Chapter 13-Trees

Thursday, January 5, 2023
2:15 PM

Trees:

Trees are the central structure for storing and organizing data in computer science. (They're also good for the environment)

A ***Tree*** is a undirected graph with a special node called the *root*, in which every node is connected to the root by **exactly one** path. When a pair of nodes are neighbors in the graph, the node nearest the root is called the *parent* and the other node is its *child.*

A ***Leaf Node*** is a node that has no children. A node that does have children is known as an ***Internal Node***.

Levels are based on how many edges away from the root nodes are.
(The root is defined to be level 0)

The ***Height*** of a tree is the maximum level of any of its nodes or, the maximum level of any of its leaves or, the maximum length of a path from the root to a leaf.

If you can get from x to y by following zero or more parent links, then y is an ancestor of x and x is a ***descendent*** of y. So x is an ancestor/descendent of itself. The ancestors/descendents of x other than itself are its ***proper ancestors/descendents***. If you pick some random node z in a tree T, the ***subtree rooted at*** z consists of z (its root), all of z's descendents, and all the edges linking these nodes.

A binary tree allows each node to have at most two children. An m-ary tree allows each node to have up to m children.

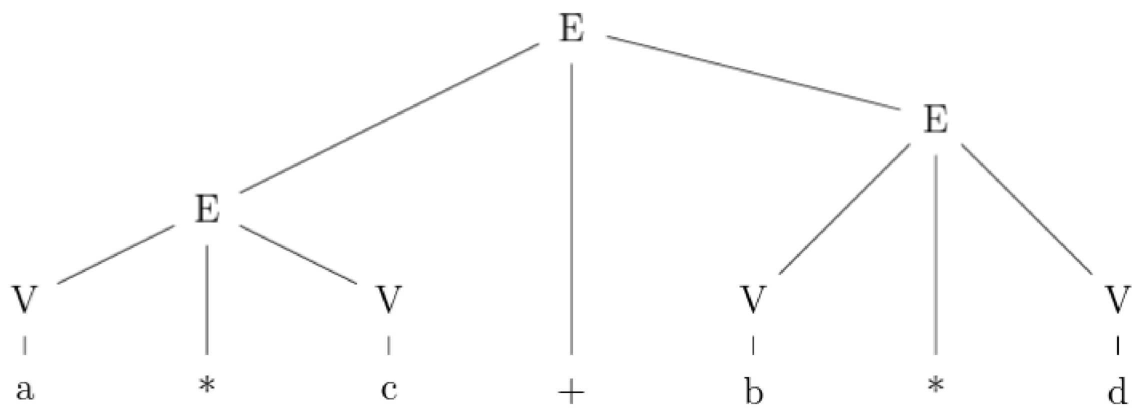
In a **full** m-ary tree, each node has either zero or m children. Never an intermediate number.

In a ***complete m-ary tree***, all leaves are at the same height.

Balanced binary trees are binary trees in which all leaves are at approximately the same height.

(Height is proportional to
$\log_2 n$
)

Parse Trees are trees that show the structure of a sequence of word/characters/symbols(***Terminal Symbols***).
Parse Tree example: (a*c) + (b*d)



*(a, *, C, +, b, d are all terminal symbols)*

This kind of labeling is called ***Context-free Grammar.***

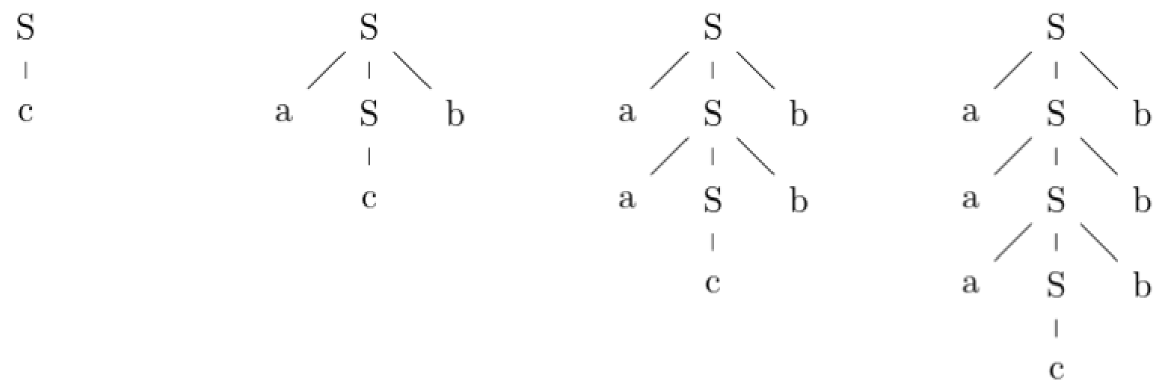
The ***Terminal Sequence*** is ***Generated*** by a specific grammar/rule.
*(with the above example, a*c + b*d is the Terminal Sequence)*

