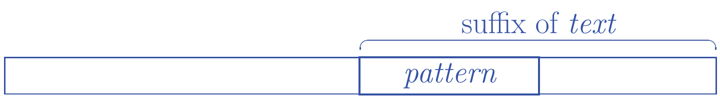
Suffix Automata

Maxime Crochemore

King's College London

`Maxime.Crochemore@kcl.ac.uk`
`http://www.dcs.kcl.ac.uk/staff/mac/`

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

Implementation with efficient algorithm

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

Suffixes

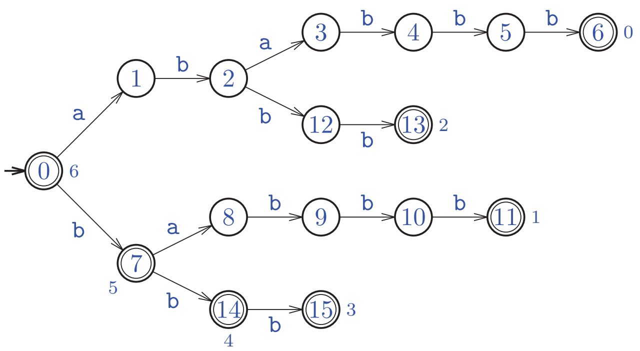
Text $y \in \Sigma^*$

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

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)

Trie of suffixes

- ⋆ $\mathcal{T}(y)$ = digital tree which branches are labeled by suffixes of y
  = tree-like deterministic automaton accepting $\mathit{Suff}(y)$
- ⋆ **Nodes**
  identified with factors (subwords) of y
- ⋆ **Terminal nodes**
  identified with suffixes of y, output = position of the suffix
- ⋆ **Suffix trie of ababbb**

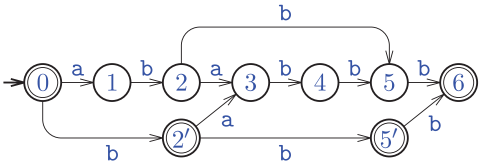


Suffix Automaton

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

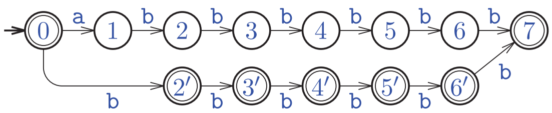
⋆ **Size**:

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

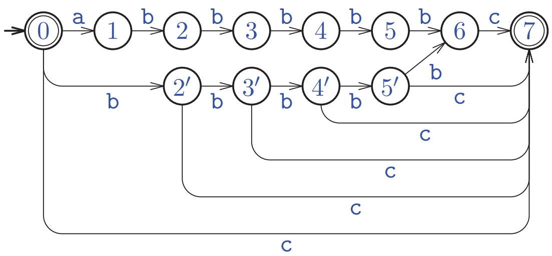
$$n \leq \#arcs \leq 3n - 4$$

Maximal size

⋆ **Maximal number of states**

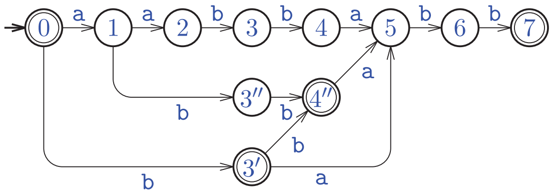


⋆ **Maximal number of arcs**



Suffix link

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



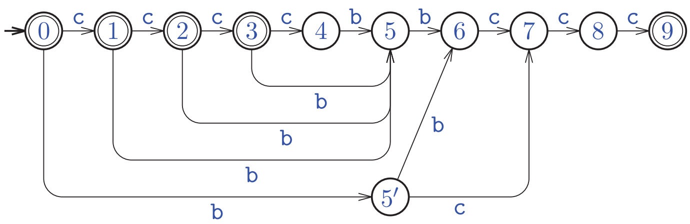
⋆ $f[1] = 0$, $f[2] = 1$, $f[3] = 3''$, $f[3''] = 3'$, $f[3'] = 0$,
$f[4] = 4''$, $f[4''] = 3'$, $f[5] = 1$, $f[6] = 3''$, $f[7] = 4''$.

