

TRIE

Maxime Crochemore

King's College London and Université Paris-Est, <http://www.dcs.kcl.ac.uk/staff/mac/>

Thierry Lecroq

Université de Rouen, <http://monge.univ-mlv.fr/~lecroq>

SYNONYMS

Prefix tree

DEFINITION

A trie is a rooted tree used for storing associative arrays where keys are usually strings. Edges are often labeled by individual symbols. Then common prefixes are factorized. Each node of the trie is associated with a prefix of a string of the set of strings: concatenation of the labels of the path from the root to the node. The root is associated with the empty string. Strings of the set are stored in terminal nodes (leaves) but not in internal nodes. A trie can be seen as a Deterministic Finite Automaton.

Tries can be compacted. To get a compact trie from a trie, internal nodes with exactly one successor are removed. Then labels of edges between remaining nodes are concatenated. Thus:

Edges are labeled by strings.

- Internal nodes have at least two children.
- Edges outgoing an internal node are labeled by strings starting with different symbols.

HISTORICAL BACKGROUND

The word “trie” comes from information *retrieval* and was suggested by Fredkin [1]. Flajolet and Sedgewick [5] provide an average case analysis of tries. The reader can refer to [3] for further details on tries.

SCIENTIFIC FUNDAMENTALS

In binary search trees, keys are stored in all the nodes of the tree and the search method is based on comparison between keys. In tries, keys are stored in the leaves and the search method involves left-to-right comparison of prefixes of the keys.

The trie of the set of strings $X = \{\text{in, integer, interval, string, structure}\}$ is presented Figure 1. Note that the space sign \square has been added at the end of each of the strings of X so that no string of X is a prefix of another string of X . Then each string of X is associated with a leaf of the trie (not with an internal node).

We consider that the strings are build over an alphabet of size σ .

Construction

The algorithm $\text{TRIE}(X)$ shown in Figure 2, builds the trie containing all the strings in the set X . It works by inserting all the strings of X successively in the trie starting with the tree consisting with a single node (the root). Then for each string $x \in X$ it spells the longest prefix of x corresponding to an existing path from the root of the trie. When this longest prefix is found, it creates the nodes and the edges of the remaining suffix of x .

The sum of the lengths of all the strings of X is denoted by M . Then, the algorithm $\text{TRIE}(X)$ run in time $O(M)$ when the branching time from a node with a given symbol is constant or $O(M \times \log \sigma)$ when the branching time

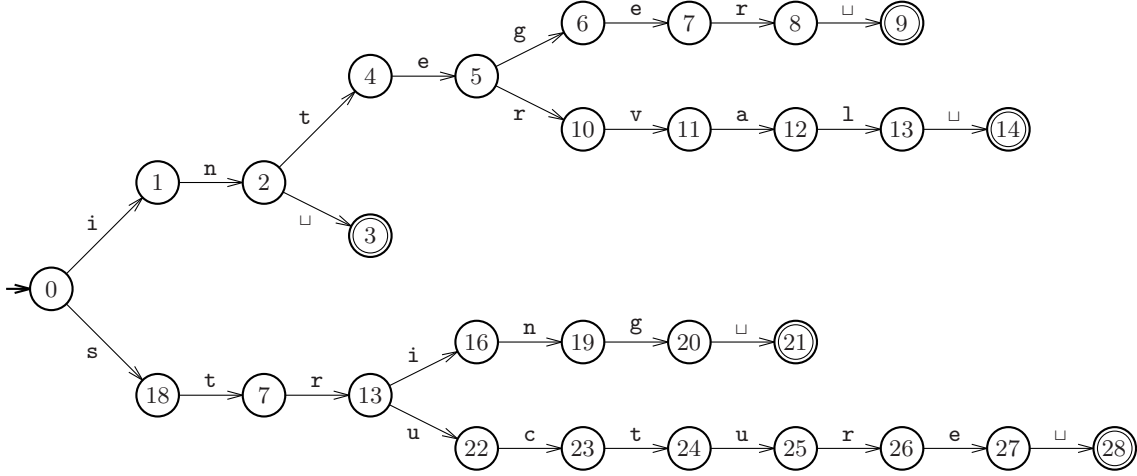


Figure 1: Trie of $X = \{\text{in, integer, interval, string, structure}\}$.

```

TRIE( $X$ )
1   $root \leftarrow$  new node
2  for each string  $x \in X$  do
3     $p \leftarrow root$ 
4    for  $i \leftarrow 0$  to  $|x| - 1$  do
5       $q \leftarrow \text{TARGET}(p, x[i])$ 
6      if  $q = \text{NIL}$  then
7         $q \leftarrow$  new node
8         $\text{Succ}[p] \leftarrow \text{Succ}[p] \cup \{(x[i], q)\}$ 
9       $p \leftarrow q$ 
10    $\text{info}[p] \leftarrow x$ 
11  return  $root$ 

```

Figure 2: Algorithm that builds the trie containing of the string of a set X .

depends on the alphabet size (see Implementation below).