$$
\begin{aligned}
E &\rightarrow E + V \mid E * V \mid V + V \mid V * V \\
V &\rightarrow a \mid b \mid c \mid d
\end{aligned}
$$

This is how you define a grammar, E and V are the ***Start Symbols***, which are allowed to appear on the root node. The nodes that are at the lower end of a path (the lowest row) are called the ***Terminals***.

Another Example, S:

$$
\begin{aligned}
S &\rightarrow aSb \\
S &\rightarrow c
\end{aligned}
$$

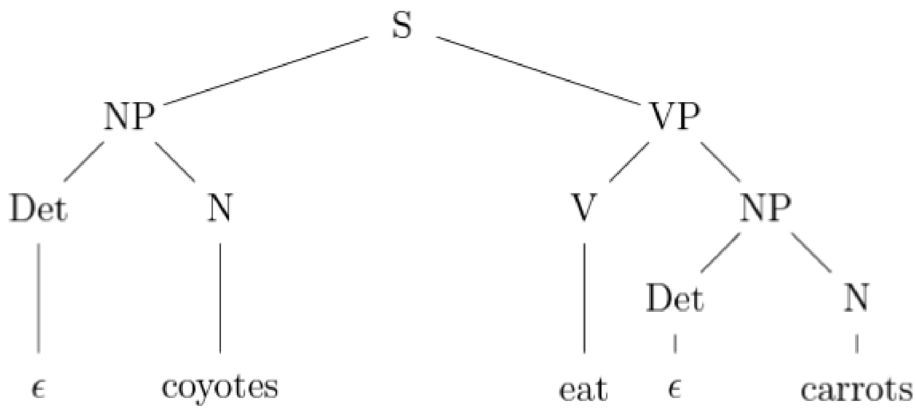


The sequences of terminals for the above trees are (left to right): c, acb, aacbb, aaacbbb. The sequences from this grammar always have a c in the middle, with some number of a's before it and the same number of b's after it.

Engrish grammar example:

$$\begin{aligned} S &\rightarrow NP\ VP \\ VP &\rightarrow V\ \ NP \mid V\ \ NP\ \ NP \\ NP &\rightarrow Det\ \ N \\ N &\rightarrow coyotes \mid rabbits \mid carrots \\ V &\rightarrow eat \mid kill \mid wash \\ Det &\rightarrow \epsilon \mid the \mid all \mid some \end{aligned}$$

This will generate terminal sequences such as "All coyotes eat some rabbits." Every noun phrase (NP) is required to have a determiner (Det), but this determiner can expand into a word that's invisible in the terminal sequence. So we can also generate sequences like "Coyotes eat carrots."

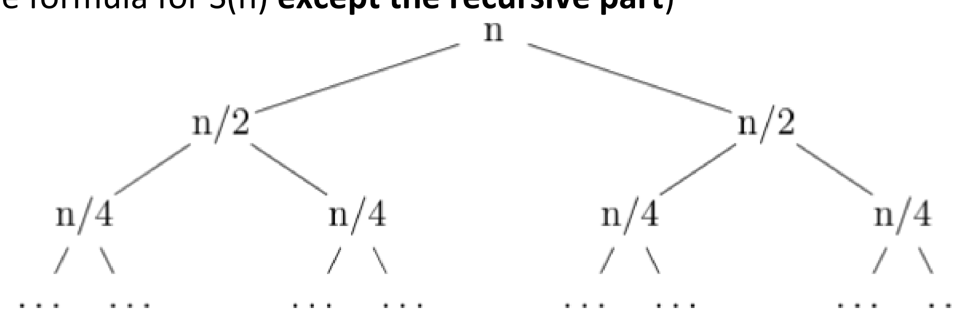


Recursion Trees:
The behavior of recursive functions can be written out as trees.

