HAL
open science

Suffix tree

Maxime Crochemore, Thierry Lecroq

To cite this version:

Maxime Crochemore, Thierry Lecroq. Suffix tree. Ling Liu and M. Tamer Özsu. Encyclopedia of Database Systems, Springer Verlag, pp.2876-2880, 2009. hal-00470102

HAL Id: hal-00470102

https://hal.science/hal-00470102

Submitted on 19 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SUFFIX TREE

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

Compact suffix trie

DEFINITION

The suffix tree $\mathcal{S}(y)$ of a non-empty string y of length n is a compact trie representing all the suffixes of the string.
The suffix tree of y is defined by the following properties:

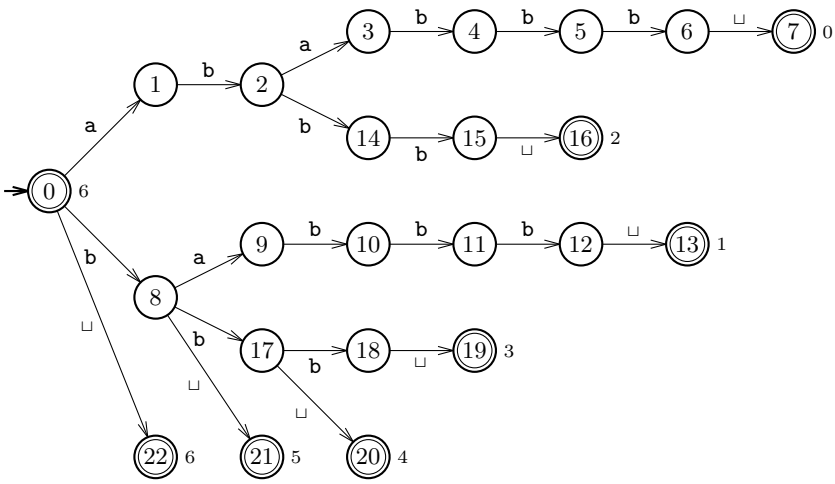
All branches of $\mathcal{S}(y)$ are labeled by all suffixes of y.

- Edges of $\mathcal{S}(y)$ are labeled by strings.
- Internal nodes of $\mathcal{S}(y)$ have at least two children.
- Edges outgoing an internal node are labeled by segments starting with different letters.
- The segments are represented by their starting position on y and their lengths.

Moreover, it is assumed that y ends with a symbol occurring nowhere else in it (the space sign $\sqcup$ is used in the examples of the present entry). This avoids marking nodes, and implies that $\mathcal{S}(y)$ has exactly n leaves (number of non-empty suffixes).
All the properties then imply that the total size of $\mathcal{S}(y)$ is $O(n)$, which makes it possible to design a linear-time construction of the suffix tree.

HISTORICAL BACKGROUND

The first linear time algorithm for building a suffix tree in from Weiner [15] but it requires quadratic space: $O(n \times \sigma)$ where σ is the size of the alphabet. The first linear time and space algorithm for building a suffix tree is from McCreight [13]. It works "off-line" ie it inserts the suffixes from the longest one to the shortest one. A strictly sequential version of the construction of the suffix tree was described by Ukkonen [14]. When the alphabet is potentially infinite, the construction algorithms of the suffix tree can be implemented to run in time $O(n \log \sigma)$ and then are optimal since they imply an ordering on the letters of the alphabet. On particular integer alphabets, Farach [5] showed that the construction can be done in linear time.
The minimization of the suffix trie gives the suffix automaton. The suffix automaton of a string is also known under the name of *DAWG*, for *Directed Acyclic Word Graph*. Its linearity was discovered by Blumer *et al.* (see [1]), who gave a linear construction (on a fixed alphabet). The minimality of the structure as an automaton is from Crochemore [2] who showed how to build with the same complexity the factor automaton of a text
The compaction of the suffix automaton gives the compact suffix automaton (see [1]), A direct construction algorithm of the compact suffix automaton was presented by Crochemore and Vérin [4]. The same structure arises when minimizing the suffix tree.
The suffix array of the string y consists of both the permutation of positions on the text that gives the sorted list of suffixes and the corresponding array of lengths of their longest common prefixes (LCP). The suffix array of a