Searching

The algorithm $\text{ISINTRIE}(root, x)$, see Figure 3, tests if the string x is present in the trie and consequently if the string x is a prefix of strings represented by the trie. It works, similarly to the creation of the trie, by spelling, from the root of the trie, the longest prefix of x corresponding to a branch in the trie. If this longest prefix is x itself, then the algorithm returns TRUE and the string x belongs to the trie, otherwise the algorithm returns FALSE and the string is not a prefix of any string in the set. The algorithm $\text{ISINTRIE}(root, x)$ works in time $O(|x|)$ or $O(|x| \times \log \sigma)$ depending on the branching time.

Implementation

There exist different possible representations of a trie, each with different branching times. It is possible to use a transition table whose size is the product of the number of nodes times the size of the alphabet. The branching time, for finding the successor of a given node with a given symbol, is then constant.

The use of adjacency lists has the advantage to minimize the required space but the branching time depends on the alphabet size. It can be as low as $\log(\sigma)$ if the alphabet is ordered and outgoing edges are stored in a balanced search tree. A representation by binary tree is achieved by using a “first child — right sibling” representation of the trie.

```

ISINTRIE( $p, x$ )
1 if  $x$  is empty then
2   return TRUE
3 else  $q \leftarrow$  TARGET( $p, x[0]$ )
4   if  $q = \text{NIL}$  then
5     return FALSE
6   else return ISINTRIE( $q, x[1 \dots |x| - 1]$ )

```

Figure 3: Algorithm that enables to test if a string x belongs to a trie.

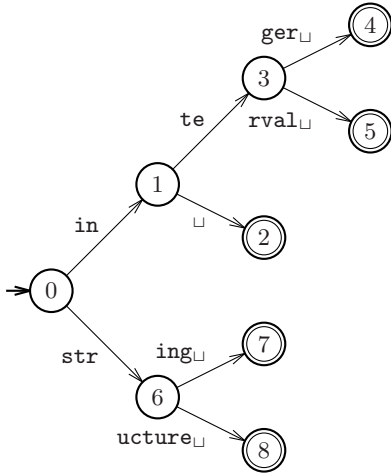


Figure 4: Compact trie of $X = \{\text{in}, \text{integer}, \text{interval}, \text{string}, \text{structure}\}$.

Hashing techniques can be used as a good trade-off between transition tables and adjacency lists. The branching time is then constant on average.

A mixed technique known as the “deep shallow” technique consists in representing the nodes up to a certain level k with a transition table and to use adjacency lists for the nodes with level greater than k . Most of the times nodes with small level have many successors while nodes with large level have only one successor.

Sorting

A trie can be used to sort a set of strings by doing the following: insert all the strings in the trie and output them in lexicographically increasing order by applying a pre-order traversal of the trie (respectively lexicographically decreasing order by applying a post-order traversal of the trie).

Compact tries

The compact trie of the set of strings $X = \{\text{in}, \text{integer}, \text{interval}, \text{string}, \text{structure}\}$ is presented Figure 4. It is obtained from the trie of Figure 1 by removing internal nodes with exactly one successor. Then labels of edges between remaining nodes are concatenated.

PATRICIA trees

Morrison [4] designed specific compact tries known as PATRICIA trees. The PATRICIA tree of the set of strings $X = \{\text{in}, \text{integer}, \text{interval}, \text{string}, \text{structure}\}$ is presented Figure 5. PATRICIA trees are binary trees that consider the binary encoding of the strings. On Figure 5 nodes 0, 1 and 3 actually point to in , nodes 4 and 7 point to integer , node 8 points to interval , nodes 2 and 5 point to string and node 6 points to structure . Small numbers close to the nodes are skip numbers, they indicate on which bit the branching has to be decided. The skip number of node 0 is 1: it corresponds to bit at position 1 (bolded on Figure 5), where

