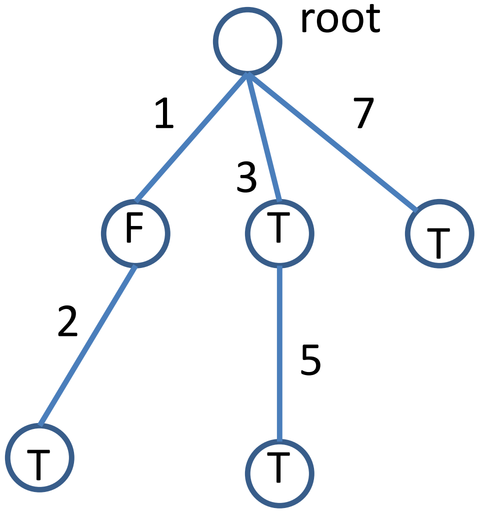
# Tries

•

A *trie* or *prefix tree* is a data structure in which the keys are broken down into a sequence of elements. Tries are most often used for strings, where the elements of the key are the letters it is composed of, or for numbers, where the elements are the digits. The word "trie" comes from "retrieval". Some authors pronounce it like "tree"; others like "trie".

To insert into a trie, we have a loop that starts at the root and iterates through the elements of the key. If the next element is x, we go to child x of the current node; if there is no such child we make one, then go there. When we get to the end of the key, we change the boolean flag of the current node to true.

With integer keys we get the next element with k%10 and replace k by k/10. This goes through the digits from right to left. Here is a picture of an integer trie storing the numbers 21, 53, 3 and 7:



The idea behind this data structure is very simple. We build trees out of nodes, where a node has one possible child for each value of the next element -- in an integer trie each node has 10 possible children; in a String trie each has 26 possible children (assuming we map everything to lower case; otherwise there are 52 possible children). Each node also carries a boolean flag, which is initialized to false.

Note that the data is stored nowhere in the structure; it is implicit in the sequence of steps needed to get to the node that represents the data.

Tries give lookups that depend only on the number of elements that make up the key -- a string of 4 letters can be looked up in a trie in 4 steps, regardless of the size of the trie.   We pay for this efficient lookup with the size of the trie.  Note that each node of a String trie has an array of 26 references and a boolean flag -- this is about 105 bytes of data for one node.

If we used to trie to hold a dictionary of 20,000 words and if the trie had another 20,000 placeholder nodes, this structure would be over 4 MB. By contract, a sorted list of 20,000 Strings averaging 6 letters each would take up about 0.2 MB and could be searched in about 14 $=\log_2(20000)$ steps and an AVL tree for this data might take up about 0.8MB, again needing about 14 steps for a search.

Questions:

a) What is the search algorithm for a trie?

b) How would you remove an element from a trie?

c) How would you make a list of the values stored in a trie?