# Binary Trees; Structure and Properties

## *1. Definitions*

A "binary tree" is a special case of a directed graph. While it can be defined as such, more helpful is an inductive definition given below, since then we are able to prove properties of trees by induction on their definition (or "structure"; that is, by *structural induction*).

We define trees as sets of points with "structure". The "structure" is given for any tree as an *ordered* "partition" of the set of points in the tree. First in the partition comes what we intuitively associate with the concept of "left" subtree, then comes the "root", and finally comes what we understand as the "right" subtree.

The empty set of points is conveniently thought of as a tree with the "empty structure" (i.e., since there are no points to partition, the partition itself is empty). We put these ideas together as:

**1.1 Definition.** A *binary tree*, in short *tree*, is a pair $(S, T)$ consisting of a set of points $S$ with *structure* $T$ defined as follows:

(*Basis*) $(\emptyset, \emptyset)$ is a tree; the empty tree.

If $(S_l, T_l)$ and $(S_r, T_r)$ are trees, where $S_l \cap S_r = \emptyset$ (i.e., $S_l$ and $S_r$ are *disjoint*) and if the point $r \notin S_l \cup S_r$, then $(S_l \cup \{r\} \cup S_r, \langle T_l, r, T_r \rangle)$ is a tree.

$r$ is called the *root* of the tree, while $(S_l, T_l)$ and $(S_r, T_r)$ are the left and right subtrees of $r$ respectively. The points in $S_l \cup \{r\} \cup S_r$ are called the *nodes* of the tree. □

Let us take the mystery out of definition 1.1: It really says, in its formal way, what we expect it to say: There are two ingredients that make a tree; the "data", namely, the point set $S_l \cup \{r\} \cup S_r$, and the "geometry" (or "structure", or "link"-, or "edge"-information), namely, $\langle T_l, r, T_r \rangle$.

One can "imagine" that there are *edges* introduced in trees by the inductive construction:

(1) $(\emptyset, \emptyset)$ has no edges.

(2) Assuming we have introduced edges to each of $(S_l, T_l)$ and $(S_r, T_r)$, we introduce *at most two* edges to $(S_l \cup \{r\} \cup S_r, \langle T_l, r, T_r \rangle)$ as follows: One from $r$ to the root of $(S_l, T_l)$ if $S_l \neq \emptyset$ and one from $r$ to the root of $(S_r, T_r)$ if $S_r \neq \emptyset$.

Why bother with $(S,T)$ and not just say the tree *is* $T$ (i.e., the part with the structure subsumes the data, doesn't it?). Well, it's a subtle technical point. Suppose we follow this idea and try to re-do definition 1.1 to say things like "$\ldots r$ is not in either of $T_l \ldots$". *Loosely* interpreted (i.e., "by abuse of language") probably we understand this as the same as "$\ldots r$ is not in either $S_l \ldots$". But that is *not* the same thing.

Here's an example to convince you, taken from a simpler but similar context, that of set theory: First of all, the "ordered pair" $\langle a,b\rangle$ is imposing a structure on the set $\{a,b\}$, right?

Also, we know that $\langle a,b\rangle$ is *really* the set $\{\{a\},\{a,b\}\}$. So while it is the case that $a \in \{a,b\}$ ($a$ is in the "data set") it is *not* correct to say that $a \in \langle a,b\rangle$! Correspondingly, the negative statements "$a$ not in $\{a,b\}$" and "$a$ not in $\langle a,b\rangle$" do not say the same thing, *technically*. The second is not a substitute of the first.

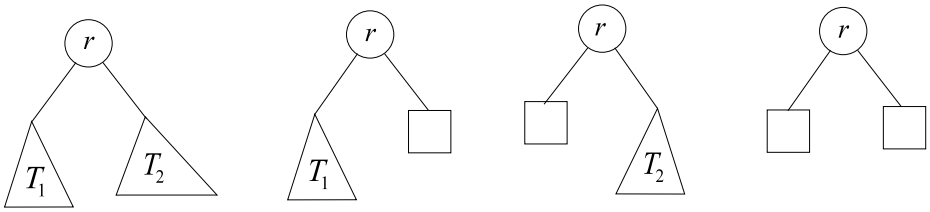
The above nit picking will not affect us, for as usual, mathematicians (and theoretical computer scientists) make a big deal of being 100% "correct" in a definition—on paper—but then they turn around, and in the interest of clarity, they hand-wave and bend the rules a bit ...

