ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo

5.3 Complete Binary Trees

A perfect binary tree is only possible if the tree has 1, 3, 7, 15, 63, ... nodes and, thus, for most numbers of nodes, it is not possible to create a perfect binary tree. We will now look at a class of binary trees that have a shape that is reasonably close to that of perfect binary trees and we will also see a nice array representation of this tree.

5.3.1 Description

A complete binary tree is a tree where each depth is filled from left to right and we do not proceed to the next lower depth until a given depth is filled. This is essentially the depth-first traversal order shown in Figure 1.

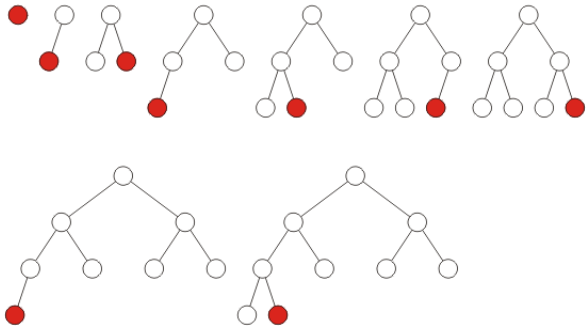

Figure 1. Filling a binary tree in breadth-first traversal order.

The recursive definition of a complete binary tree of height h is any tree where:

1. The left sub-tree is a complete tree of height $h - 1$ and the right sub-tree is a perfect tree of height $h - 2$, or
2. The left sub-tree is a perfect tree of height $h - 1$ and the right sub-tree is a complete tree with height $h - 1$.

Examples of each of these cases are shown in Figure 2.

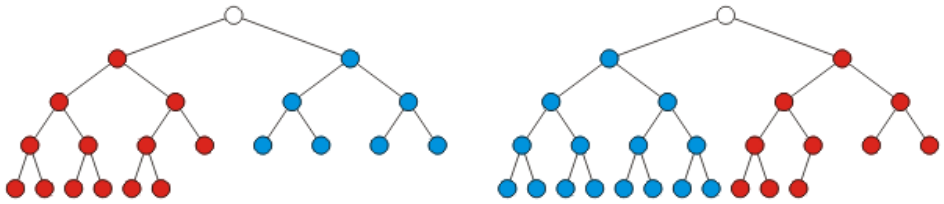

Figure 2. Examples of the two recursive definitions of a complete binary tree.

5.3.2 Logarithmic Height

Theorem

The height of a complete binary tree with n nodes is $h = \lfloor \lg(n) \rfloor$.

Proof:

When $n = 1$, $\lfloor \lg(1) \rfloor = 0$, so our formula is correct when we have a single node.

Assume the formula is true for an arbitrary n, that is, a complete binary tree with n nodes has a height equal to $\lfloor \lg(n) \rfloor$. We must show that for $n + 1$, this formula must also be correct.

We note that the value of $\lfloor \lg(n) \rfloor$ and $\lfloor \lg(n+1) \rfloor$ will only be different if $n = 2^h - 1$ and $n + 1 = 2^h$:

$$\lfloor \lg(2^h - 1) \rfloor = h - 1 \text{ and } \lfloor \lg(2^h) \rfloor = h.$$

But if $n = 2^h - 1$, it must be a perfect tree, so essentially, we have two cases:

1. If the perfect tree with n nodes, and
2. If the tree with n nodes is complete but not perfect.

Case 1: If the tree is perfect, then adding one more node will increase the height by 1. Before any insertion, the perfect tree had $n = 2^h - 1$ nodes and

$$\begin{aligned} 2^h < 2^{h+1} - 1 < 2^{h+1} \\ h = \lg(2^h) < \lg(2^{h+1} - 1) < \lg(2^{h+1}) = h + 1 \\ h \le \lfloor \lg(2^{h+1} - 1) \rfloor < h + 1 \end{aligned}$$

Therefore, $\lfloor \lg(n) \rfloor = h$, but $\lfloor \lg(n+1) \rfloor = \lfloor \lg(2^{h+1} - 1 + 1) \rfloor = \lfloor \lg(2^{h+1}) \rfloor = h + 1$. Thus, we may conclude that if we add one node to a perfect tree, the height must increase by 1.