i	0	1	2	3	4	5	6
$y[i]$	a	b	a	b	b	b	␣

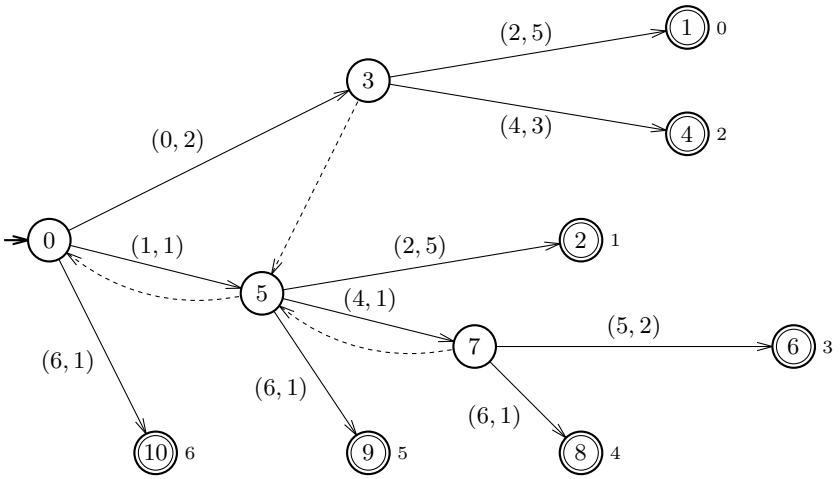
Figure 1: Suffix trie of `ababbb`. Nodes are numbered in the order of creation. The small numbers closed to each leaf correspond to the position of the suffix associated to the leaf.

string, with the associated search algorithm based on the knowledge of the common prefixes is from Manber and Myers [12]. It can be built in linear time on integer alphabets (see [8, 9, 10]).
For the implementation of index structures in external memory, the reader can refer to Ferragina and Grossi [6].

SCIENTIFIC FUNDAMENTALS

Suffix trees

The suffix tree $\mathcal{S}(y)$ of the string $y = \texttt{ababbb}\sqcup$ is presented Figure 2. It can be seen as a compaction of the suffix trie $\mathcal{T}(y)$ of y given Figure 1.
Nodes of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with segments of y. Leaves of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with suffixes of y. An output is defined, for each leaf, which is the starting position of the suffix in y.
The two structures can be built by successively inserting the suffixes of y from the longest to the shortest.
In the suffix trie $\mathcal{T}(y)$ of a string y of length n there exist n paths from the root to the n leaves: each path spells a different non-empty suffix of y. Edges are labeled by exactly one symbol. The suffix trie can have a quadratic number of nodes since the sum of the lengths of all the suffixes of y is quadratic.
To get the suffix tree from the suffix trie, internal nodes with exactly one successor are removed. Then labels of edges between remaining nodes are concatenated. Edges are now labeled with strings. This gives a linear number of nodes since there are exactly n leaves and since every internal node (called a fork) has at least two successors, there can be at most $n-1$ forks. This also gives a linear number of edges.
Now, in order that the space requirement becomes linear, since the labels of the edges are all segments of y, a segment $y[i\,..\,i+\ell-1]$ is represented by the pair (i,ℓ). Thus each edge can be represented in constant space. This technique requires to have y residing in main memory.
Overall, there is a linear number of nodes and a linear number of edges, each node and each edge can be represented in constant space thus the suffix tree requires a linear space.
There exist several direct linear time construction algorithms of the suffix tree that avoid the construction of the suffix trie followed by its compaction.
The McCreight algorithm [13] directly constructs the suffix tree of the string y by successively inserting the suffixes of y from the longest one to the shortest one. The insertion of the suffix of y beginning at position i (ie $y[i\,..\,n-1]$) consist first in locating (creating it if necessary) the fork associated with the longest prefix of $y[i\,..\,n-1]$ common with a longest suffix of y: $y[0\,..\,n-1], y[1\,..\,n-1], \ldots,$ or $y[i-1\,..\,n-1]$. Let us call the head u such a longest prefix and the tail v the remaining suffix such that $y[i\,..\,n-1] = uv$. Once the fork p associated with u has been



i	0	1	2	3	4	5	6
$y[i]$	a	b	a	b	b	b	␣

Figure 2: Suffix tree of `ababbb`. Nodes are numbered in the order of creation. The small numbers closed to each leaf correspond to the position of the suffix associated to the leaf.

located it is enough to add a new leaf q labeled by i and a new edge labeled by $(i + |u|, |v|)$ between p and q to complete the insertion of $y[i \,..\, n-1]$ into the structure. The reader can refer to [3] for further details.
The linear time of the construction is achieved by using a function called suffix link defined on the forks as follows: if fork p is identified with segment av, $a \in A$ and $v \in V^*$, then $s_y(p) = q$ where fork q is identified with v.
Suffix links are represented by dotted arrows on Figure 2.
The suffix links create shortcuts that are used to accelerate heads computations. If the head of $y[i-1 \,..\, n-1]$ is of the form au ($a \in A, u \in V^*$) then u is a prefix of the head of $y[i \,..\, n-1]$. Therefore, using suffix links, the insertion of the suffix $y[i \,..\, n-1]$ consists first in finding the fork corresponding to the head of $y[i \,..\, n-1]$ (starting from suffix link of the fork associated with au) and then in inserting the tail of $y[i \,..\, n-1]$ from this fork.
Ukkonen algorithm [14] works on-line ie it builds the suffix tree of y processing the symbols of y from the first to the last. It also uses suffix links to achieve a linear time computation.
The Weiner, McCreight and Ukkonen works in $O(n)$ time whenever $O(n \times \sigma)$ space is used. If only $O(n)$ space is used then the $O(n)$ time bound should be replaced by $O(n \times \log \min\{n, \sigma\})$. This is due to the access time for a specific edge stored in each nodes. The reader can refer to [11] for specific optimized implementations of suffix trees.
For particular integer alphabets, if the alphabet of y is in the interval $[1..n^c]$ for some constant c, Farach [5] showed that the construction can be done in linear time.
Generalized suffix trees are used to represent all the suffixes of all the strings of a set of strings.
Word suffix trees can be used when processing natural languages in order to represent only suffixes starting after separators such as space or line feed.