Here we will, from now on, take the "data" part $S$ (in $\langle S,T\rangle$) for granted, and say *by abuse of language* that $T$ (of $\langle S,T\rangle$) is a tree (instead of the correct $\langle S,T\rangle$ is a tree).

**1.2 Example.** $\emptyset$, $\langle\emptyset,1,\emptyset\rangle$, and $\langle\langle\emptyset,1,\emptyset\rangle,2,\emptyset\rangle$ are trees (obviously on the data sets $\emptyset$, $\{1\}$ and $\{1,2\}$ respectively).

We draw trees as follows: A "square" denotes the empty tree while a small circle denotes any tree node (see below).

Thus, the "general tree" is drawn as one of the following, where $r$ is the root.




The leftmost drawing uses the notation of a "triangle" to denote a tree. The other three cases are used when we want to draw attention to the fact that the right (respectively, left, both) subtree(s) is (are) empty. □

**1.3 Definition.** We agree that we have two types of tree-*notations* (Again, abusing language we say that we have two types of *trees*).

*Simple Trees* are those drawn *only* with "round" nodes (i.e., we do not draw the empty subtrees).

*Extended Trees* are those that all empty subtrees are drawn as "square nodes". We call, in this case, the round nodes *internal* and the square nodes *external*. □

Clearly, the "external nodes" of an extended tree cannot hold any information since they are (notations for) empty subtrees.

Alternatively, we may think of them (in implementation terms) as notations for *null links*. That is, in the case of *simple trees* we do *not* draw any null links, while in the case of *extended trees* we draw *all* null links as square nodes.

**1.4 Definition.** We recall standard graph theory terminology:

If node $a$ points to node $b$ by an edge, then $b$ is $a$ child of $a$ and $a$ is *the* parent of $b$. If two nodes are the children of the same node, then they are *siblings*.

A sequence of nodes $a_1, a_2, \ldots, a_n$ in a tree is a *path* or *chain* iff for *all* $i = 1, 2, \ldots, n-1$, $a_i$ is the parent of $a_{i+1}$. We say that this is a chain *from* $a_1$ *to* $a_n$. We say that $a_n$ is a *descendant* of $a_1$ and that $a_1$ is an *ancestor* of $a_n$.

A node is a *leaf* iff it has no children. □

In an extended tree the only leaves are the external (square) nodes.

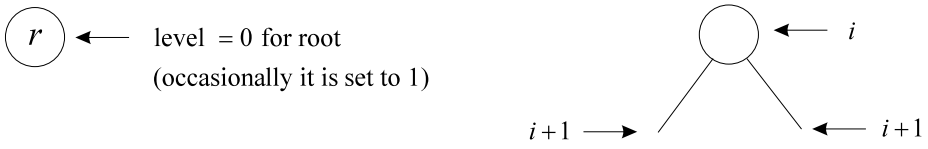
**1.5 Definition.** We define *levels* of nodes in a tree recursively (inductively):

The root has level 0 (sometimes we assign level 1 to the root, as it may prove convenient).

If $b$ is any child of $a$ and $a$ has level $i$, then $b$ has level $i+1$.

