© 2011 by Douglas Wilhelm Harder. All rights reserved.
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 4.05.
Assistances and comments will be acknowledged.

5.2 Perfect Binary Trees

Having introduced binary trees, the next two topics will cover two classes of binary trees: perfect binary trees and complete binary trees. We will see that a perfect binary tree of height h has $2^{h+1} - 1$ nodes, the height is $\Theta(\ln(n))$, and the number of leaf nodes is 2^h or $(n + 1)/2$.

5.2.1 Description

A perfect binary tree of height h is a binary tree where:

1. all leaf nodes have the same depth, h, and
2. all other nodes are full nodes.

A perfect binary tree of height 5 is shown in Figure 1.

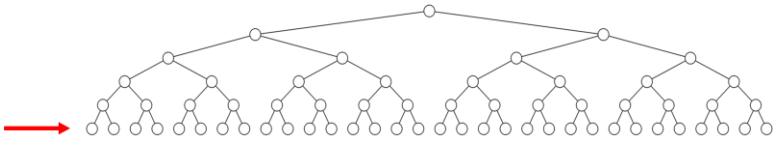


Figure 1. A perfect binary tree of height $h = 5$.

A recursive definition of a perfect binary tree is:

1. A single node with no children is a perfect binary tree of height $h = 0$,
2. A perfect binary tree with height $h > 0$ is a node where both sub-trees are non-overlapping perfect binary trees of height $h - 1$.

Perfect binary trees of heights 0, 1, 2, 3 and 4 are shown in Figure 2.

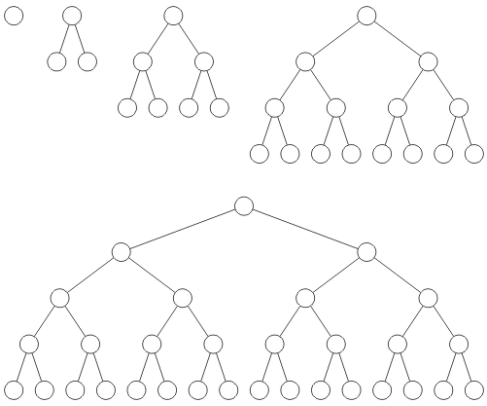


Figure 2. Perfect binary trees of height $h = 0, 1, 2, 3,$ and 4.

© 2011 by Douglas Wilhelm Harder. All rights reserved.

5.2.2 Theorems

We will now proceed to prove a number of theorems about perfect binary trees. In each case, n is the number of nodes in the tree and h is the height of the tree.

5.2.2.1 There are $2^{h+1} - 1$ Nodes

Theorem 5.2.2.1

A perfect binary tree of height h has $2^{h+1} - 1$ nodes.

Proof: We will use induction on the recursive definition of a perfect binary tree.

When $h = 0$, the perfect binary tree is a single node, $n = 1$ and $2^{0+1} - 1 = 2 - 1 = 1$.

We will assume that for an arbitrary height h that the statement is true. We must therefore show that a binary search tree of height $h + 1$ has $2^{(h+1)+1} - 1 = 2^{h+2} - 1$ nodes.

Assume we have a perfect tree of height $h + 1$ as shown in Figure 3.

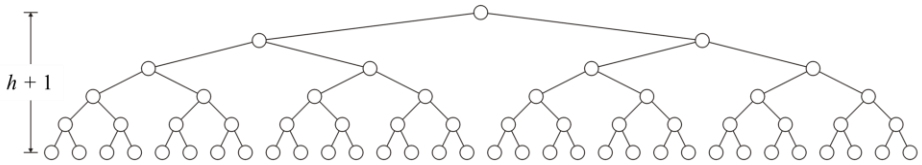


Figure 3. A perfect binary tree of height $h + 1$.

By the recursive definition of a binary tree, both sub-trees must be of height h, as is shown in Figure 4.

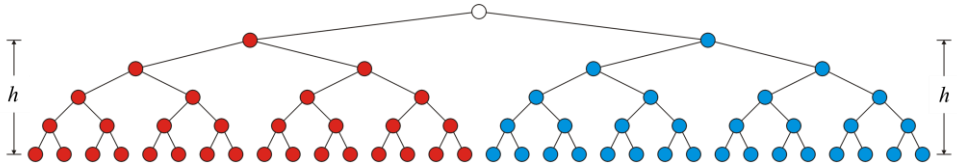


Figure 4. Both sub-trees are perfect binary trees of height h.

By assumption, the total number of nodes in each of these trees must be $2^{h+1} - 1$, and therefore, the number of nodes in the entire tree must be

$$\begin{aligned} 1 + \left(2^{h+1} - 1\right) + \left(2^{h+1} - 1\right) &= 2 \cdot 2^{h+1} + 1 - 2 \\ &= 2^{h+2} - 1, \end{aligned}$$

which is what we were required to demonstrate.