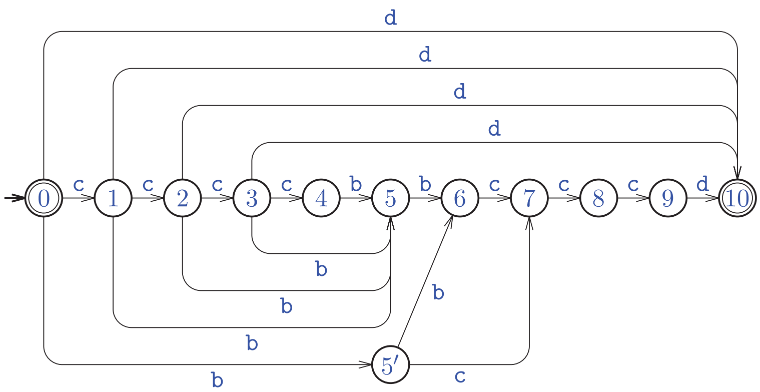
⋆ **Suffix path**
example for state 7: $\langle 7, 4'', 3', 0\rangle$, sequence of terminal states

⋆ **Use**
same but more efficient than suffix link in suffix trees

Construction—one step (1)

⋆ **From** $\mathcal{A}(\texttt{ccccbbccc})$ **to** $\mathcal{A}(\texttt{ccccbbcccd})$

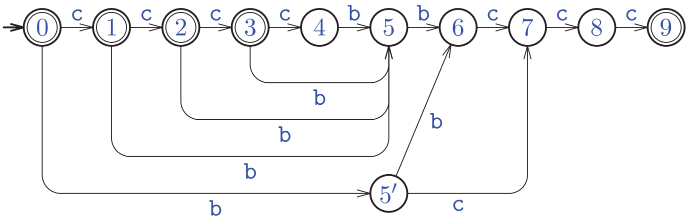


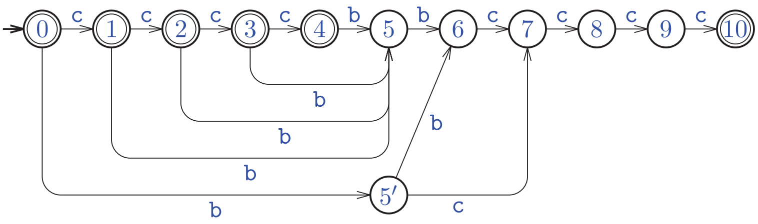


⋆ New arcs from states of the suffix path $\langle 9, 3, 2, 1, 0\rangle$.

Construction—one step (2)

⋆ **From** $\mathcal{A}$(`ccccbbccc`) **to** $\mathcal{A}$(`ccccbbcccc`)

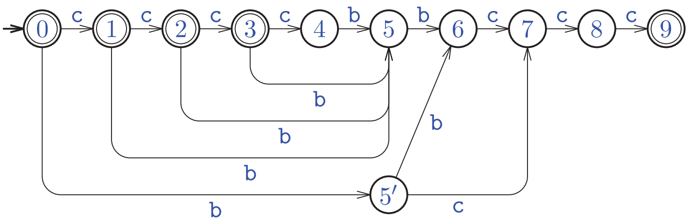


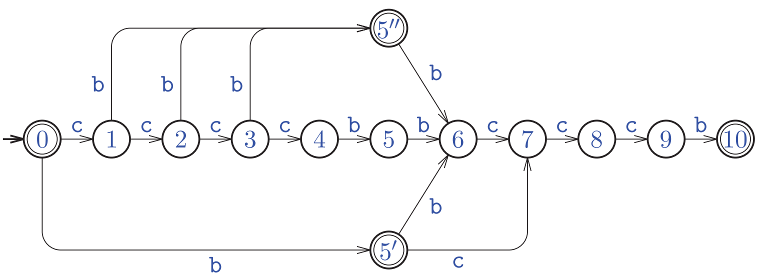


⋆ Link $3 = f[9]$ and solid arc $(3, \texttt{c}, 4)$ (not a shortcut)
then, $f[10] = \textsc{Target}(3, \texttt{c}) = 4$ that becomes a terminal state

Construction—one step (3)

