Trie Structures

Jeffrey A. Barnett
jbb@notatt.com

Keywords: Data structures, information retrieval, tries, storage complexity.

1 Summary

Trie structures [3] have been part of the computer science toolkit for a long time. The name comes from the word retrieval but is often pronounced "try" instead of "tree." A trie provides an efficient method to store sets of strings. Common initial sequences are shared. For example, the strings "CAB," "CAR," "CART," "CAN," and "CANE," are stored as shown in Figure 1 where asterisks indicate string-terminating characters. In this example, seventeen string characters are stored in seven nodes. The savings in storage space can be quite substantial in some cases, e.g., storing a large dictionary, even though a node structure is substantially larger than a single character.

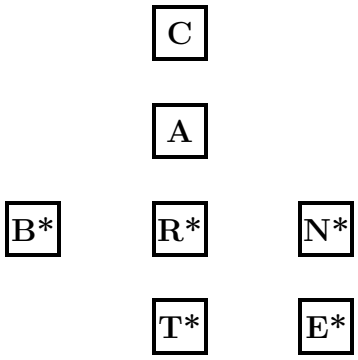

Figure 1: An example trie.

Retrieval time—the time to see if a string is in a trie or add the string if it isn't—is bounded by the number of characters in the key string times the number of characters in the alphabet. Therefore, tries provide an access method that is linear in the length of the string. A novel theorem-proving application is reported in [2] where trie structures are used to store prime implicants in order to save storage, speed up access, and do subsumption operations.

This note makes two modest contributions. The first is the idea of sharing common final sequences. This provides another source of potential space savings. The second is a rough analysis of the amount of storage necessary for tries. The analysis investigates the case where all permutations of n characters are stored in a trie. If a trie is not used, $n \cdot n!$ characters are necessary because there are $n!$ permutations of length n each.

It is shown that the trie size is $\lfloor e \cdot n! - 1 \rfloor$ nodes, where $e = 2.718\ldots$ is the base of the natural logarithms, when initial sequence sharing is implemented. Only $n \cdot 2^{n-1}$ nodes are necessary when both initial and final sequence sharing is used. Since storing all permutations exposes the densest set of sharing possibilities, these results provide upper bounds on the storage reductions offered by tries.

2 Node Implementation

A trie node can have zero, one, or more descendants as depicted in Figure 1. Therefore, an implementation must provide for a variable branching factor. Perhaps the simplest node structure, shown in Figure 2, is to represent each

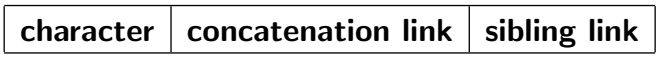

Figure 2: A trie node template.

node as a triple that contains the character, a link to following characters, and a link to its siblings.

The example in Figure 1 is implemented by the trie in Figure 3 where "/" indicates a null link, i.e., it is a terminator, and asterisks mark terminal characters. The interpretation is that all siblings are direct descendants of the same initial string. Siblings appear in sorted order for two reasons: (1) it speeds up the search to retrieve a string because the search can stop with a negative result as soon as a "larger" character is found and (2) a

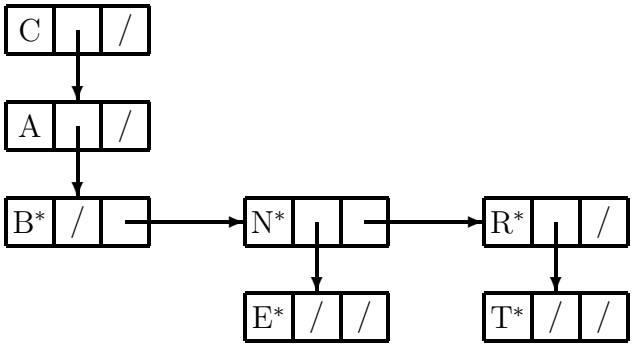

Figure 3: The example trie implementation.

canonical order increases the probability of final sequence sharing as will become evident below.

