Structure for indexing texts

Thierry Lecroq
Thierry.Lecroq@univ-rouen.fr

Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes.

International PhD School in Formal Languages and Applications

Tarragona, November 20th and 21st, 2006

Plan

1. Introduction
2. Suffix Tree
3. Suffix Automaton
4. Compact Suffix Automaton

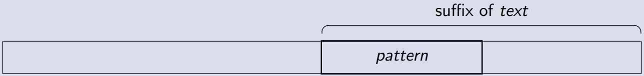
Plan

1. Introduction
2. Suffix Tree
3. Suffix Automaton
4. Compact Suffix Automaton

Indexes

- **Pattern matching** in static texts
- **Basic operations**
  - existence of patterns in the text
  - number of occurrences of patterns
  - list of positions of occurrences
- **Other applications**
  - finding repetitions in texts
  - finding regularities in texts
  - approximate matchings
  - ...

Implementation of indexes



Implementation

with efficient data structures

- **Suffix Trees**
  digital trees, PATRICIA tree (compact trees)
- **Suffix Automata or DAWG's**
  minimal automata, compact automata

with efficient algorithm

- **Suffix Arrays**
  binary search in the ordered list of suffixes

Efficient constructions

- **Position tree, suffix tree**
  [Weiner 1973], [McCreight, 1976], [Ukkonen, 1992] [Farach, 1997]
- **Suffix DAWG, suffix automaton, factor automaton**
  [Blumer *et al.*, 1983], [Crochemore, 1984]
- **Suffix array, PAT array**
  [Manber, Myers, 1993], [Gonnet, 1987]
  [Kärkkäinen, Sanders, 2003]
  [Kim et al., 2003], [Ko, Aluru, 2003]
- **Some other implementations of suffix trees**
  [Andersson, Nilsson, 1993], [Irving, 1995]
  [Kärkkäinen, 1995], [Munro et al., 1999]
- **For external memory** (*SB-trees*)
  [Ferragina, Grossi, 1995]
- **Compact suffix automaton**
  [Crochemore, Vérin, 1997], [Inenaga et al., 2001]

Suffixes

Text $y \in \Sigma^*$

- $\mathit{Suff}(y)$ = set of suffixes of y,
- $\mathrm{card}\,\mathit{Suff}(y) = |y| + 1$

Example

$\mathit{Suff}(\mathtt{ababbb})$

i	0	1	2	3	4	5		
$y[i]$	a	b	a	b	b	b		
							position	
	a	b	a	b	b	b	0	
		b	a	b	b	b	1	
			a	b	b	b	2	
				b	b	b	3	
					b	b	4	
						b	5	
						ε	6	(empty string)

Plan

Trie of suffixes

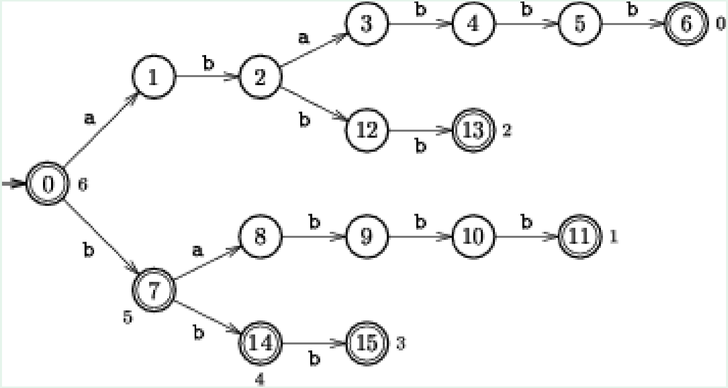
Text $y \in \Sigma^*$, *Suff*(y) set of suffixes of y

$\mathcal{T}(y)$ = digital tree which branches are labeled by suffixes of y
= tree-like deterministic automaton accepting *Suff*(y)

- **Nodes**
  identified with subwords of y
- **Terminal nodes**
  identified with suffixes of y, output = position of the suffix

