Suffix Trees

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 $y \in \Sigma^*$

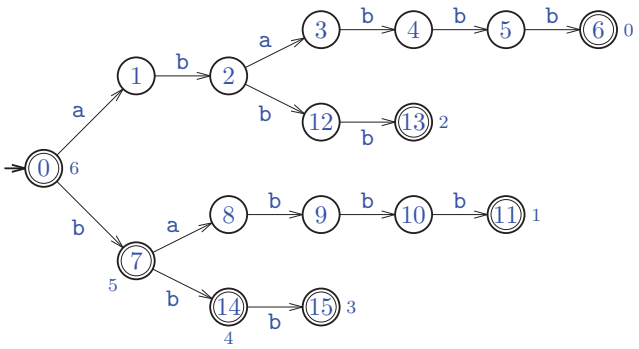
- $\mathit{Suff}(y)$ = set of suffixes of y,
- card $\mathit{Suff}(y) = |y| + 1$
- $\mathit{Suff}(\texttt{ababbb})$

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

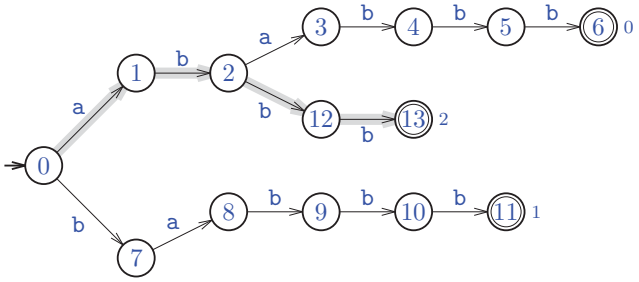
- $\mathcal{T}(y)$ = digital tree which branches are labeled by suffixes of y
  = tree-like deterministic automaton accepting $\mathit{Suff}(y)$
- **Nodes**
  identified with factors (subwords) of y
- **Terminal nodes**
  identified with suffixes of y, output = position of the suffix
- **Suffix trie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



Forks

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

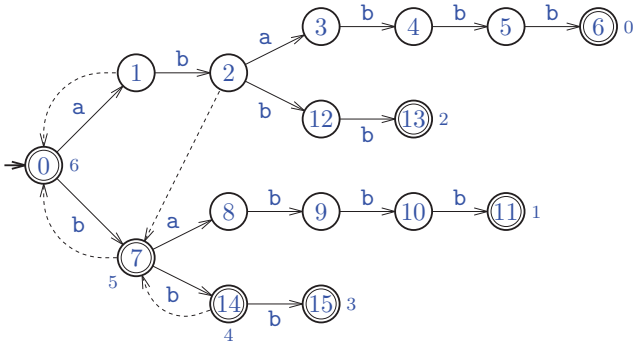
- **Head** of u: longest prefix $y[i \,..\, k-1]$ of u occurring before i
- **Tail** of u: rest $y[k \,..\, n-1]$ of suffix u
- $y =$ `ababbb`; head of `abbb` is `ab`; tail of `abbb` is `bb`



- **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

Suffix link

- **Function** s_y, *suffix link*
  if node p identified with factor av, $a \in \Sigma, v \in \Sigma^*$
  $s_y(p) = q$, node identified with v



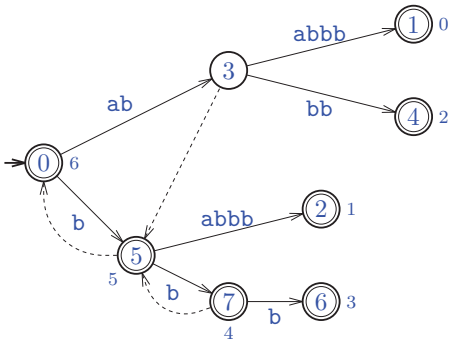
- **Use**
  creates shortcuts used to accelerate heads computations
- **Useful for forks only**
  undefined on initial node
- **Note**: if p is a fork, so is $s_y(p)$

Suffix Tree

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

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

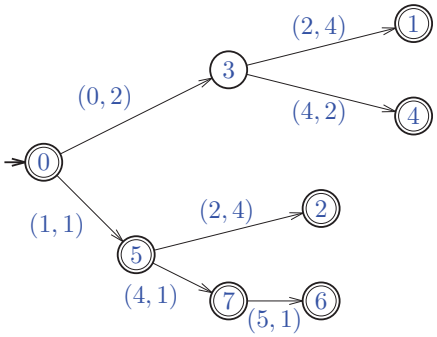
- ⋆ **Definition**
  tree obtained from the suffix trie of y by deleting all nodes having outdegree 1 that are not terminal
- ⋆ **Edges labeled by factors** of y instead of letters