Case 2: If the tree is not perfect, it follows that there must be some h such that $2^h \le n < 2^{h+1} - 1$. Therefore, it follows that

$$\begin{aligned} 2^h + 1 \le n + 1 < 2^{h+1} \\ h < \lg(2^h + 1) \le \lg(n+1) < \lg(2^{h+1}) = h + 1 \\ h = \lfloor \lg(2^h + 1) \rfloor \le \lfloor \lg(n+1) \rfloor < h + 1 \end{aligned}$$

Consequently, the value of $\lfloor \lg(n) \rfloor = \lfloor \lg(n+1) \rfloor$.

Therefore, by the process of mathematical induction, this must be true for all $n \ge 1$. ◊

5.3.3 Array Storage

We will now look at how we can store a complete array as an array, as opposed to using the `Binary_node` class. If we use the `Binary_node` class, each node requires two pointers in addition to the object being stored. One goal in software engineering is to always minimize the unnecessary overhead and this will be a nice example where this is possible.

Because a complete tree is filled using the breadth-first traversal order, why not store the entries in an array in the given order, as is shown in Figure 3.

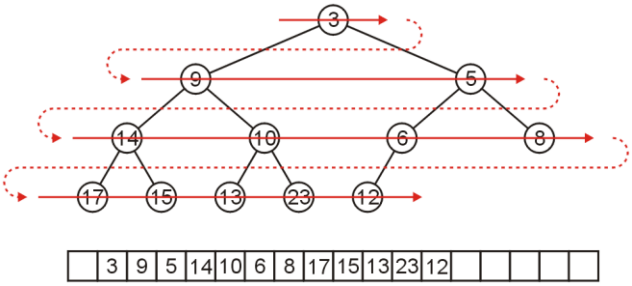

Figure 3. Filling an array with the entries of a complete tree using a bread-first traversal.

Now, if a new entry is to be placed into the complete tree, there is only one place that new node can go: to the right of node 12. It's location in the array would also be the next empty location, as is shown in Figure 4.

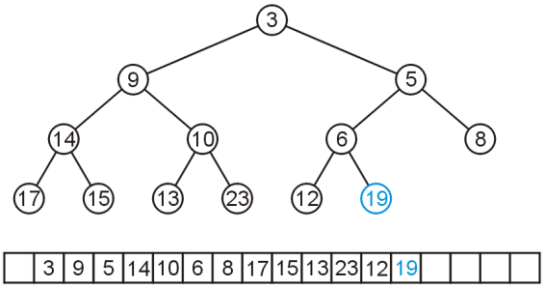

Figure 4. Inserting a new node into a complete tree.

Similarly, if we were to remove a node from Figure 3, the only node we could remove that would maintain the complete tree structure would be 12. This would also be removed from the last entry in the array, as is shown in Figure 5.

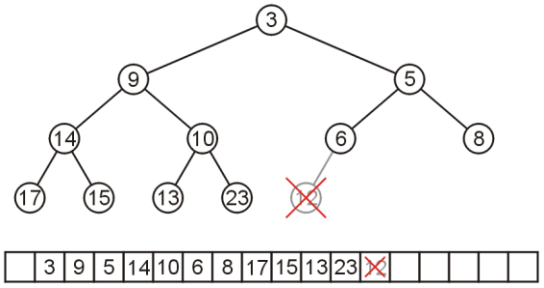

Figure 5. Removing a node from a complete tree.

Looking at 14 in the array, queries that may be asked are "What is its parent?" and "What are its children (if any)?" You probably noticed at this point that we did not fill the 0^{th} entry of the array choosing instead to store the root at the array index 1. If you look the indices in Figure 6, you will see that node 14 is stored at index 4, its parent is at index 2, and its children are at indices 8 and 9.

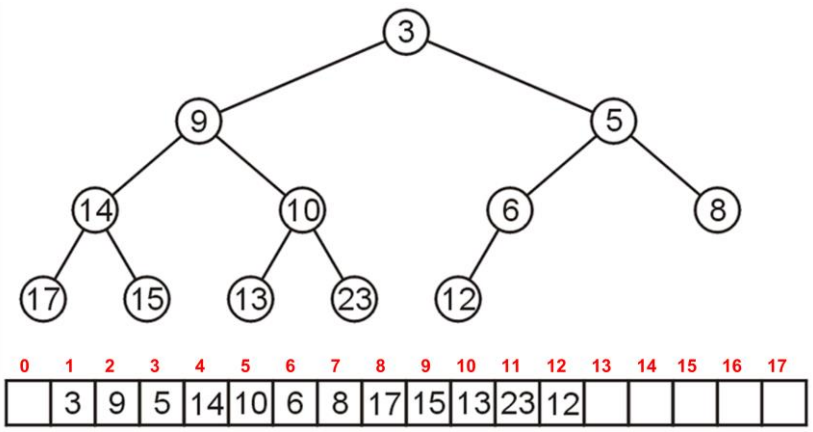

Figure 6. The array representation with indices.