Trie of suffixes

Suffix trie of ababbb



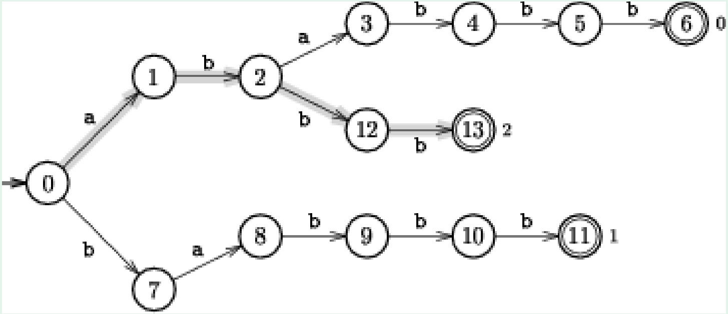
Forks

Insertion of $u = y[i \mathinner{.\,.} n-1]$ in the structure accepting longer suffixes

- **Head** of u: longest prefix $y[i \mathinner{.\,.} k-1]$ of u occurring before i
- **Tail** of u: rest $y[k \mathinner{.\,.} n-1]$ of suffix u
- $y =$ ababbb; head of abbb is ab; tail of abbb is bb
- **Fork**
  any node that has outdegree 2 at least,
  or that both has outdegre 1 and is terminal
- **Note**: the node associated with the head of u is a fork
  initial node is a fork iff y non empty

Forks

Example

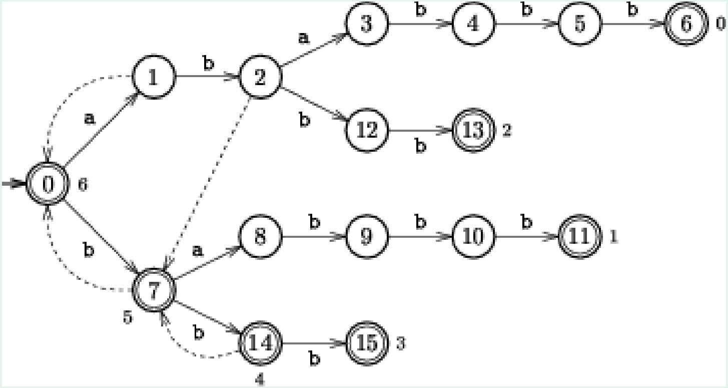


Suffix link

- **Function** s_y, *suffix link*
  if node p identified with subword av, $a \in \Sigma, v \in \Sigma^*$
  $s_y(p) = q$, node identified with v
- **Use**
  creates shortcuts used to accelerate heads computations
- **Useful for forks only**
  undefined on initial node
- **Note**: if p is a fork, so is $s_y(p)$

Suffix link

Example



Suffix Tree

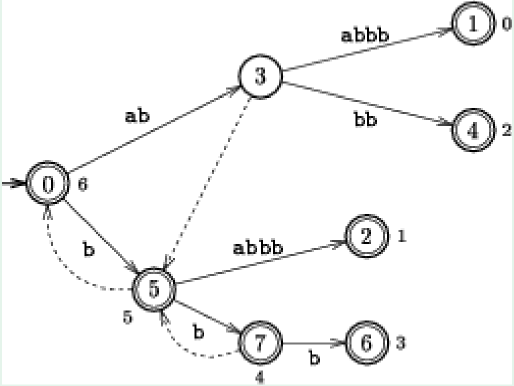
Text $y \in \Sigma^*$ of length n

$\mathcal{S}(y)$ suffix tree of y: compact trie accepting *Suff* (y)

- **Definition**
  tree obtained from the suffix trie of y by deleting all nodes having outdegree 1 that are not terminal
- **Edges labeled by subwords** of y instead of letters
- **Number of nodes**: no more than $2n$ (if $n > 0$)
  because all internal nodes have two children at least and there are at most n external nodes

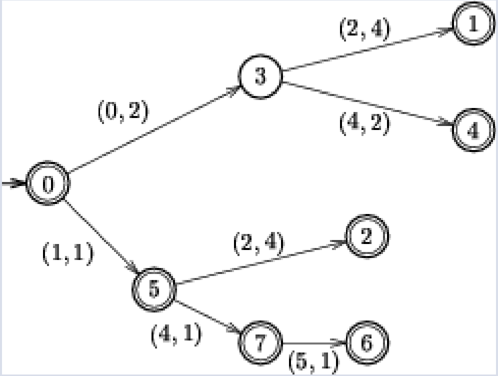
Suffix Tree

Example



Labels of edges

- **Labels represented by pairs** $(pos, Length)$



i	0	1	2	3	4	5
$y[i]$	a	b	a	b	b	b

- Requires to have y in main memory
- **Size of** $\mathcal{S}(y)$: $O(n)$

Scheme of suffix tree construction

```
SUFFIX-TREE(y)
T ← NEW-TREE()
for i ← 0 to n − 1 do
        find fork of head of y[i . . n − 1] using
            FAST-FIND from node s[r], and then SLOW-FIND
        k ← position of tail of y[i . . n − 1]
        if k < n then
                q ← NEW-STATE()
                Adj[fork] ← Adj[fork] ∪ {((k, n − k), q)}
                output[q] ← i
        else output[fork] ← i
return T
```