The *highest* level in a tree is called the *height* of the tree. □

**1.6 Example.** (Assignment of levels).



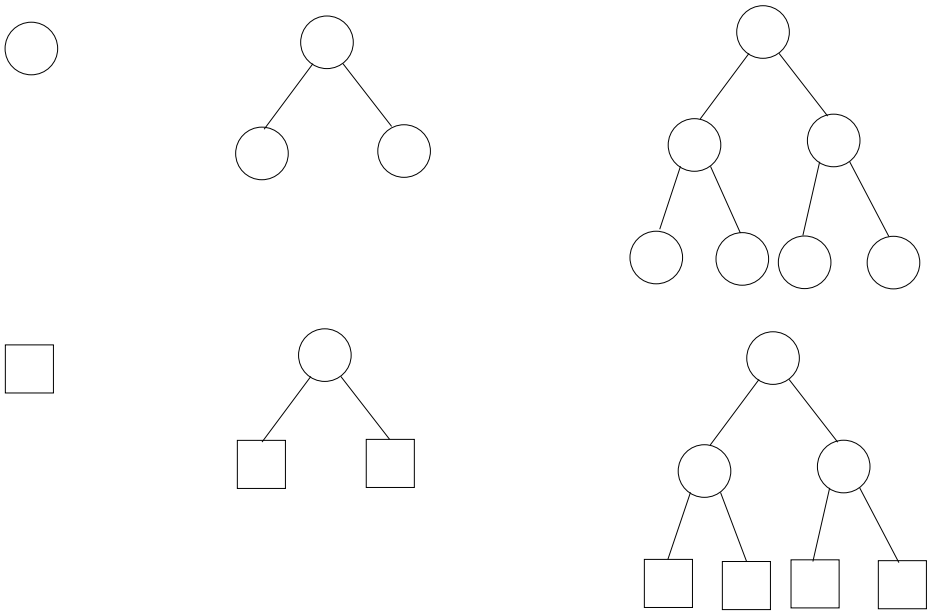

□

**1.7 Definition.** A *non leaf* node is *fertile* iff it has two children. A tree is fertile iff *all* its non leaves are fertile.

A tree is *full* iff it is fertile *and all* the leaves are at the same level. □

An extended tree is always fertile. The last sentence above then simplifies, if restricted to such trees, to "All square nodes are at the same level".

**1.8 Example.** (Full Trees). A "full tree" has all the possible nodes it deserves.

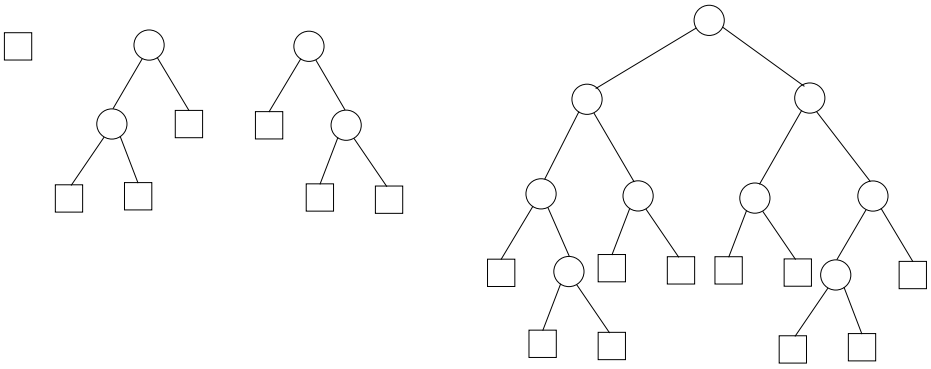


□

**1.9 Definition.** A tree is *complete* iff it is fertile *and* all the leaves occupy *at most* two consecutive levels (obviously, one is going to be the last (highest) level). □

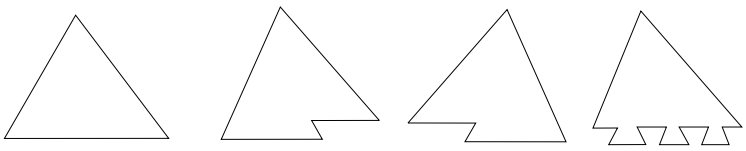
Again, for extended trees we need only ask that all square nodes occupy "at most two consecutive levels".

**1.10 Example.** (Complete Trees).



Redraw the above so that all the square nodes are "rounded" and you get examples of complete *Simple Trees*. □

**1.11 Example.** There is a variety of complete trees, the general case having the nodes in the highest level scattered about in any manner. In practice we like to deal mostly with complete trees whose highest level nodes are left justified (left-complete) or right-justified (right-complete). See the following, where we drew (abstractly) a full tree (special case of complete!), a left complete, a right complete, and a "general" complete tree.