3 Trie Storage Complexity

The storage complexity of a trie that represents all permutations of n characters is calculated here. The term, n-trie, is used to denote a trie that encodes all permutations of n characters. Only initial sequences are shared. The result is developed in three steps: (1) the recurrence relation $s_n = n(1 + s_{n-1})$ is established, where s_n is the number of nodes in an n-trie, (2) it is shown that $s_n = \sum_{c=0}^{n-1} \frac{n!}{c!}$, and (3) the formula, $s_n = \lfloor e \cdot n! - 1 \rfloor$, is verified.

Figure 4 shows n-tries for $n = 1$, 2, and 3 with respective sizes $s_1 = 1$, $s_2 = 4$, and $s_n = 15$ nodes. Assume that we desire to construct an n-trie on the characters $C = \{c_1, \ldots, c_n\}$. It is clear that each c_i will be an initial character. The final strings concatenated to each initial c_i are all of the permutations of the characters $C \backslash c_i$. That trie is simply a $(n-1)$-trie with size s_{n-1}. Therefore,

$$\begin{aligned} s_n &= n + n s_{n-1} \\ &= n(1 + s_{n-1}). \end{aligned}$$

The general solution to this recurrence relation is

$$s_n = \sum_{c=0}^{n-1} \frac{n!}{c!} + b \cdot n!,$$

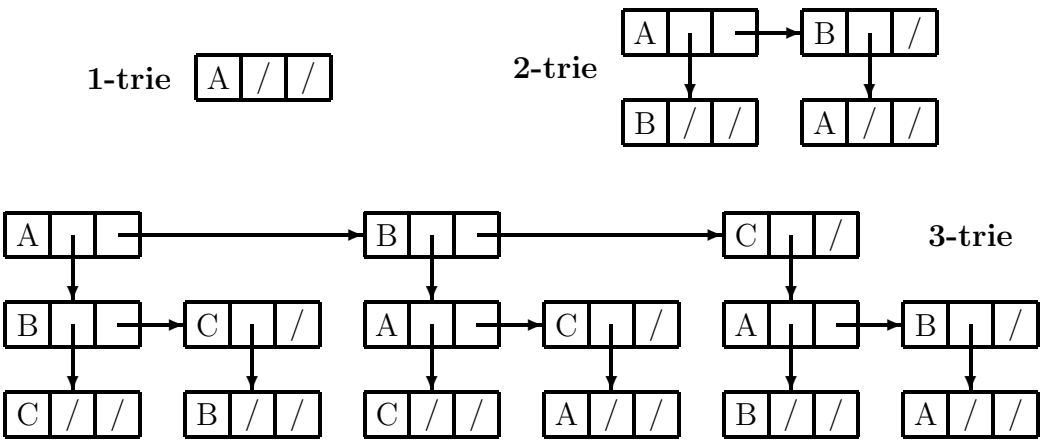

Figure 4: Examples of 1-, 2-, and 3-tries.

where b is an arbitrary constant. This result is easily checked by substitution. Since $s_1 = 1$, it follows that $b = 0$ and the desired particular solution is

$$s_n = \sum_{c=0}^{n-1} \frac{n!}{c!}.$$

To prove $s_n = \lfloor e \cdot n! - 1 \rfloor$, it suffices to show that $0 \leq e \cdot n! - 1 - s_n < 1$. Note that $e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \cdots$ and proceed as follows:

$$\begin{aligned} e \cdot n! - 1 - s_n &= \sum_{c=0}^{\infty} \frac{n!}{c!} - 1 - \sum_{c=0}^{n-1} \frac{n!}{c!} \\ &= \sum_{c=n+1}^{\infty} \frac{n!}{c!} \end{aligned}$$

so the value is surely positive. It remains to show that it is less than 1.

$$\begin{aligned} \sum_{c=n+1}^{\infty} \frac{n!}{c!} &= \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} + \cdots \\ &< \frac{1}{n+1} + \frac{1}{(n+1)^2} + \cdots \qquad (1) \\ &= \frac{1}{1 - \frac{1}{n+1}} - 1 \qquad (2) \\ &= \frac{1}{n} \\ &< 1. \end{aligned}$$

Line (2) follows because (1) is the sum of a geometric progression. This establishes that $s_n = \lfloor e \cdot n! - 1 \rfloor$.