Adjacency-list representation of labeled arcs

Straight insertion

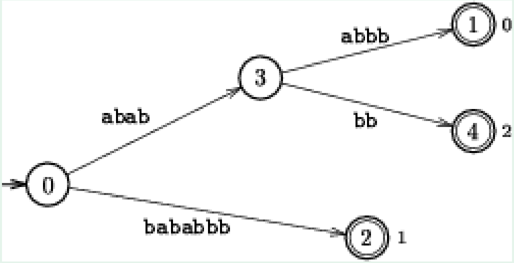
- Insertion of suffix ababbb is done by letter comparisons from the initial node (current node)

- It leads to create node 3 which suffix link is still undefined,
- and node 4 associated with suffix ababbb at position 2
- Head is abab, tail is bb

Straight insertion

Results of the insertion of ababbb

i	0	1	2	3	4	5	6	7
$y[i]$	a	b	a	b	a	b	b	b



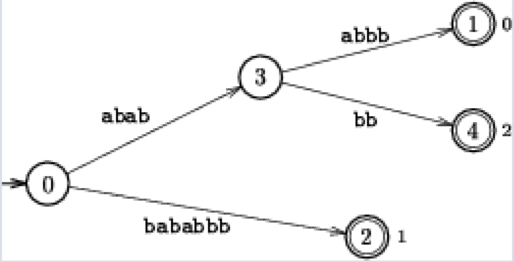
Slow find

$\textsc{Slow-Find}(p, k)$

```
while k < n and TARGET(p, y[k]) ≠ NIL do
    q ← TARGET(p, y[k])
    (j, ℓ) ← label(p, q)
    i ← j
    do  i ← i + 1
        k ← k + 1
    while i < j + ℓ and k < n and y[i] = y[k]
    if i < j + ℓ then
        Adj[p] ← Adj[p] \ {((j, ℓ), q)}
        r ← NEW-STATE()
        Adj[p] ← Adj[p] ∪ {((j, i − j), r)}
        Adj[r] ← Adj[r] ∪ {((j + i − j, ℓ − i + j), q)}
        return (r, k)
    p ← q
return (p, k)
```

New suffix link

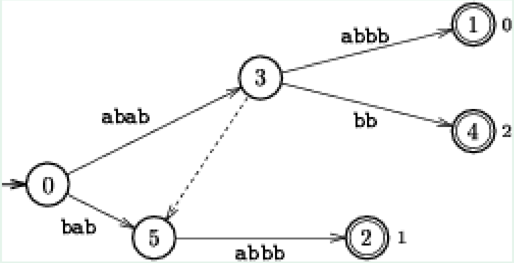
- Computing $s[3] = s_y(3)$ remains to find the node associated with bab



- Arc $(0,(1,7),2)$ is split into $(0,(1,3),5)$ and $(5,(4,4),2)$
- Execution in constant time (here)
- In general, iteration in time proportional to the number of nodes along the path (and not proportional to the length of the string)

