23.6 Tries

A ***trie*** (pronounced "try") is a tree-based data structure for storing strings in order to support fast pattern matching. The main application for tries is in information retrieval. Indeed, the name "trie" comes from the word "re*trie*val." In an information retrieval application, such as a search for a certain DNA sequence in a genomic database, we are given a collection S of strings, all defined using the same alphabet.

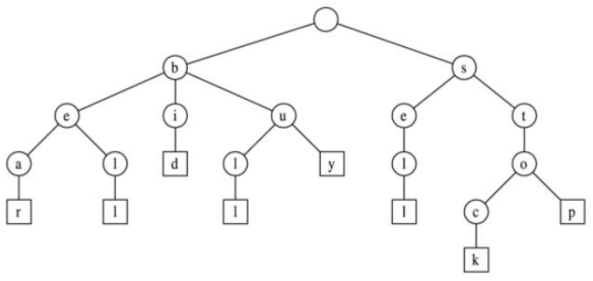
The primary query operations that tries support are pattern matching and ***prefix matching***. The latter operation involves being given a string X, and looking for all the strings in S that contain X as a prefix.

Standard tries

Let S be a set of s strings from alphabet Σ, such that no string in S is a prefix of another string. A ***standard trie*** for S is an ordered tree T with the following properties (see Figure 23.6.1):

- Each node of T, except the root, is labeled with a character of Σ.
- The ordering of the children of an internal node of T is determined by a canonical ordering of the alphabet Σ.
- T has s external nodes, each associated with a string of S, such that the concatenation of the labels of the nodes on the path from the root to an external node v of T yields the string of S associated with v.

Figure 23.6.1: Standard trie for the strings {bear, bell, bid, bull, buy, sell, stock, stop}.



Thus, a trie T represents the strings of S with paths from the root to the external nodes of T. Note the importance of assuming that no string in S is a prefix of another string. This ensures that each string of S is uniquely associated with an external node of T. We can always satisfy this assumption by adding a special character that is not in the original alphabet Σ at the end of each string.

An internal node in a standard trie T can have anywhere between 1 and d children, where d is the size of the alphabet. There is an edge going from the root r to one of its children for each character that is first in some string in the collection S. In addition, a path from the root of T to an internal node v at depth i corresponds to an i-character prefix $X[0..i-1]$ of a string X of S. In fact, for each character c that can follow the prefix $X[0..i-1]$ in a string of the set S, there is a child of v labeled with character c. In this way, a trie concisely stores the common prefixes that exist among a set of strings.

If there are only two characters in the alphabet, then the trie is essentially a binary tree, although some internal nodes may have only one child (that is, it may be an improper binary tree). In general, if there are d characters in the alphabet, then the trie will be a multi-way tree where each internal node has between 1 and d children. In addition, there are likely to be several internal nodes in a standard trie that have fewer than d children. For example, the trie shown in Figure 23.6.1 has several internal nodes with only one child. We can implement a trie with a tree storing characters at its nodes.

Theorem 23.6.1: Theorem.

A standard trie storing a collection S of s strings of total length n from an alphabet of size d has the following properties:

- Every internal node of T has at most d children
- T has s external nodes
- The height of T is equal to the length of the longest string in S
- The number of nodes of T is $O(n)$.