⋆ **From** $\mathcal{A}$(`ccccbbccc`) **to** $\mathcal{A}$(`ccccbbcccb`)





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

Construction—main

```
SUFFIX-AUTO(y, n)
M ← NEW-AUTOMATON()
L[initial[M]] ← 0
F[initial[M]] ← NIL
last[M] ← initial[M]
for each letter a of y sequentially do
        ▷ Extension of M by the letter a
        EXTENSION(a)
p ← last[M]
do    terminal[p] ← TRUE
      p ← F[p]
while p ≠ NIL
return M
```


Construction—extension

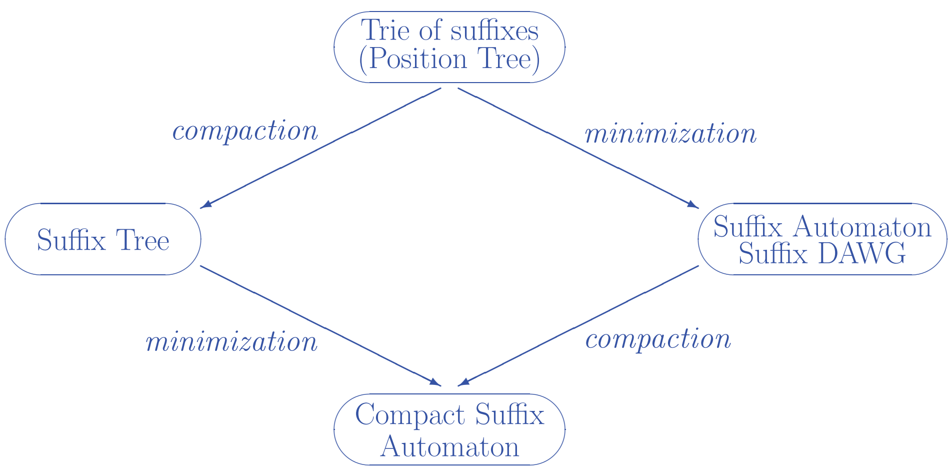
```
EXTENSION(a)
new_last ← NEW-STATE()
L[new_last] ← L[last[M]] + 1
p ← last[M]
do   Adj[p] ← Adj[p] ∪ {(a, new_last)}
     p ← F[p]
while p ≠ NIL and TARGET(p, a) = NIL
if p = NIL then
     F[new_last] ← initial[M]
else q ← TARGET(p, a)
     if (p, a, q) is solid, i.e. L[p] + 1 = L[q] then
          F[new_last] ← q
     else clone ← NEW-STATE()
          L[clone] ← L[p] + 1
          for each pair (b, q′) ∈ Succ[q] do
               Adj[clone] ← Adj[clone] ∪ {(b, q′)}
          F[new_last] ← clone
          F[clone] ← F[q]
          F[q] ← clone
          do   Adj[p] ← Adj[p] \ {(a, q)}
               Adj[p] ← Adj[p] ∪ {(a, clone)}
               p ← F[p]
          while p ≠ NIL and TARGET(p, a) = q
last[M] ← new_last
```

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 factor of y occurring at least k times: time $O(n)$
- **Marker**
  computing a shortest factor of y occurring exactly once: time $O(n)$

Saving space

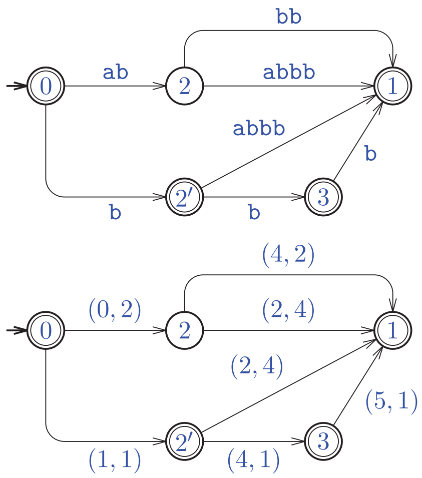


Compact Suffix Automaton

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

$\mathcal{A}^c(y)$ = compact minimal automaton accepting $\mathit{Suff}(y)$

- ⋆ **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**: $O(n \log \mathrm{card}\,\Sigma)$ time, $O(n)$ space
  50% **saved** on space of Suffix Automaton