New suffix link

i	0	1	2	3	4	5	6	7
$y[i]$	a	b	a	b	a	b	b	b



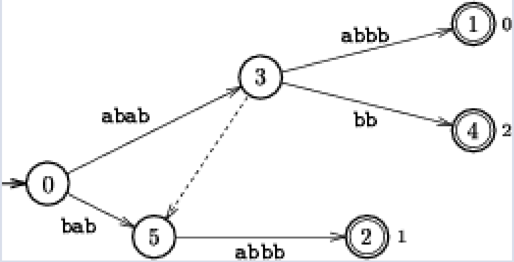
Fast find

$\text{Fast-Find}(r, j, k)$

$\triangleright$ computes $\text{Target}(r, y[j \mathinner{.\,.} k-1])$
if $j \geqslant k$ **then**
  return r
else $q \leftarrow \text{Target}(r, y[j])$
  $(j', \ell) \leftarrow \mathit{label}(r, q)$
  if $j + \ell \leqslant k$ **then**
    return $\text{Fast-Find}(q, j+\ell, k)$
  else $\mathit{Adj}[r] \leftarrow \mathit{Adj}[r] \setminus \{((j', \ell), q)\}$
    $p \leftarrow \text{New-state}()$
    $\mathit{Adj}[r] \leftarrow \mathit{Adj}[r] \cup \{((j, k-j), p)\}$
    $\mathit{Adj}[p] \leftarrow \mathit{Adj}[p] \cup \{((j'+k-j, \ell-k+j), q)\}$
    return p

Next insertion

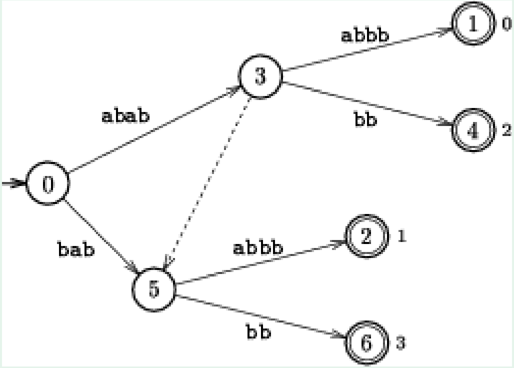
- End of insertion of suffix babbb



- Execution in constant time
- Head is bab, tail is bb

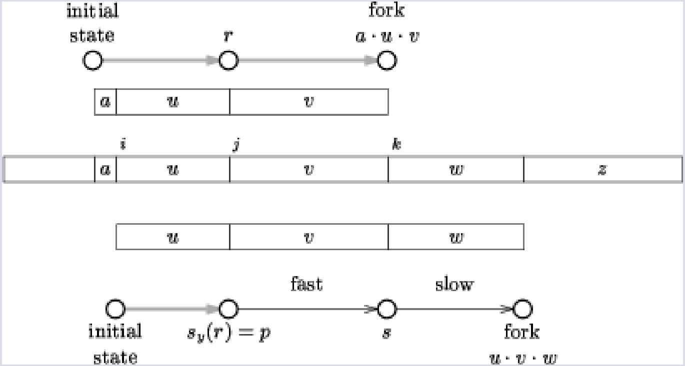
Next insertion

i	0	1	2	3	4	5	6	7
$y[i]$	a	b	a	b	a	b	b	b



Scheme for insertion

- Scheme for the insertion of suffix $y[i \,.\,.\, n-1] = u \cdot v \cdot w \cdot z$



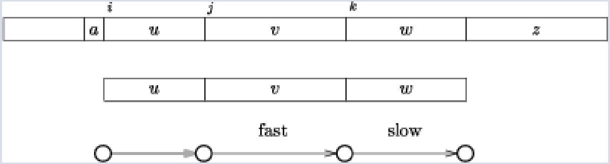
- It first computes $p = \textsc{Target}(s[r], v)$ with $\textsc{Fast-Find}$ (if necessary)
- then the fork of the current suffix with $\textsc{Slow-Find}$

Complete algorithm

```
SUFFIX-TREE(y)
T ← NEW-TREE()
s[initial[T]] ← initial[T]
(fork, k) ← (initial[T], 0)
for i ← 0 to n − 1 do
    k ← max{k, i}
    if s[fork] = NIL then
        r ← parent of fork
        (j, ℓ) ← label(r, fork)
        if r = initial[T] then
            ℓ ← ℓ − 1
        s[fork] ← FAST-FIND(s[r], k − ℓ, k)
    (fork, k) ← SLOW-FIND(s[fork], k)
    if k < n then
        q ← NEW-STATE()
        Adj[fork] ← Adj[fork] ∪ {((k, n − k), q)}
        output[q] ← i
    else output[fork] ← i
return T
```

Running time

- Scheme for insertion



- Main iteration increments i, which never decreases
- Iteration in $\textsc{Fast-Find}$ increments j, which never decreases
- Iteration in $\textsc{Slow-Find}$ increments k, which never decreases
- Basic operations run in constant time or in time $O(\log \operatorname{card} \Sigma)$