The statement is true for $h = 0$ and the truth of the statement for and arbitrary h implies the truth of the statement for $h + 1$. Therefore, by the process of mathematical induction, the statement must be true for all $h \geq 0$. ∎

© 2011 by Douglas Wilhelm Harder. All rights reserved.

5.2.2.2 Logarithmic Height

Theorem 5.2.2.2

A perfect binary tree with n nodes has height $\lg(n + 1) - 1 = \Theta(\ln(n))$.

Proof:

From the previous theorem, a perfect tree of height h has $n = 2^{h+1} - 1$ nodes. Solving this for h yields:

$$\begin{aligned} & n = 2^{h+1} - 1 \\ \Leftrightarrow \quad & n + 1 = 2^{h+1} \\ \Leftrightarrow \quad & \lg\left(n+1\right) = h + 1 \\ \Leftrightarrow \quad & h = \lg\left(n+1\right) - 1. \end{aligned}$$

It now remains to be shown that the height grows logarithmically. Using l'Hopital's rule, we also see that

$$\begin{aligned} \lim_{n\to\infty} \frac{\lg(n+1)-1}{\ln(n)} &= \lim_{n\to\infty} \frac{\dfrac{1}{\left(n+1\right)\ln(2)}}{\dfrac{1}{n}} \\ &= \lim_{n\to\infty} \frac{n}{\left(n+1\right)\ln(2)} = \lim_{n\to\infty} \frac{1}{\ln(2)} = \frac{1}{\ln(2)}, \end{aligned}$$

and because this limit is a non-zero constant, it follows $\lg(n + 1) - 1 = \Theta(\ln(n))$. ∎

5.2.2.3 There are 2^h Leaf Nodes

Theorem 5.2.2.3

A perfect binary tree of height h has 2^h leaf nodes.

Proof:

Again, we will use induction on the height. When $h = 0$, there is $2^0 = 1$ node and that node is a leaf node.

If we assume a perfect tree of height h has 2^h leaf nodes, then a perfect tree of height $h + 1$ has two sub-trees, both of which are of height h and both of which have 2^h leaf nodes. Therefore, the perfect tree of height $h + 1$ must have $2 \cdot 2^h = 2^{h+1}$ leaf nodes. ∎

© 2011 by Douglas Wilhelm Harder. All rights reserved.

Corollary 5.2.2.3

Over half of all the nodes in a perfect binary tree are leaf nodes.

Proof:

From theorems 5.2.2.3 and 5.2.2.1, we have the number of leaf nodes and the total number of nodes. Calculating the ratio, we have

$$\frac{2^h}{2^{h+1}-1} > \frac{2^h}{2^{h+1}} = \frac{1}{2}.$$

■

5.2.2.4 Logarithmic Average Depth

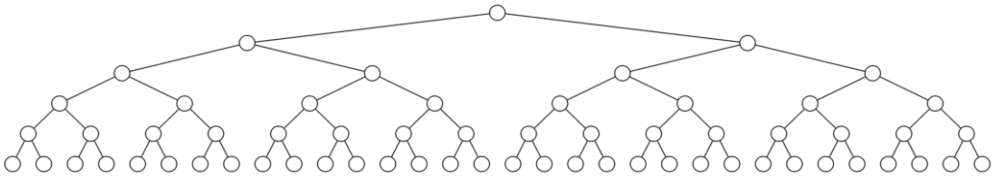
The leaf nodes in a perfect binary tree have a depth of $\Theta(\ln(n))$; however, what is the average depth of a randomly selected node? Could it be $o(\ln(n))$?

Theorem 5.2.2.4

The average depth of a node in a perfect binary tree is $\Theta(\ln(n))$.

Proof:

As is demonstrated in Figure5, there are 2^k nodes at a depth k.



Depth	Count
0	1
1	2
2	4
3	8
4	16
5	32

Figure 5. The depth and number of nodes of at each depth of a perfect binary tree of height $h = 5$.

Therefore, if there are 2^k nodes at depth k, the total sum of these depths is $k\,2^k$. We must sum this for all depths $k = 0, 1, ..., h$ (which we can find using Maple: `sum( k*2^k, k = 0..h );`) and then divide by the total number of nodes in the tree (Theorem 5.2.2.1), that is,

$$\frac{\sum_{k=0}^{h} k2^k}{2^{h+1}-1} = \frac{h2^{h+1}-2^{h+1}+2}{2^{h+1}-1} = \frac{h(2^{h+1}-1)-(2^{h+1}-1)+1+h}{2^{h+1}-1} = h-1+\frac{h+1}{2^{h+1}-1}.$$

We note that $\lim_{h\to\infty} \frac{h+1}{2^{h+1}-1} = 0$, and therefore the average depth approach $h - 1 = \Theta(\ln(n))$. ■

© 2011 by Douglas Wilhelm Harder. All rights reserved.

5.2.3 Applications

There are not many applications of perfect binary trees—they do not appear naturally. They do, however, present a good sample case. In future examples, we will determine the run time of algorithms on perfect binary trees which will often simplify the analysis after which we will generalize the results to other trees that are *close enough* to a perfect binary tree.