- ⋆ **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

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** $\mathcal{S}(y)$: $O(n)$

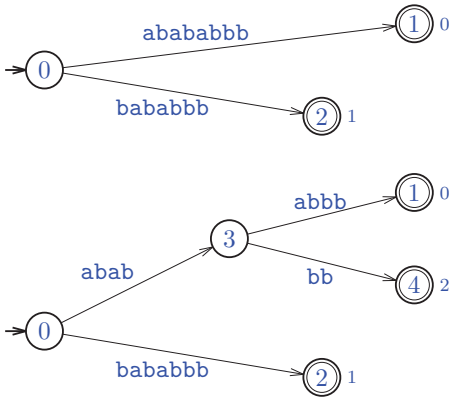
Scheme of suffix tree construction

```
SUFFIX-TREE(y)
 1  T ← NEW-TREE()
 2  for i ← 0 to n − 1 do
 3        find fork of head of y[i . . n − 1] using
            FAST-FIND from node s[parent] if needed
            and then SLOW-FIND
 4        k ← position of tail of y[i . . n − 1]
 5        if k < n then
 6              q ← NEW-STATE()
 7              Adj[fork] ← Adj[fork] ∪ {((k, n − k), q)}
 8              output[q] ← i
 9        else output[fork] ← i
10  output[initial] ← n
11  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)



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

⋆ It leads to create node 3 which suffix link is still undefined,

⋆ and node 4 associated with suffix `ababbb` at position 2

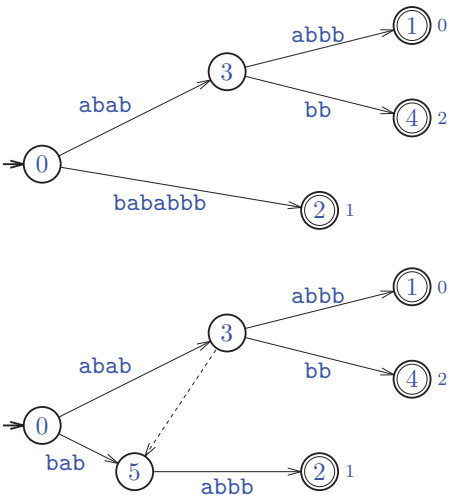
⋆ Head is `abab`, tail is `bb`

Slow find