Theorem

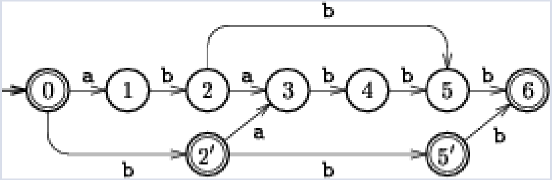
Execution of $\textsc{Suffix-tree}(y) = \mathcal{S}(y)$ *takes* $O(|y| \times \log \operatorname{card} \Sigma)$ *time in the comparison model.*

Plan

Suffix Automaton

Text $y \in \Sigma^*$ of length n
$\mathcal{A}(y)$ = minimal deterministic automaton accepting *Suff* (y)

- **Minimization** of the trie of suffixes



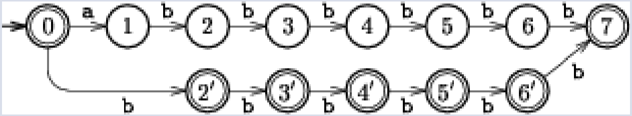
- States are classes of factors (subwords) of y
- **Size**:

$$n + 1 \leqslant \#states \leqslant 2n - 1$$

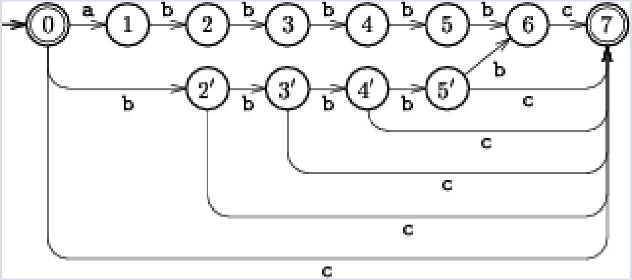
$$n \leqslant \#arcs \leqslant 3n - 4$$

Maximal size

- Maximal number of states

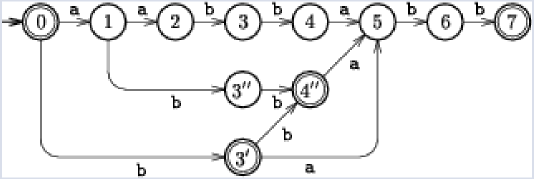


- Maximal number of arcs



Suffix link

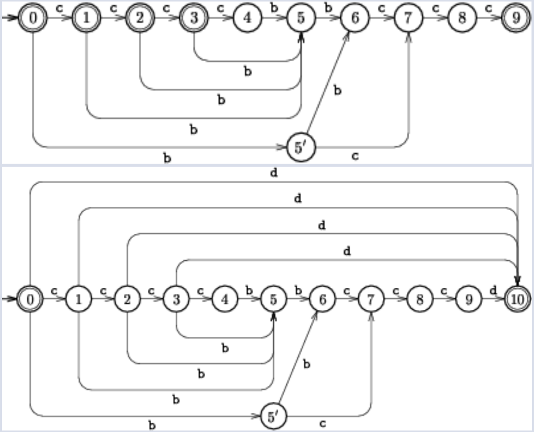
- **Function** f_y, *suffix link*
  let $p = \textsc{Target}(\textit{initial}[\mathcal{A}], v,)$, $v \in \Sigma^+$
  $f_y(p) = \textsc{Target}(\textit{initial}[\mathcal{A}], u,)$, where u is the longest suffix of v occurring in a different right context



- $s[1] = 0$, $s[2] = 1$, $s[3] = 3''$, $s[3''] = 3'$, $s[3'] = 0$, $s[4] = 4''$, $s[4''] = 3'$, $s[5] = 1$, $s[6] = 3''$, $s[7] = 4''$.
- **Suffix path**
  example for state 7: $\langle 7, 4'', 3', 0 \rangle$, sequence of terminal states
- **Use**
  same but more efficient than suffix link in suffix trees

One step (1)

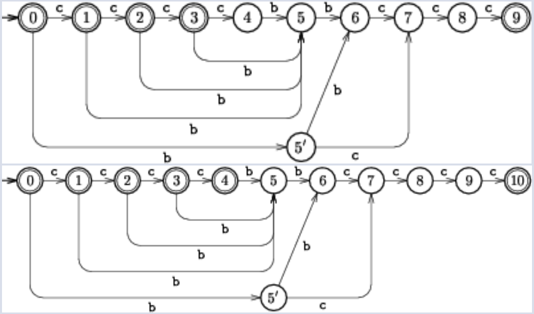
- **From** $\mathcal{A}(\text{ccccbbccc})$ **to** $\mathcal{A}(\text{ccccbbcccd})$



- New arcs from states of the suffix path $\langle 9, 3, 2, 1, 0 \rangle$.

