compressed Tree T'

Each Red vertex of T has been pulled up into its parent - this has

the effect of raising some or all of the leaves to higher levels. We've lost some values but that's not important - we are not going to use this tree for anything except proving the desired property of the original RB Tree. So we don't really care about the values at this point - all we care about is the structure of this tree.

Note that - as claimed - all the leaves are now on the same level. Also note that the number of leaves has not changed (since they are all Black, none of them get absorbed into their parents).

Now, how many leaves does a RB Tree have, given that it contains n values? It may be a bit surprising to discover that a RB tree with n values always has exactly $n + 1$ leaves. We can see this is true by an inductive argument:

THEOREM: Every RB Tree that contains n values has $n + 1$ leaves,
 $\forall n \geq 1$

PROOF:

Base: Let $n = 1$. A RB Tree containing 1 value will consist of a single internal vertex with 2 leaves below it. Thus the theorem is true for $n = 1$.

Inductive Hypothesis: Suppose the statement is true for all values of $n \ni 1 \leq n \leq k$ for some $k \geq 1$.

Now let T be a RB Tree with $n = k + 1$ values. Consider what happened when we added the most recent value during the construction of T : prior to this action the RB Tree contained k values, and therefore had $k + 1$ leaves. When we add the $k + 1^{st}$ value, one of the leaves is replaced by a new internal vertex with two leaves below it. So the number of values in the tree goes up by 1, and the number of leaves also goes up by 1. Any balancing operation that we do after adding the next vertex does not affect the number of leaves. Thus the theorem is true for $n = k + 1$.


Thus the theorem is true $\forall n \geq 1$.



Now back to T and T' . Let $m = n + 1$ be the number of leaves of T' .

Since the "branch-in" factor at each internal vertex of T' is at least 2 - that is, each internal vertex has at least two children, the number of vertices one level above the leaves is $\leq \frac{m}{2}$, and the level above that

has $\leq \frac{m}{4}$ vertices, etc. The number of vertices on each level decreases by a factor of at least 2 for each level until we reach the root. Clearly the number of levels before we reach the root is no more than $\log_2 m$. Thus in the original tree T , each path from the root to the leaves has $\leq \log_2 m$ Black vertices (since the Black vertices of T are all in T' as well). If we recreate T from T' by restoring the Red vertices to their original positions, the lengths of paths from the root of T' to its leaves can at most double, due to Rule 4. Thus in T the longest path from the root to a leaf has length $\leq 2 * \log_2 m$.

Since $m = n + 1$, we finally arrive at the conclusion that the longest path from the root of a RB Tree to a leaf has length $\leq 2 * \log_2(n + 1)$, where n is the number of values in the tree. Thus the height of the tree is in $O(\log n)$. 

Thus each of our search, insert and delete operations can be performed in $O(\log n)$ time on RB Trees.

Now, **finally**, we can conclude that for search, insert and delete operations, RB trees are better than sorted arrays.

CLRS Proof

THE CLRS TEXT gives a more mathematical proof of this result that I encourage you to read. I'll summarize it here. It is an excellent example of a formal inductive argument about a data structure.

THEOREM: A RB Tree containing n values has height $\leq 2 * \log_2(n + 1)$.

PROOF: Let x be any vertex of a RB Tree T . Define $bh(x)$ to be the number of Black vertices below x on any path from x to a leaf (by virtue of the properties of RB trees, $bh(x)$ is well defined and is independent of the specific path chosen). Define T_x to be the subtree rooted at x .

Define $IV(x)$ to be the number of **internal** vertices in T_x .

CLAIM: $IV(x) \geq 2^{bh(x)} - 1$

PROOF BY INDUCTION:

Base Case: Suppose $bh(x) = 0$. This means x is a leaf.

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

T_x has no internal vertices, so $IV(x) = 0 \geq 0 = 2^{bh(x)} - 1$. Thus the base case holds.

Inductive Hypothesis: Suppose for some $k \geq 0$,

$$bh(x) \leq k \Rightarrow IV(x) \geq 2^{bh(x)} - 1$$

(ie. assume the claim is true for all $bh(x)$ values $\leq k$).

Let x be a vertex with $bh(x) = k + 1$. This means x is an internal vertex, and thus x has two children - call them y and z . There are two cases, based on the colour of x .

Case 1: x is Red. In this case, y and z must both be Black, and it is clear that $bh(y) = bh(z) = k$.

Case 2: x is Black. In this case, each of y and z could be either Red or Black. If either of them is Black, it has $bh() = k$. If either of them is red, it must have two children with $bh() = k$.

In both Case 1 and Case 2, x has (at least) two descendants with $bh() = k$. By the Inductive Hypothesis, the subtrees rooted at those descendants each have $\geq 2^k - 1$ internal vertices.

$$\text{Thus } IV(x) \geq 2 * (2^k - 1) + 1$$

$$\text{Thus } IV(x) \geq 2^{(k+1)} - 1$$

$$\text{Thus } IV(x) \geq 2^{bh(x)} - 1$$

(End of embedded proof)

The "+ 1" is there because x is an internal vertex of T_x .



Now let T be a RB Tree with n internal vertices and height h , and let r be the root of T . From the theorem we just proved, we get

$$n \geq 2^{bh(r)} - 1$$

so

$$n + 1 \geq 2^{bh(r)}$$


and

$$\log(n+1) \geq bh(r)$$

which we simply turn around to write as $bh(r) \leq \log(n+1)$

But clearly $h \leq 2 * bh(r)$, since the only difference between h and $bh(r)$ is the number of Red vertices on the paths from r to the leaves, and Red vertices can form no more than $\frac{1}{2}$ of any path.

Combining these inequalities we get $h \leq 2 * \log(n+1)$

Thus the height of a RB tree with n internal vertices (ie. a tree that stores n values) is $\leq 2 * \log(n+1)$ 

All in all, I prefer the "push the red vertices up into their parent vertices" proof - I think it is far more intuitive.