Example 1.10 provides a number of more concrete examples □

# 2. Some Theorems

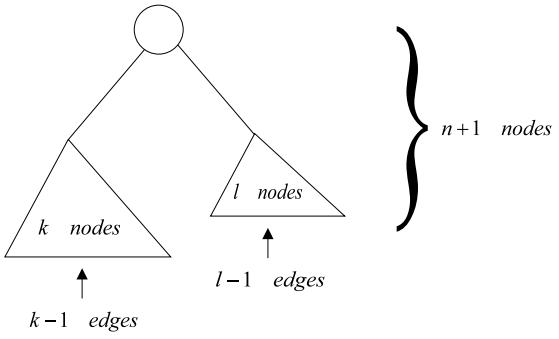
**2.1 Theorem.** *An* extended *tree with a total number of nodes n (this accounts for internal* and *external nodes) has* $n-1$ *edges.*

*Proof.* We do induction with respect to the definition of trees, or as we say in short, *induction on trees.*

*Basis.* The smallest tree is $\emptyset$, i.e., exactly one "square" node. It has no edges, so the theorem verifies in this case.

*I.H.* Assume the claim for "small" trees.

*I.S.* Consider the "big" tree below composed of two "small" trees of $k$ and $l$ nodes and a root. Say the total number of nodes is $n+1$.†




† No magic with $n+1$. We could have called the total $n$, but then we would have to add "where $n \geq 1$" to account for the presence of the root. The "$\geq 1$" part is built-in if you use $n+1$ instead.

By I.H., the left and right (small) subtrees have $k-1$ and $l-1$ edges respectively. Thus the total number of edges is $k-1+l-1+2=k+l$ (Note that in an extended tree all round nodes are fertile, so both edges emanating from the root *are* indeed present).

On the other hand, the total number of nodes $n+1$ is $k+l+1$. We rest our case. □

**2.2 Corollary.** *An extended tree of* $n$ internal *nodes has* $n+1$ external *nodes.*

*Proof.* Let us have $\varepsilon$ internal and $\phi$ external nodes. Given that $\varepsilon = n$.

By the above, the tree has $\varepsilon+\phi-1$ edges. That is, accounting differently, $2\varepsilon$ edges since all round nodes are fertile, and the square nodes are all leaves. Thus,

$$\varepsilon+\phi-1=2\varepsilon$$

from which, $\phi=\varepsilon+1$. Thus there are $n+1$ square nodes as claimed. □

**2.3 Corollary.** *A* simple *tree of* $n\geq 1$ *nodes has* $n-1$ *edges.*

*Proof.* Let $E$ be the original number of edges, still to be computed in terms of $n$. Add external nodes (two for each "original" leaf). What this does is:

It adds $n+1$ square nodes, by the previous corollary.

It adds $n+1$ new edges (one per square node). Thus,

$$\text{Total Nodes} = 2n+1$$
$$\text{Total Edges} = E+n+1$$

By theorem 2.1, $E+n+1=2n$, hence $E=n-1$ as claimed. □

**2.4 Theorem.** *In any nonempty* fertile simple *tree we have*

$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level}}} 2^{-l} = 1$$

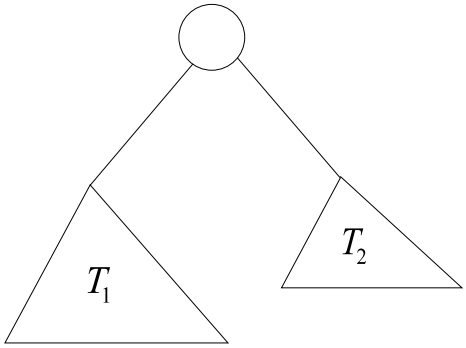
*where we assigned level* 0 *to the root.*

*Proof.* Induction on trees.

*Basis* The smallest tree is one round node. Its level is 0 and $2^{-0}=1$, so we are OK.

*I.H.* Assume for small trees, and go to the "big" case.

*I.S.* The big case (recall that the tree is fertile, so even though simple, the root has two children).