One step (2)

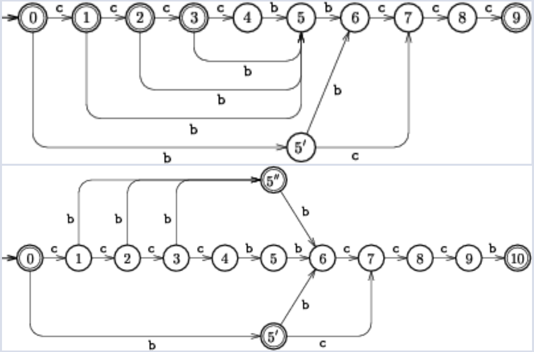
- **From** $\mathcal{A}(\texttt{ccccbbccc})$ **to** $\mathcal{A}(\texttt{ccccbbcccc})$



- Link $3 = s[9]$ and solid arc $(3, \mathtt{c}, 4)$ (not a shortcut) then, $s[10] = \textsc{Target}(3, \mathtt{c}) = 4$

One step (3)

- **From** $\mathcal{A}(\text{ccccbbccc})$ **to** $\mathcal{A}(\text{ccccbbcccb})$



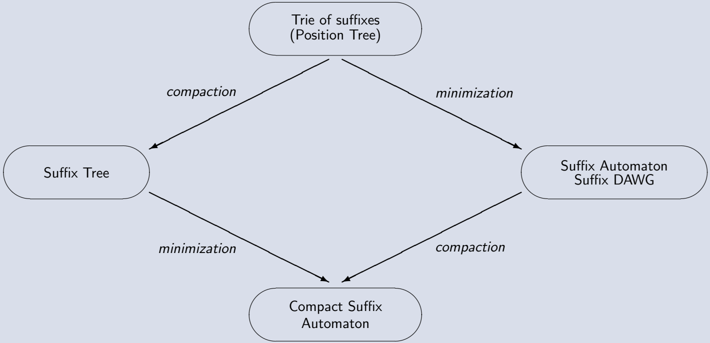
- Link $3 = s[9]$, non-solid arc $(3, \text{b}, 5)$, cccb suffix but ccccb not state 5 is cloned into $5'' = s[10] = s[5]$, $s[5''] = 5'$ arcs $(3, \text{b}, 5)$, $(2, \text{b}, 5)$ et $(1, \text{b}, 5)$ are redirected onto $5''$

Operations on indexes

Text y of length n

- **Index implemented by suffix tree or suffix automaton of y**
  memory space $O(n)$, construction time $O(n \times \log \operatorname{card} \Sigma)$
- **String matching**
  searching y for x of length m: time $O(m \times \log \operatorname{card} \Sigma)$
  number of occurrences of x in y: same complexity after $O(n)$ preprocessing
- **All occurrences**
  finding all occurrences of x in y: time $O(m \times \log \operatorname{card} \Sigma) + |output|)$
- **Repetitions**
  computing a longest subword of y occurring at least k times: time $O(n)$
- **Marker**
  computing a shortest subword of y occurring exactly once: time $O(n)$

Saving space



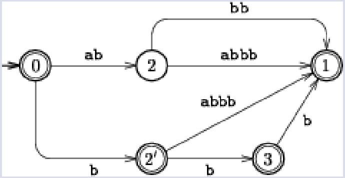
Plan

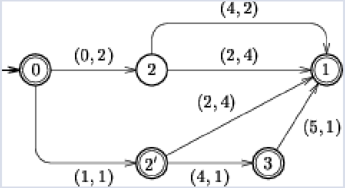
Compact Suffix Automaton

Text $y \in \Sigma^*$ of length n

$\mathcal{A}^{\mathsf{C}}(y)$ = compact minimal automaton accepting *Suff* (y)