```
SLOW-FIND(p, k)
 1  while k < n and TARGET(p, y[k]) ≠ NIL do
 2        q ← TARGET(p, y[k])
 3        (j, ℓ) ← label(p, q)
 4        i ← j
 5        do  i ← i + 1
 6            k ← k + 1
 7        while i < j + ℓ and k < n and y[i] = y[k]
 8        if i < j + ℓ then
 9            Adj[p] ← Adj[p] \ {((j, ℓ), q)}
10            r ← NEW-STATE()
11            Adj[p] ← Adj[p] ∪ {((j, i − j), r)}
12            Adj[r] ← Adj[r] ∪ {((j + i − j, ℓ − i + j), q)}
13            return (r, k)
14        p ← q
15  return (p, k)
```


New suffix link

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



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

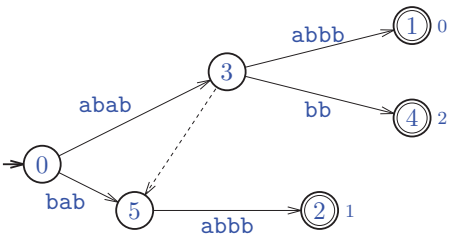
- Arc $(0, (1, 7), 2)$ is split into $(0, (1, 3), 5)$ and $(5, (4, 4), 2)$
- Execution in constant time (here)
- In general, iteration in time proportional to the number of nodes along the path (and not proportional to the length of the string)

Fast find

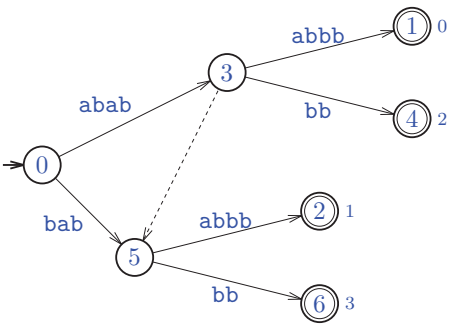
```
FAST-FIND(r, j, k)
 1  ▷ computes TARGET(r, y[j .. k − 1])
 2  if j ≥ k then
 3        return r
 4  else q ← TARGET(r, y[j])
 5        (j′, ℓ) ← label(r, q)
 6        if j + ℓ ≤ k then
 7              return FAST-FIND(q, j + ℓ, k)
 8        else Adj[r] ← Adj[r] \ {((j′, ℓ), q)}
 9              p ← NEW-STATE()
10              Adj[r] ← Adj[r] ∪ {((j, k − j), p)}
11              Adj[p] ← Adj[p] ∪ {((j′ + k − j, ℓ − k + j), q)}
12              return p
```


Next insertion

- ⋆ End of insertion of suffix babbb



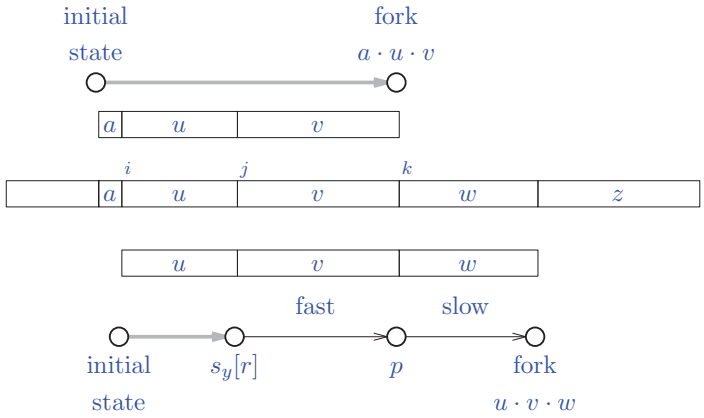
i	0	1	2	3	4	5	6	7
$y[i]$	a	b	a	b	a	b	b	b



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

Scheme for insertion

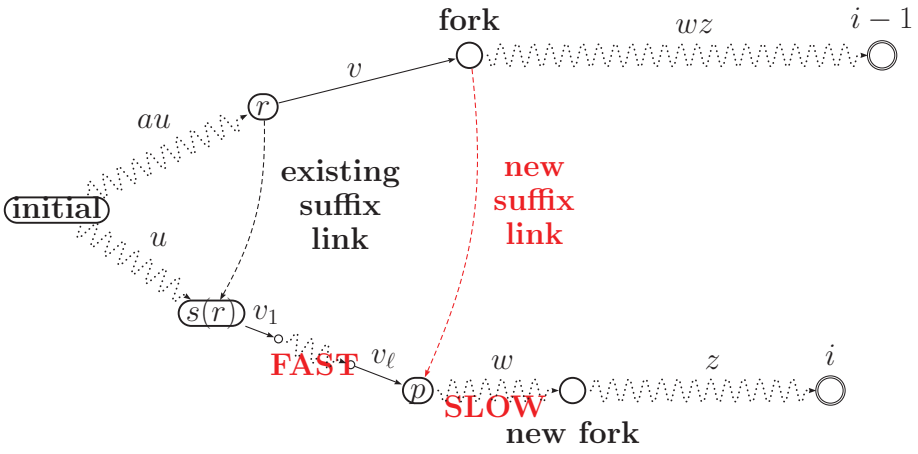
- Scheme for the insertion of suffix $y[i\,..\,n-1] = u \cdot v \cdot w \cdot z$



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

Scheme for insertion (continued)

- General scheme for inserting the next suffix in the data structure when the suffix target of the current fork is not defined

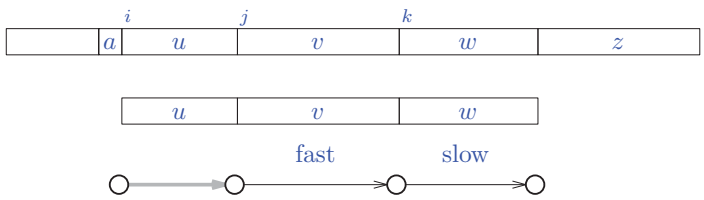


Complete algorithm

```
SUFFIX-TREE(y)
 1  T ← NEW-TREE()
 2  s[initial[T]] ← initial[T]
 3  (fork, k) ← (initial[T], 0)
 4  for i ← 0 to n − 1 do
 5       k ← max{k, i}
 6       if s[fork] = NIL then
 7            r ← parent of fork
 8            (j, ℓ) ← label(r, fork)
 9            if r = initial[T] then
10                 ℓ ← ℓ − 1
11            s[fork] ← FAST-FIND(s[r], k − ℓ, k)
12       (fork, k) ← SLOW-FIND(s[fork], k)
13       if k < n then
14            q ← NEW-STATE()
15            Adj[fork] ← Adj[fork] ∪ {((k, n − k), q)}
16            output[q] ← i
17       else output[fork] ← i
18  output[initial] ← n
19  return T
```


Running time

- ⋆ Scheme for insertion



- ⋆ Main iteration increments i, which never decreases
- ⋆ Iteration in FAST-FIND increments j, which never decreases
- ⋆ Iteration in SLOW-FIND increments k, which never decreases
- ⋆ Basic operations run in constant time or in time $O(\log \operatorname{card} \Sigma)$

Theorem 1 *Execution of* SUFFIX-TREE$(y) = \mathcal{S}(y)$ *takes* $O(|y| \times \log \operatorname{card} \Sigma)$ *time in the comparison model.*