Since each of $T_1$ and $T_2$ are "small", I.H. applies to give

$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level in } T_1}} 2^{-l} = 1 \tag{1}$$

and

$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level in } T_2}} 2^{-l} = 1 \tag{2}$$

It is understood that (1) and (2) are valid for $T_1$ and $T_2$ "free-standing" (i.e., root level is 0 in each). When they are incorporated in the overall tree, call it $T$, then their roots obtain a level value of 1, so that formulas (1) and (2) need adjustment: All levels now in $T_1, T_2$ are by one larger than the previous values. Thus,

$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level in } T}} 2^{-l} = \sum_{\substack{l \text{ is a leaf's} \\ \text{level in } T_1 \\ \text{free standing}}} 2^{-(l+1)} + \sum_{\substack{l \text{ is a leaf's} \\ \text{level in } T_2 \\ \text{free standing}}} 2^{-(l+1)} \overset{\text{I.H.}}{=} 1/2 + 1/2 = 1$$

□

**2.5 Corollary.** *In an* extended *tree*

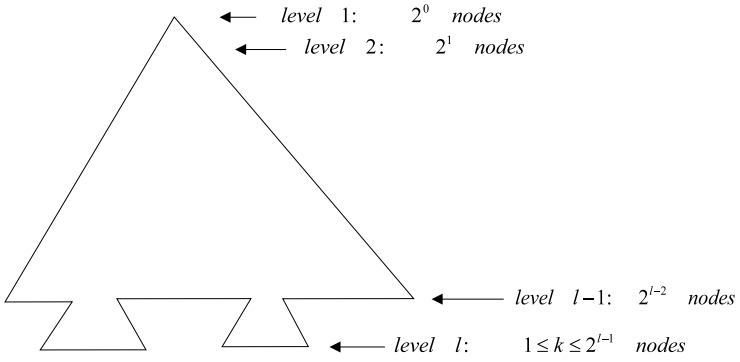
$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level}}} 2^{-l} = 1$$

*where we assigned level* 0 *to the root.*

**2.6 Corollary.** *In both 2.4 and 2.5, if the root is assigned level 1, then*

$$\sum_{\substack{l \text{ is a leaf's} \\ \text{level}}} 2^{-l} = 1/2$$

Next we address the relation between $n$, the number of nodes in a complete tree, with its height $l$.




Clearly,

$$n = 2^0 + 2^1 + \cdots + 2^{l-2} + k \le 2^l - 1 \qquad (A)$$

thus

$$2^{l-1} - 1 < n \le 2^l - 1$$

From this follows

$$2^{l-1} < n + 1 \le 2^l$$

or

$$2^{l-1} \le n < 2^l$$

leading to

$$\boxed{l = \lceil \log_2(n+1) \rceil = \lfloor \log_2 n \rfloor + 1} \qquad (*)$$

a good formula to remember.

Of course, all this holds when counting levels from 1 up. Check to see what happens if the root level is 0.

How does $k$, the number of nodes at level $l$, relate to $n$, the number of nodes in the tree?

From $(A)$,

$$\boxed{k = n + 1 - 2^{l-1}} \qquad (**)$$

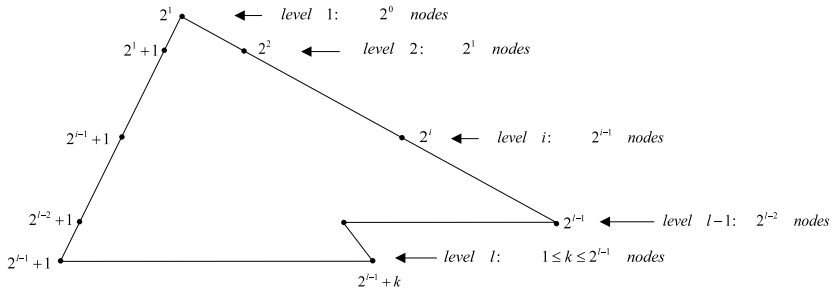
another very important formula to remember, which can be also written (because of $(*)$) as

$$\begin{aligned} k &= n + 1 - 2^{\lceil \log_2(n+1) \rceil - 1} \\ &= n + 1 - 2^{\lfloor \log_2 n \rfloor} \end{aligned} \qquad (B)$$

Note that $(**)$ or $(B)$ hold even if some or all nodes at level $l-1$ have no more than one child (in which case the tree fails to be complete, or fertile for that matter).

