Suffix Arrays

Maxime Crochemore

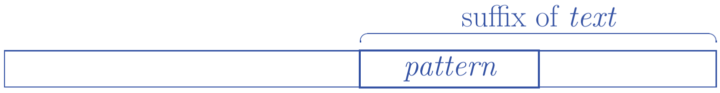
King's College London

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

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

Implementation 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, 1990], **[Gonnet, 1986]**
  [Kärkkäinen, Sanders, 2003]
  [Kim et al., 2003], **[Ko, Aluru, 2003]**, **[Nong et al., 2009]**
- ⋆ **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,
- ⋆ card $\mathit{Suff}(y) = |y| + 1$
- ⋆ $\mathit{Suff}($`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)

Suffix array

- ⋆ **Text** $y \in \Sigma^*$ of length n
- ⋆ **Permutation of suffixes positions**

  SUF: $\{0, 1, \ldots n-1\} \to \{0, 1, \ldots n-1\}$ such that

  $$y[SUF[0] \mathinner{.\,.} n-1] < y[SUF[1] \mathinner{.\,.} n-1] < \ldots < y[SUF[n-1] \mathinner{.\,.} n-1]$$
- ⋆ **LCP's of suffixes**

  $LCP[i] = |lcp(y[SUF[i-1] \mathinner{.\,.} n-1], y[SUF[i] \mathinner{.\,.} n-1])|$

i	$SUF[i]$	$LCP[i]$	
0	10	0	`a`
1	0	1	`a a b a a b a a b b a`
2	3	6	`a a b a a b b a`
3	6	3	`a a b b a`
4	1	1	`a b a a b a a b b a`
5	4	5	`a b a a b b a`
6	7	2	`a b b a`
7	9	0	`b a`
8	2	2	`b a a b a a b b a`
9	5	4	`b a a b b a`
10	8	1	`b b a`

Operations on indexes

Text y of length n

- **Index implemented by suffix array of y**
  memory space $O(n)$
- **String matching**
  searching y for x of length m: time $O(m + \log n)$
  number of occurrences of x in y: same complexity
- **All occurrences**
  finding all occurrences of x in y: time $O(m + \log n + |output|)$
- **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 $O(n)$

Computing a suffix array

- **Sorting suffixes**
  previous solution runs in time $O(n^2)$ because $||Suff(y)|| = O(n^2)$
  $O(n \log n)$-time algorithm based on the **doubling technique**
  $O(n)$-time algorithm on integer alphabet
  [Manber and Myers, 1993], **[Kärkkäinen, Sanders, 2003]**
- **Computing LCP's of suffixes**
  previous solution runs in time $O(n^2)$ because card $Suff(y) = O(n^2)$
  $O(n)$-time simple algorithm using SUF and its inverse r
  [Kasai et al., 2001]
- see also **[Kim et al., 2003]**, **[Ko, Aluru, 2003]**,
  [Nong et al., 2009]

Ranks of suffixes

Text y of length n, string $u \in \Sigma^*$, integer $k > 0$

⋆ **First**

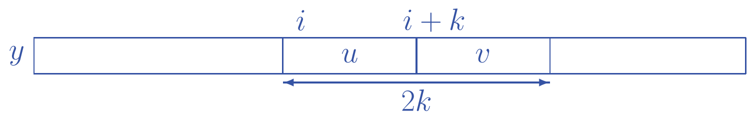
$$First_k(u) = \begin{cases} u & \text{if } |u| \leq k \\ u[0 \,.\,.\, k-1] & \text{otherwise} \end{cases}$$

⋆ **Rank**

i position of suffix $y[i \,.\,.\, n-1]$

$R_k[i]$ = rank of $First_k(y[i \,.\,.\, n-1])$ inside the ordered list of all $First_k(u)$, u non-empty suffix of y

Lemma 1 (doubling) *$R_{2k}[i]$ is the rank of $(R_k[i], R_k[i+k])$ in the ordered list of these pairs.*



Doubling technique

Input

i	0	1	2	3	4	5	6	7	8	9	10
$y[i]$	a	a	b	a	a	b	a	a	b	b	a

Output

$k=1$	$\{0,1,3,4,6,7,10\}$							$\{2,5,8,9\}$			
$k=2$	$\{10\}$	$\{0,3,6\}$			$\{1,4,7\}$			$\{2,5,9\}$			$\{8\}$
$k=4$	$\{10\}$	$\{0,3\}$		$\{6\}$	$\{1,4\}$		$\{7\}$	$\{9\}$	$\{2,5\}$		$\{8\}$
$k=8$	$\{10\}$	$\{0\}$	$\{3\}$	$\{6\}$	$\{1\}$	$\{4\}$	$\{7\}$	$\{9\}$	$\{2\}$	$\{5\}$	$\{8\}$

One step, doubling to get R_4 from R_2

i	0	1	2	3	4	5	6	7	8	9	10
$R_2[i]$	1	2	3	1	2	3	1	2	4	3	0
pair	(1,3)	(2,1)	(3,2)	(1,3)	(2,1)	(3,2)	(1,4)	(2,3)	(4,0)	(3,-1)	(0,-1)
$R_4[i]$	1	3	6	1	3	6	2	4	7	5	0