- **Compaction** of $\mathcal{A}(y)$, or **minimization** of $\mathcal{S}(y)$





i	0	1	2	3	4	5
$y[i]$	a	b	a	b	b	b

- **Linear size**: $O(n)$

Direct construction of CSA

- Similar to both
  - Suffix Tree construction
  - Suffix Automaton construction
- Sequential addition of suffixes in the structure from the longest to the shortest
- Used features:
  - "slow-find" and "fast-find" procedures
  - suffix links
  - solid and non-solid arcs
  - state splitting
  - re-directions of arcs
- **Complexity**: $O(n \log \operatorname{card} \Sigma)$ time, $O(n)$ space
  50% **saved** on space of Suffix Automaton

References

A. Andersson and S. Nilsson.
Improved behavior of tries by adaptive branching.
Inf. Process. Lett., 46(6):295–300, 1993.

A. Apostolico et Z. Galil, editors. *Pattern matching algorithms*. Oxford University Press, 1997.

A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnel.
Linear size finite automata for the set of all subwords of a word: an outline of results.
Bull. Eur. Assoc. Theor. Comput. Sci., 21:12–20, 1983.

M. Crochemore.
Linear searching for a square in a word.
Bull. Eur. Assoc. Theor. Comput. Sci., 24:66–72, 1984.

M. Crochemore, C. Hancart and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 20017, to appear.

References

- M. Crochemore and T. Lecroq. Text Searching and Indexing. *Recent Advances in Formal Languages and Applications*, Series: Studies in Computational Intelligence, Volume 25, (2006), Z. Esik, C. Martin-Vide and V. Mitrana editors, pages 43-80, Springer Verlag.

- M. Crochemore et W. Rytter. *Jewels of Stringology*. World Scientific, 2002.

- M. Crochemore and R. Vérin.
  Direct construction of compact directed acyclic word graphs.
  In A. Apostolico and J. Hein, editors, *CPM*, LNCS 1264, pages 116–129, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.

- M. Farach.
  Optimal suffix tree construction with large alphabets.
  In *FOCS*, pages 137–143, Miami Beach, FL, 1997.

References

P. Ferragina and R. Grossi.
The string B-tree: A new data structure for string search in external memory and its applications.
J. Assoc. Comput. Mach., 46(2):236–280, 1999.

G. H. Gonnet.
The PAT text searching system.
Report, Department of Computer Science, University of Waterloo, Ontario, 1987.

D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi.
On-line construction of compact directed acyclic word graphs.
In A. Amir and G. M. Landau, editors, *CPM*, LNCS 2089, pages 169–180, Jerusalem, Israel, 2001. Springer-Verlag, Berlin.

References

R. W. Irving.
Suffix binary search trees.
Technical report, University of Glasgow, Computing Science Department, 1996.

J. Kärkkäinen.
Suffix cactus: a cross between suffix tree and suffix array.
In Z. Galil and E. Ukkonen, editors, *CPM*, LNCS 937, pages 191–204, Espoo, Finland, 1995. Springer-Verlag, Berlin.

J. Kärkkäinen and P. Sanders.
Simple linear work suffix array construction.
In *ICALP*, LNCS 2719, pages 943–955, Eindhoven, The Netherlands, 2003. Springer-Verlag, Berlin.

References

D. K. Kim, J. S. Sim, H. Park, and K. Park.
Linear-time construction of suffix arrays.
In R. A. Baeza-Yates, E. Chávez, and M. Crochemore, editors, *CPM*, LNCS 2676, pages 186–199, Morelia, Michocán, Mexico, 2003. Springer-Verlag, Berlin.

P. Ko and S. Aluru.
Space efficient linear time construction of suffix arrays.
In R. A. Baeza-Yates, E. Chávez, and M. Crochemore, editors, *CPM*, LNCS 2676, pages 200–210, Morelia, Michocán, Mexico, 2003. Springer-Verlag, Berlin.

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

E. M. McCreight.
A space-economical suffix tree construction algorithm.
J. Algorithms, 23(2):262–272, 1976.

References

J. I. Munro, V. Raman, and S. S. Rao.
Space efficient suffix trees.
J. Algorithms, 39(2):205–222, 2001.

G. A. Stephen. *String searching algorithms*. World Scientific Press, 1994.

E. Ukkonen.
Approximate string matching with q-grams and maximal matches.
Theor. Comput. Sci., 92(1):191–212, 1992.

P. Weiner.
Linear pattern matching algorithm.
In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.