Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Applications of Trees

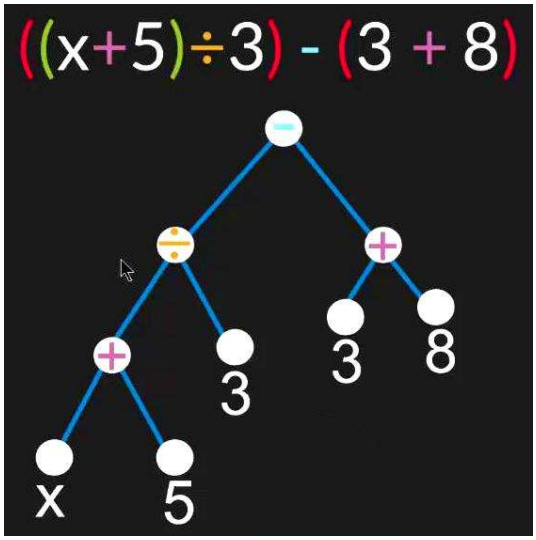
We have already had an overview of applications of trees in the introductory lecture. Here we will discuss in detail some of the ***most popular*** applications of trees in computer science. Following are some of the most popular applications of Trees:

- *Expression Trees*
- *Decision Trees*
- *Huffman Coding (Very Important)*

Expression Trees

An ***Expression Tree*** is a specific kind of a ***binary tree*** used to represent expressions. Two common types of expressions that a binary expression tree can represent are algebraic and boolean. These trees can represent expressions that contain both ***unary*** and ***binary*** operators.

The ***leaves*** of a binary expression tree are ***operands***, such as constants or variable names, and the ***internal nodes*** contain ***operators***. These particular trees happen to be binary, because all of the operations are binary, and although this is the simplest case, it is possible for nodes to have more than two children. It is also possible for a node to have only one child, as is the case with the unary operator. An expression tree can be evaluated by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees.



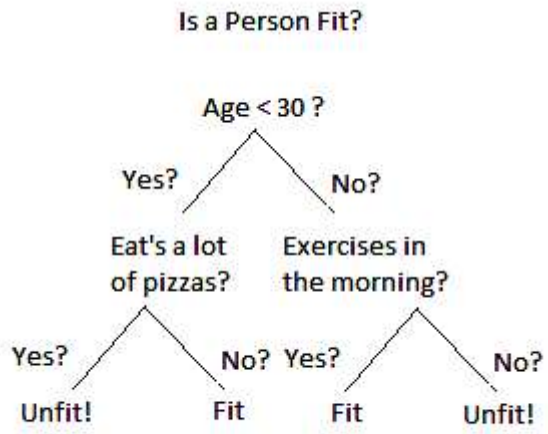
Traversal of these trees provide different notation of the expression such as ***In-Order*** traversal will produce the ***Infix Expresion***, ***Pre-Order*** traversal will give ***Prefix Expression***, and ***Post-Order*** traversal will provide ***Postfix Expression***.

Decision Trees

A ***decision tree*** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a ***"test"*** on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). ***Decision Tree algorithms*** are referred to as ***CART (Classification and Regression Trees)***.



Huffman Coding (Very Important)

Huffman coding assigns codes to characters such that the length of the code depends on the relative frequency or weight of the corresponding character. Huffman codes are of variable-length, and prefix-free (no code is prefix of any other). Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.

Huffman coding tree or ***Huffman tree*** is a binary tree in which each leaf of the tree corresponds to a letter in the given alphabet.

Define the weighted path length of a leaf to be its weight times its depth. The Huffman tree is the binary tree with minimum external path weight, i.e., the one with the minimum sum of weighted path lengths for the given set of leaves. So the goal is to build a tree with the minimum external path weight.

See an example below:

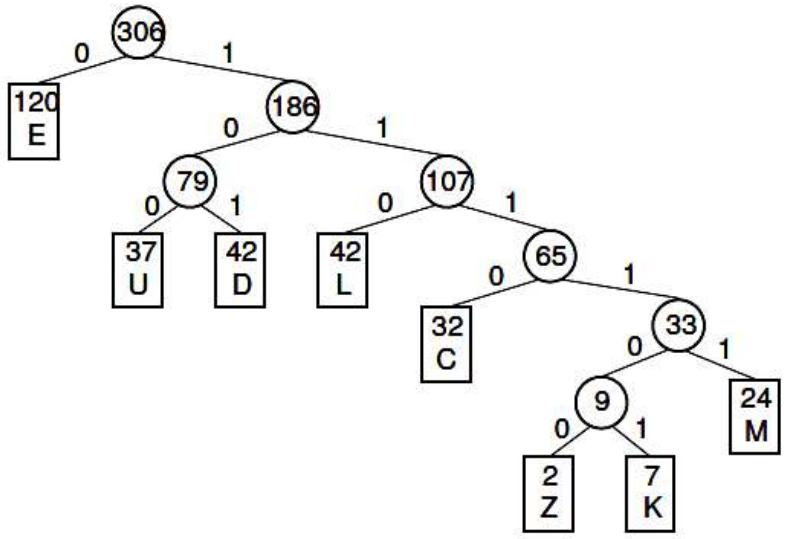
Letter frequency table:

Letter	Z	K	M	C	U	D	L	E
Frequency	2	7	24	32	37	42	42	120

Huffman code:

Letter	**Freq**	**Code**	**Bits**
E	120	**0**	1
D	42	**101**	3
L	42	**110**	3
U	37	**100**	3
C	32	**1110**	4
M	24	**11111**	5
K	7	**111101**	6
Z	2	**111100**	6

How do we get these codes?



Now Consider you have a string **`ABRACADABRA,`** you want to encode, how you will do this? Let's apply Huffman Coding.