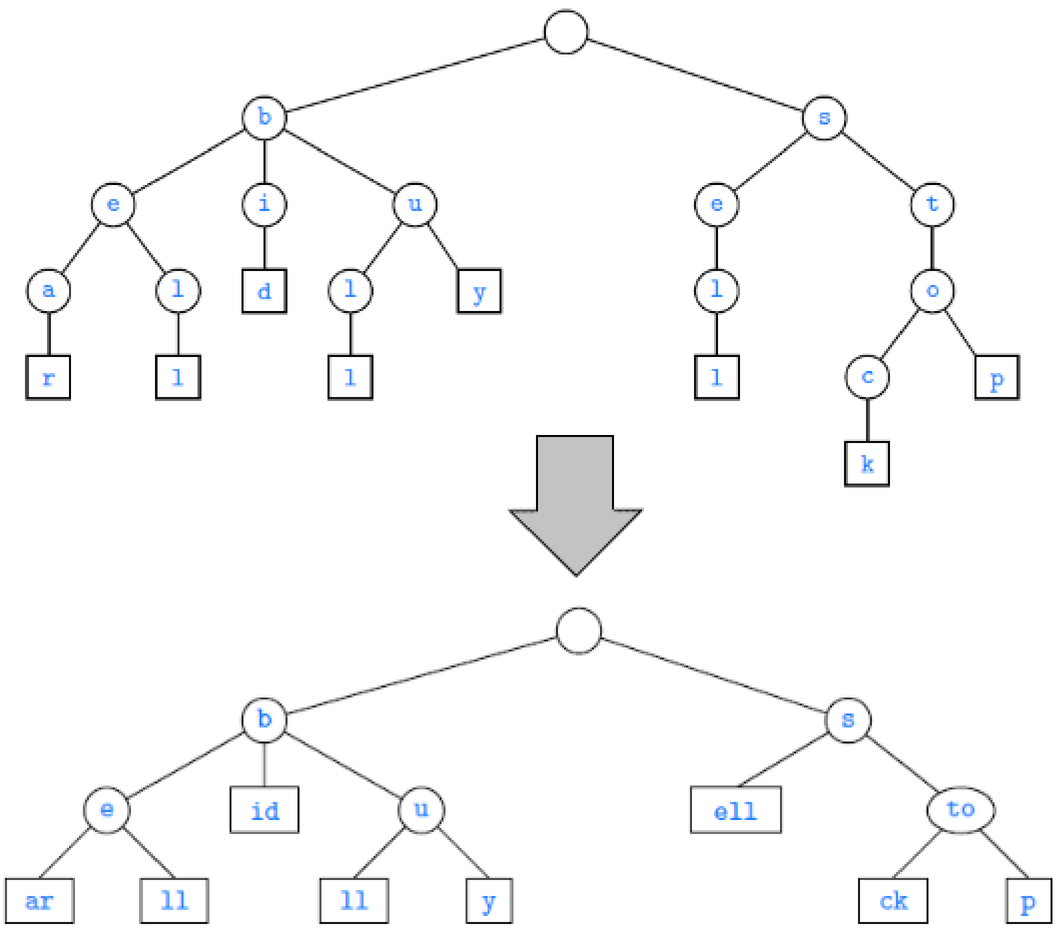
Compressed Trie

Prerequisite: Tries

Overview

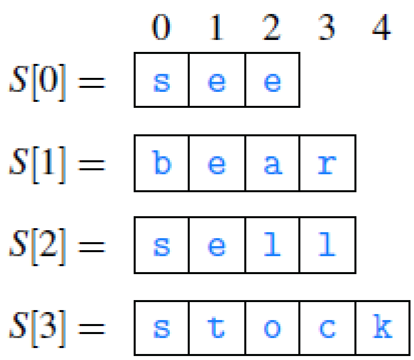
A *compressed trie* is similar to a standard trie but it ensures that each internal node in the trie has at least two children. It enforces this rule by compressing chains of single-child nodes into individual edges. Let T be a standard trie. We say that an internal node v of T is redundant if v has one child and is not the root.



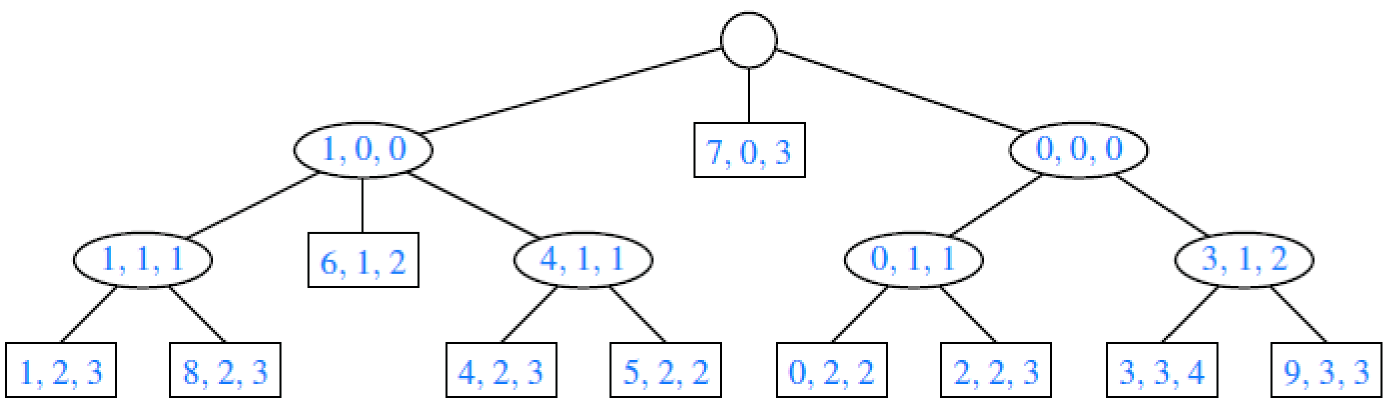
The advantage of a compressed trie over a standard trie is that the number of nodes of the compressed trie is proportional to the number of strings s and not to their total length.

Representation

Let collection S of strings is an array of strings $S[0], S[1],$ ... $S[s-1]$. Instead of storing the label X of a node explicitly, we represent it implicitly by a combination of three integers (i, j, k), such that $X = S[i][j \,..\, k]$ that is, X is the substring of $S[i]$ consisting of the characters from the j^{th} to the k^{th} inclusive.



(a)



This additional compression scheme allows us to reduce the total space for the trie itself from $O(n)$ for the standard trie to $O(s)$ for the compressed trie, where n is the total length of the strings in S and s is the number of strings in S.

Time and Space Complexity

The number of nodes of T is $O(s)$, where s is number of strings from an alphabet of size d.

Searching in a compressed trie is not necessarily faster than in a standard tree, since there is still need to compare every character of the desired pattern with the potentially multi-character labels while traversing paths in the trie.

Applications

A compressed trie is truly advantageous only when it is used as an *auxiliary index* structure over a collection of strings already stored in a primary structure, and is not required to actually store all the characters of the strings in the collection.

References

- Data Structures and Algorithms in Java Book