It should not be difficult to see that if a node is located at index *k*:

1. Its parent is in the index k ÷ 2, and
2. Its children (if they exist) are at indices 2k and 2k + 1.

For example, as can be seen in Figure 7, the node storing 10 is at index 5, its parent (9) is in entry 5 ÷ 2 = 2, and its children are in indices 10 and 11 (13 and 23, respectively).

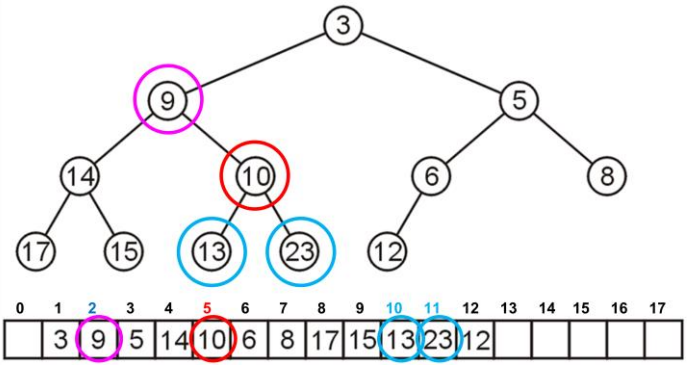

Figure 7. The parent and children of the node storing 10.

5.3.4 Array Storage for General Trees

Now that we have shown that we can store complete trees using an array, the next logical question is: can we store a general tree using an array? For example, we could just leave blanks in entries where there is an empty tree, as is shown in Figure 8.

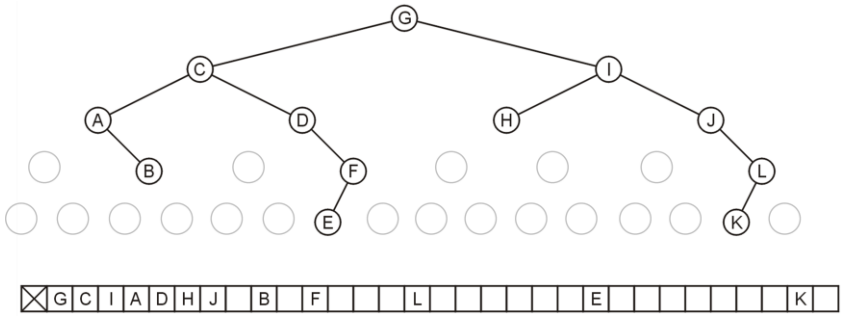

Figure 8. Storing a random binary tree as an array.

The problem is immediate: there is a significant amount of wasted space. The worst case is if the binary tree was essentially a linked list, where all but the last node has exactly one child, as is shown in Figure 9.

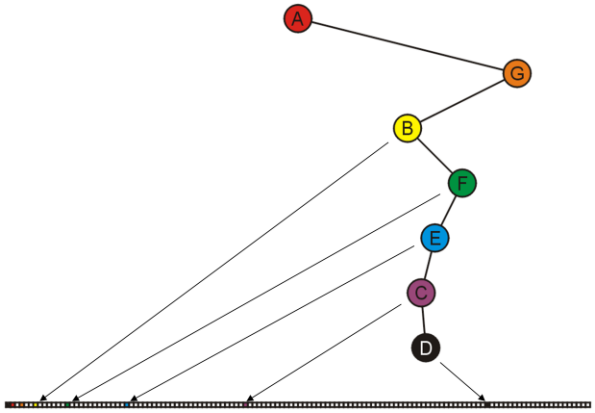

Figure 9. A degenerate binary tree with 7 nodes requiring 105 entries in the array.

It should be obvious that the worst case could require exponentially more memory: $O(2^n)$ memory to store n nodes.