It is interesting to note that an almost identical proof can be used to show that e is irrational [1]. Assume that $e = m/n$ for integers m and n. Then $e \cdot n! = \sum_{c \geq 0} \frac{n!}{c!}$ is an integer. Further, $\sum_{c=0}^{c=n} \frac{n!}{c!}$ must be an integer. Therefore, the difference of the sums must be an integer too but, as we have just seen, that difference lies between 0 and 1. Contradiction. So e is irrational.

4 A More Compact Representation

In this section, tries that share final sequences as well as initial sequences are analyzed and it is shown that $z_n = n \cdot 2^{n-1}$ nodes will be used to represent an n-trie. Figure 5 depicts a 4-trie, for permutations of ABCD, with a complete

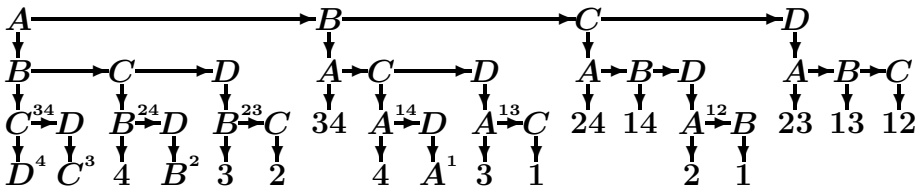

Figure 5: A example of a 4-trie with initial and final sequence sharing.

sharing implementation. Vertical lines are concatenation links and horizontal lines are sibling links. Missing lines are null links.

If a link terminates at a number, it is a link to the node with that number as a superscript. Thus, only those nodes labelled by characters exist in the implementation: $z_4 = 32 = 4 \cdot 2^{4-1}$. The right-most node, a C, has a concatenation link to the A with the superscript 12 and that node represents the final sequences AB and BA. Both are concatenated, by reference, to the initial string DC as well as CD. Note that both the A and B nodes share their final B and A nodes, respectively the nodes with superscripts 2 and 1, with other permutations.

A structural observation is in order before the complexity result can be derived. The crucial insight is that if a node is referenced multiple times, it is always referenced by concatenation (down) links. Consider the permutations, $h_1, \ldots, h_c, t_1, \ldots, t_{n-c}$ and $h_{i_1}, \ldots, h_{i_c}, t_1, \ldots, t_{n-c}$, where $i_1, \ldots, i_c$ is a permutation of $1, \ldots, c$, that have the final t sequence in common. It is clear that both h sequences have all $(n-c)!$ final sequences formed from the

permutations of $t_1, \ldots, t_{n-c}$ in common too. Since sibling links are ordered (alphabetically in the examples), each of the subtries made of those characters will be identically arranged. Therefore, a single pointer can concatenate all of the final permutations with a single reference. That in fact will be the only source of node sharing in the implementations of n-tries.

Another way to state the observation is that there is one final c-trie for every subset, S, of the n characters, where $|S| = c$ and $1 \leq c \leq n$. That subtrie represents all permutations of the c characters in S. The unique contribution of such a subtrie to z_n is just the c top-level sibling nodes, one for each character. Therefore,

$$z_n \;=\; \sum_{c=1}^{n} c \cdot \binom{n}{c}$$

and this form has the well-known evaluation $z_n = n \cdot 2^{n-1}$.

References

[1] M. Aigner and G. M. Ziegler, *Proofs from the Book*, Springer, Chapter 6, 1991.

[2] J. de Kleer, An improved incremental algorithm for generating prime implicants, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 780–785 1992.

[3] E. Fredkin, Trie memory, *Communications of the Association for Computer Machinery* 3, pp. 490–500, 1960.