Indexes

The suffix tree serves as a full index on the string: it provides a direct access to all segments of the string, and gives the positions of all their occurrences in the string. An index on y can be considered as an abstract data type whose basic set is the set of all the segments of y, and that possesses operations giving access to information relative to these segments. The utility of considering the suffixes of a string for this kind of application comes from the obvious remark that every segment of a string is the prefix of a suffix of the string
Once the suffix tree of a text y is built, searching for x in y remains to spell x along a branch of the tree. If this walk is successful the positions of the pattern can be output. Otherwise, x does not occur in y.
Any kind of trie that represents the suffixes of a string can be used to search it. But the suffix tree has additional features which imply that its size is linear.
We consider four operations relative to the segments of a string y: the membership, the first position, the number of occurrences and the list of positions.

The first operation on an index is the membership of a string x to the index, that is to say the question to know whether x is a segment of y. This question can be specified in two complementary ways whether we expect to find an occurrence of x in y or not. If x does not occur in y, it is often interesting in practice to know the longest beginning of x that is a segment of y. This is the type of usual answer necessary for the sequential search tools in a text editor.
The methods produce without large modification the position of an occurrence of x, and even the position of the first or last occurrence of x in y.
Knowing that x is in the index, another relevant information is constituted by its number of occurrences in y. This information can differently direct the ulterior searches.
Finally, with the same assumption than previously, a complete information on the localization of x in y is supplied by the list of positions of its occurrences.
Suffix trees can easily answer these questions. It is enough to spell x from the root of $\mathcal{S}(y)$. If it is not possible, then x does not occur in y. Whenever x occurs in y, let w be the shortest segment of y that is such x is a prefix of w and w is associated with a node p of $\mathcal{S}(y)$. Then the number of leaves of the subtree rooted in the node p gives the number of occurrences of x in y. The smallest (respectively largest) of these leaves gives the position of the first (resp. last) position of x in y. The list of the number of these leaves gives the list of position of x in y.

KEY APPLICATIONS*

Suffix trees are used to solve string searching problems mainly when the text into which a pattern has to be found is fixed. It is also used in other string related problems such as Longest Repeated Substring, Longest Common Substring. It can be used to perform text compression. Gusfield [7] gives many applications of suffix trees in computational biology.

CROSS REFERENCE*

Trie

RECOMMENDED READING

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

[1] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen and J. Seiferas, The smallest automaton recognizing the subwords of a text *Theor. Comput. Sci.* 40(1):31–55, 1985.

[2] M. Crochemore, Transducers and repetitions, *Theor. Comput. Sci.* 45(1):63–86, 1986.

[3] M. Crochemore, C. Hancart and T. Lecroq, *Algorithms on strings*, Cambridge University Press, 2007.

[4] M. Crochemore and R. Vérin, On compact directed acyclic word graphs, in *Structures in Logic et Computer Science*, LNCS 1261, 192–211, 1997.

[5] M. Farach, Optimal suffix tree construction with large alphabets, in *Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science*, Miami Beach, Florida, 137–143, 1997.

[6] P. Ferragina and R. Grossi, The string B-tree: A new data structure for string search in external memory et its applications, *J. Assoc. Comput. Mach.* 46:236-280, 1999.

[7] D. Gusfield, *Algorithms on strings, trees and sequences*, Cambridge University Press, 1997.

[8] J. Kärkkäinen and P. Sanders, Simple linear work suffix array construction, in *ICALP*, LNCS 2719, 943–955, 2003.

[9] D. K. Kim, J. S. Sim, H. Park and K. Park, Linear-Time Construction of Suffix Arrays, in *CPM03*, LNCS 2676, 186–199, 2003.

[10] P. Ko and S. Aluru, Space Efficient Linear Time Construction of Suffix Arrays, in *CPM03*, LNCS 2676, 200–210, 2003.

[11] S. Kurtz, Reducing the space requirement of suffix trees, *Softw., Pract. Exper.* 29(13):1149–1171 (1999)

[12] U. Manber et G. Myers, Suffix arrays: a new method for on-line string searches, *SIAM J. Comput.* 22(5):935–948, 1993.

[13] E. M. McCreight, A space-economical suffix tree construction algorithm, *J. Algorithms* 23(2):262–272, 1976.
[14] E. Ukkonen, On-line construction of suffix trees, *Algorithmica* 14(3):249–260, 1995.
[15] P. Weiner, Linear pattern matching algorithm, in *Proceedings of the 14th Annual IEEE Symposium on Switching et Automata Theory*, Washington, DC, 1–11, 1973.