Example:

$$S(1) = c$$
$$S(n) = 2S(n/2) + n, \quad \forall n \geq 2 \ \ (n \text{ a power of } 2)$$

(The top node relative to each 2 rows represents S(n) and contains everything in the formula for S(n) **except the recursive part**)

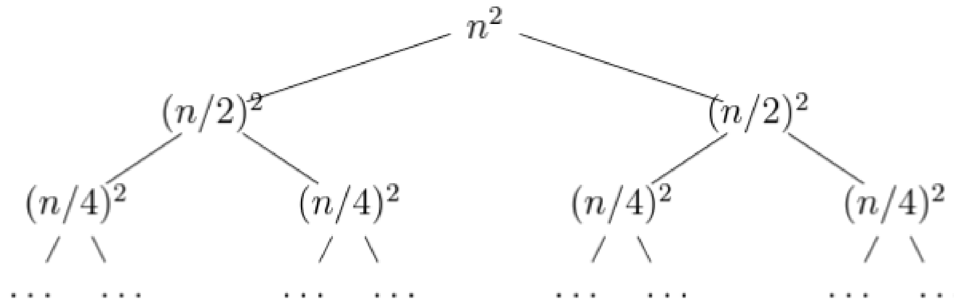


(so here in S(n) each "top row" only has ~~2S(n/2)~~ + n, and the bottom row has the next iteration, which is n/2. Since there is a 2 multiplied to S(n/2), the bottom row has 2 nodes instead of 1)

The final value of the function can be calculated by summing the value of every node in the tree. (The tree has height log(n) base 2, and the node values of each level happens to add up to just n. Plus each leaf node at the very bottom representing the base case is c, and there are n of them, so the final formula is n*log(n) + cn)

Another Example:

$$P(1) = c$$
$$P(n) = 2P(n/2) + n^2, \quad \forall n \geq 2 \ \ (n \text{ a power of } 2)$$



The lowest non-leaf nodes are at level $\log n - 1$. So the sum of all the non-leaf nodes in the tree is

$$\begin{aligned} P(n) &= \sum_{k=0}^{\log n-1} n^2 \frac{1}{2^k} = n^2 \sum_{k=0}^{\log n-1} \frac{1}{2^k} \\ &= n^2(2 - \frac{1}{2^{\log n-1}}) = n^2(2 - \frac{2}{2^{\log n}}) = n^2(2 - \frac{2}{n}) = 2n^2 - 2n \end{aligned}$$

Adding cn to cover the leaf nodes, our final closed form is $2n^2 + (c-2)n$.

Induction with Trees:

We can write proofs using induction about trees by breaking up a tree into sub trees.

Claim 47 *Let T be a binary tree, with height h and n nodes. Then $n \leq 2^{h+1} - 1$.*

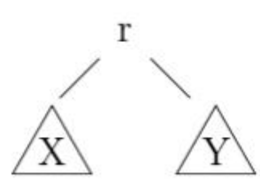
Proof by induction on h, where h is the height of the tree.

Base: The base case is a tree consisting of a single node with no edges. It has $h = 0$ and $n = 1$. Then we work out that $2^{h+1} - 1 = 2^1 - 1 = 1 = n$.

Induction: Suppose that the claim is true for all binary trees of height $< h$. Let T be a binary tree of height h ($h > 0$).

Case 1: T consists of a root plus one subtree X. X has height $h - 1$. So X contains at most $2^h - 1$ nodes. T only contains one more node (its root), so this means T contains at most 2^h nodes, which is less than $2^{h+1} - 1$.

Case 2: T consists of a root plus two subtrees X and Y. X and Y have heights p and q, both of which have to be less than h, i.e. $\leq h - 1$. X contains at most $2^{p+1} - 1$ nodes and Y contains at most $2^{q+1} - 1$ nodes, by the inductive hypothesis. But, since p and q are less than h, this means that X and Y each contain $\leq 2^h - 1$ nodes.



So the total number of nodes in T is the number of nodes in X plus the number of nodes in Y plus one (the new root node). This is $\leq 1 + (2^p - 1) + (2^q - 1) \leq 1 + 2(2^h - 1) = 1 + 2^{h+1} - 2 = 2^{h+1} - 1$

So the total number of nodes in T is $\leq 2^{h+1} - 1$, which is what we needed to show. $\square$

Same can be done with other kinds of trees, like grammar trees or trees with numerical nodes. (breaking up one tree into different sub trees and proof with different cases)