Decimal and binary ASCII codes of the symbols

	7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0			
␣	32	0	0	1	0	0	0	0	i	105	0	1	1	0	1	0	0	1	t	116	0	1	1	1	0	1	0	0	
a	97	0	1	1	0	0	0	0	1	l	108	0	1	1	0	0	1	0	0	u	117	0	1	1	1	0	1	0	1
c	99	0	1	1	0	0	0	1	1	n	110	0	1	1	0	0	1	1	0	v	118	0	1	1	1	0	1	1	0
e	101	0	1	1	0	0	1	0	1	r	114	0	1	1	1	0	0	1	0										
g	103	0	1	1	0	0	1	1	1	s	115	0	1	1	1	0	0	1	1										

Strings of X as sequences of bits

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
in_{\sqcup}	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0																
$integer_{\sqcup}$	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	1	0	...			
$interval_{\sqcup}$	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	0	0	...			
$string_{\sqcup}$	1	1	0	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	0	1	1	1	...			
$structure_{\sqcup}$	1	1	0	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	1	1	1	0	1	1	0	0	...			

PATRICIA tree of X

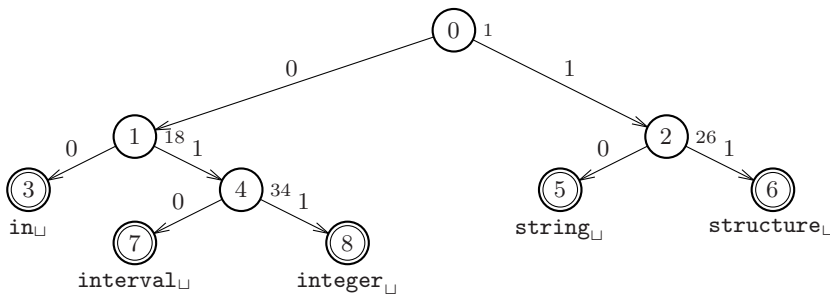


Figure 5: PATRICIA tree of $X = \{in_{\sqcup}, integer_{\sqcup}, interval_{\sqcup}, string_{\sqcup}, structure_{\sqcup}\}$.

is the leftmost difference in the bit strings representing the strings of X . The three strings in_{\sqcup} , $integer_{\sqcup}$ and $interval_{\sqcup}$ possess a 0 so they are on the left and $string_{\sqcup}$ and $structure_{\sqcup}$ possess a 1 so they are on the right. The skip number of node 1 is 18: it corresponds to bit at position 18 (bolded on Figure 5), where is the leftmost difference in the bit strings representing the three strings in_{\sqcup} , $integer_{\sqcup}$ and $interval_{\sqcup}$. The string in_{\sqcup} has a 0 so it is on the left and $integer_{\sqcup}$ and $interval_{\sqcup}$ possess a 1 so they are on the right. The skip number of node 2 is 26: it corresponds to bit at position 26, (bolded on Figure 5), where is the leftmost difference in the bit strings representing the two strings $string_{\sqcup}$ and $structure_{\sqcup}$. The string $string_{\sqcup}$ has a 0 so it is on the left and $structure_{\sqcup}$ possesses a 1 so it is on the right. The skip number of node 4 is 34: it corresponds to bit at position 34 (bolded on Figure 5), where is the leftmost difference in the bit strings representing the two strings $integer_{\sqcup}$ and $interval_{\sqcup}$. The string $interval_{\sqcup}$ has a 0 so it is on the left and $integer_{\sqcup}$ possesses a 1 so it is on the right. The skip number of a node (different from the root) in a PATRICIA tree is never larger than its parent skip number.

The reader can refer to [2] for further details on PATRICA trees.

KEY APPLICATIONS*

Tries are used for dictionary representation including spell checking systems or predictive text for mobile phones (T9 system for instance). They are use for text compression (Ziv and Lempel compression schemes), multiple string matching (or multi key search). They are at the basis of the Burstsor for sorting large sets of strings.

CROSS REFERENCE*

Data dictionary, Suffix tree, Text compression.

RECOMMENDED READING

Between 5 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

- [1] E. Fredkin, Trie Memory, *Communications of the ACM* 3(9):490–499, 1960.
- [2] G.H. Gonnet and R. Baeza-Yates, Handbook of Algorithms and Data Structures – In Pascal and C, Addison-Wesley, 2nd edition, 1991.
- [3] D. E. Knuth, *The Art of Computer Programming*, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997, Section 6.3: Digital Searching, pp. 492-512.
- [4] D. R. Morrison, PATRICIA – Practical Algorithm to Retrieve Information Coded in Alphanumeric, *Journal of the ACM* 15(4):514–534, 1968.
- [5] R. Sedgewick and Ph. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.