ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo
Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Notes 4.09.
Assistances and comments will be acknowledged.

4.9 Balanced Trees

Both perfect and complete trees have heights that are $\Theta(\ln(n))$ and we have seen that most operations on binary search trees are $O(h)$. Consequently, it is in our interest to build and maintain trees where the height remains bounded by $\Theta(\ln(n))$, as this will also place an upper limit on our run times.

Unfortunately, height is $O(n)$ and given a binary search tree with n nodes, even if it is initially height $\Theta(\ln(n))$, after a sequence of random insertions and erases, the average height will tend toward $O\left(\sqrt{n}\right)$. While this is still significantly better than $O(n)$, it is also much worse than the ideal.

4.9.1 The Concept of Balance

Consequently, we will try to

1. Have some means of describing a tree to say that it's shape is *acceptable* or *balanced* (that is, it must have a height $\Theta(\ln(n))$), and
2. If, following an insertion or removal, the tree is no longer balanced, we should have an algorithm that will transform the tree back into one that is balanced.

Figure 1 shows three trees, the first of which is perfect.

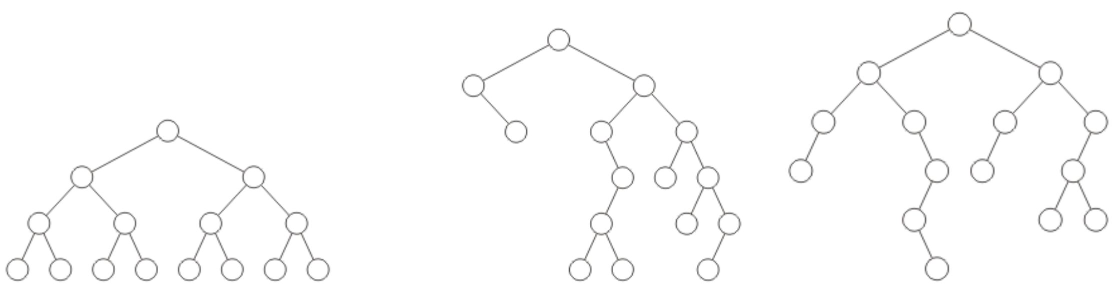

Figure 1. Three binary trees with n = 15 nodes.

Looking at the second and third trees, is unbalanced at the root, but the right sub-tree is at least numerically (the right child of the root has five strict descendants in the left sub-tree and six strict descendants in the right sub-tree). In the third tree, the root has seven strict descendants in both sub-trees; however, the sub-trees of these are essentially linked lists.

Thus, balance must be a concept that is defined throughout the tree and we will look at two possible definitions:

1. Height balancing: defining balance by comparing the heights of the two sub-trees,
2. Null-path-length balancing: defining balance by comparing the null-path lengths of each of the two sub-trees, and
3. Weight balancing: defining balance by comparing the number of empty nodes in each of the two sub-trees.

Once we create a definition of balance, we will then also to demonstrate mathematically that any tree with that property has a height $\Theta(\ln(n))$.

We will look at two definitions of height balancing and one definition of weight balancing.

4.9.2 Examples

We will look at one example of balancing that implies height balance and another that uses weight balance. The next topic, AVL trees will use height balancing.

4.9.2.1 Red-black Trees

A red-black tree is a binary tree where each node is given a colour—either red or black (0 or 1). The colours must satisfy the following properties:

1. The root node must be black,
2. All children of a red node must be black, and
3. Any path from the root node to an empty node must contain the same number of black nodes.

Figure 2 shows two examples of red-black trees.

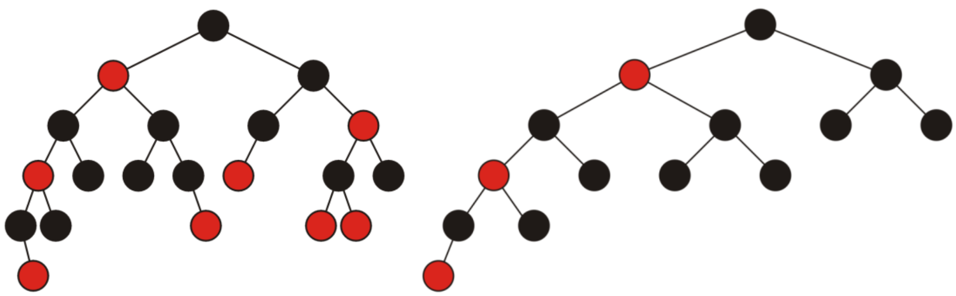

Figure 2. Two red-black trees, the second of which is at the extreme of being balanced.

If a tree satisfies these conditions, then it follows that the null-path-length passing through one child of the root cannot be more than twice the null-path length passing through the other child. It is useful to note that:

1. A perfect tree of height h has a null-path length of $h + 1$, and
2. Any other tree of height h has a null-path length less than $h + 1$.

In the optional topic on red-black trees, it is demonstrated that this implies a logarithmic height.

4.9.2.2 Weight-balance and BB(α) Trees

Consider the binary tree with n = 9 nodes in Figure 3.

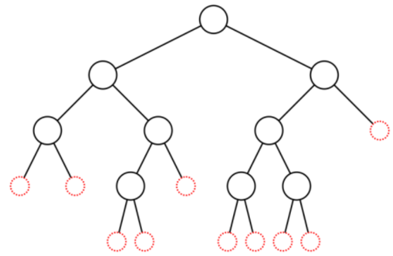

Figure 3. A binary tree with empty nodes explicitly marked.

The number of empty nodes in the left and right sub-trees is 4 and 5, respectively. Looking at the left sub-tree, the number of empty nodes between the sub-trees is 2 and 3. Looking at the right sub-tree, the ratio of the empty nodes, however, is 4 and 1. One would probably argue that the root node and its left child are weight balanced, but the right child of the root is more-than-likely not weight balanced.

A BB(α) tree maintains weight balanced if the proportion of empty nodes in the left sub-tree falls relative to the total number of empty nodes (n + 1) falls within the range [α, 1 – α]. If, in addition, we choose α so that

$$\frac{1}{4} \le \alpha \le 1 - \frac{\sqrt{2}}{2},$$

it can be shown that all operations can be performed in $\Theta(\ln(n))$ time. If $\alpha < \frac{1}{4}$, the height of the tree may grow according to $\omega(\ln(n))$; while, if $\frac{1}{4} < \alpha \le \frac{1}{3}$, the run times of the algorithms required to maintain the balance might be than $\omega(\ln(n))$.