# 3. An application to summations

Let us see next what happens if we label the nodes of a left-complete tree by numbers successively, starting with label 2 for the root.




An easy induction shows that at level $i$ we have the labels

$$2^{i-1}+1, 2^{i-1}+2, \ldots, 2^i \tag{1}$$

Note that if $t$ is *any* of the numbers in (1), then $2^{i-1} < t \le 2^i$, hence

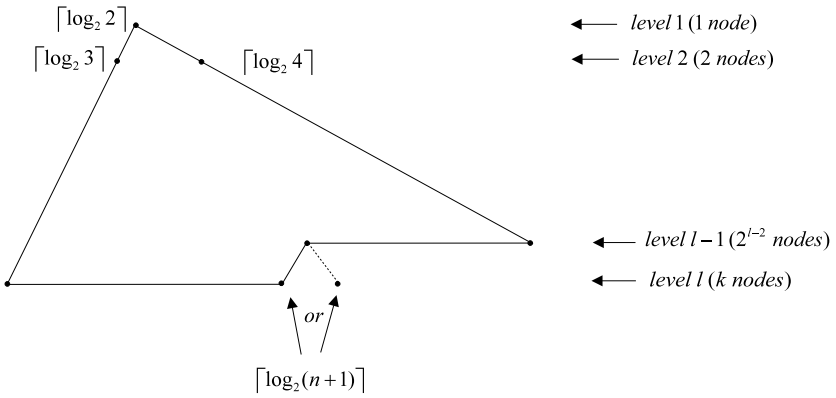
$$\lceil \log_2 t \rceil = i$$

In words, the ceiling of $\log_2$ of *any* node-label at level $i$ equals $i$.

We are in a position now to evaluate

$$A = \sum_{i=2}^{n+1} \lceil \log_2 i \rceil \tag{2}$$

which arises in the analysis of certain algorithms that we will encounter soon. The figure below helps to group terms appropriately:




Clearly,

$$A = \sum_{i=1}^{l-1} i2^{i-1} + k\lceil \log_2(n+1) \rceil \tag{3}$$

To compute (3) we need to find $k$ as a function of $n$, and to evaluate

$$B = \sum_{i=1}^{l-1} i2^{i-1} \tag{4}$$

There are two cases at level $l$ as in the previous figure. Regardless, $k$ is given in $(**)$ of the previous section as

$$k = n + 1 - 2^{l-1}$$

Thus, we only have to compute (4). Now,

$$\begin{aligned} B &= \sum_{i=1}^{l-1} i2^{i-1} \\ &= \sum_{i=0}^{l-2} (i+1)2^i \\ &= \sum_{i=0}^{l-2} i2^i + 2^{l-1} - 1 \\ &= 2\sum_{i=1}^{l-2} i2^{i-1} + 2^{l-1} - 1 \\ &= 2\sum_{i=1}^{l-1} i2^{i-1} - (l-2)2^{l-1} - 1 \\ &= 2B - (l-2)2^{l-1} - 1 \end{aligned}$$

Thus, $B = (l-2)2^{l-1}+1$, and the original becomes (recall $(*)$, $(**)$!)

$$\begin{aligned} A &= (l-2)2^{l-1}+1+kl \\ &= l2^{l-1}-2^l+1+(n+1-2^{l-1})l \\ &= (n+1)l-2^l+1 \\ &= (n+1)\lceil\log_2(n+1)\rceil-2^{\lceil\log_2(n+1)\rceil}+1 \end{aligned}$$

**Note.** A rough analysis of $A$ would go like this: Each term of the sum is $O\big(\log(n+1)\big)$ and we have $n$ terms. Therefore, $A = O\big(n\log(n+1)\big)$. However we often need the exact answer...