Sorting suffixes

SUFFIX-SORT(y, n)

for $r \leftarrow 0$ **to** $n-1$ **do**
    $SUF[r] \leftarrow r$
$k \leftarrow 1$
for $i \leftarrow 0$ **to** $n-1$ **do**
    $Rk[i] \leftarrow$ rank of $y[i]$ in the ordered list of letters in $alph(y)$
$SUF \leftarrow$ SORT$(SUF, n, Rk, 0)$
$i \leftarrow \text{card}\, alph(y)$
while $i < n$ **do**
    $SUF \leftarrow$ SORT(SUF, n, Rk, k)
    $SUF \leftarrow$ SORT$(SUF, n, Rk, 0)$
    $i \leftarrow 0$
    $R2k[SUF[0]] \leftarrow i$
    for $r \leftarrow 1$ **to** $n-1$ **do**
        if $Rk[SUF[r]] \neq Rk[SUF[r-1]]$ or $Rk[SUF[r]+k] \neq Rk[SUF[r-1]+k]$ **then**
            $i \leftarrow i+1$
        $R2k[SUF[r]] \leftarrow i$
    $(k, Rk) \leftarrow (2k, R2k)$
return SUF

Complexity of the sorting

- **Sort**
  implemented by bucket sort (radix sort)
  each call runs in $O(n)$ time
- **One step**
  runs in $O(n)$ time

Proposition 1 *Algorithm* SUFFIX-SORT *applied to text y of length n runs in $O(n \log n)$ time and $O(n)$ space.*

Sorting Suffixes on a Bounded Integer Alphabet

⋆ **Skew algorithm**

[1] bucket sort positions i according to $First_3(y[i\,..\,n-1])$, for $i=3q$ or $i=3q+1$ (include $i=n$ if n multiple of 3) $t[i]$: rank of i in the sorted list

[2] recursively sort the suffixes of the 2/3-shorter word
$t[0]t[3]\cdots t[3q]\cdots t[1]t[4]\cdots t[3q+1]\cdots$
$s[i]$: rank of suffix i in the sorted list ($i=3q$ or $i=3q+1$ position on y)

[3] sort suffixes $y[j\,..\,n-1]$ for j of the form $3q+2$ (i.e., bucket sort pairs $(y[j], s[j+1])$)

[4] merge lists obtained at steps 2 and 3
Note: comparing suffixes i (first list) and j (second list) remains to compare:
$(y[i], s[i+1])$ and $(y[j], s[j+1])$ if $i=3q$
$(y[i]y[i+1], s[i+2])$ and $(y[j]y[j+1], s[j+2])$ if $i=3q+1$

⋆ **Running time**: $T(n)=T(2n/3)+O(n)$ then $T(n)=O(n)$
[Kärkkäinen, Sanders, 2003]

Example 1

i	0	1	2	3	4	5	6	7	8	9	10
$y[i]$	a	a	b	a	a	b	a	a	b	b	a

Rank t	
0	a
1	a a b
2	a b a
3	a b b
4	b a

Rank s	i	$Suff$(11142230)
0	10	0
1	0	1 1 1 4 2 2 3 0
2	3	1 1 4 2 2 3 0
3	6	1 4 2 2 3 0
4	1	2 2 3 0
5	4	2 3 0
6	7	3 0
7	9	4 2 2 3 0

Rank	j	$(y[j], s[j+1])$
0	2	(b, 2)
1	5	(b, 3)
2	8	(b, 7)

i	0	1	2	3	4	5	6	7	8	9	10
$y[i]$	a	a	b	a	a	b	a	a	b	b	a
$r[i]$	1	4	8	2	5	9	3	6	10	7	0
$SUF[i]$	10	0	3	6	1	4	7	9	2	5	8

Example 2 — WRONG

i	0	1	2	3	4	5	6	7	8
$y[i]$	a	b	a	a	a	a	a	a	a

Rank t	
0	a a
1	a a a
2	a b a
3	b a a

Rank s	i	*Suff*(211310)
0	7	0
1	4	1 0
2	3	1 1 3 1 0
3	6	1 3 1 0
4	0	2 1 1 3 1 0
5	1	3 1 0

- ⋆ **WRONG: suffix at position 6 does not have the right rank**
- ⋆ **Solution**: when n is a multiple of 3, consider position n during steps 1 to 3 as if $y[n]$ is a symbol smaller than any other symbol

Example 2 — Correct

i	0	1	2	3	4	5	6	7	8	**9**
$y[i]$	a	b	a	a	a	a	a	a	a	

Rank t	
0	ε
1	a a
2	a a a
3	a b a
4	b a a

Rank s	i	*Suff*(3220421)
0	9	0 4 2 1
1	7	1
2	6	2 0 4 2 1
3	4	2 1
4	3	2 2 0 4 2 1
5	0	3 2 2 0 4 2 1
6	1	4 2 1

Rank	j	$(y[j], s[j+1])$
0	8	(a, 0)
1	5	(a, 2)
2	2	(a, 4)

i	0	1	2	3	4	5	6	7	8
$y[i]$	a	b	a	a	a	a	a	a	a
$r[i]$	7	8	6	5	4	3	2	1	0
$SUF[i]$	8	7	6	5	4	3	2	0	1

Computing LCP's of suffixes

⋆ **LCP's of suffixes**

$LCP[i] = |lcp(y[SUF[i-1]\,..\,n-1], y[SUF[i]\,..\,n-1])|$, for $0 \le i \le n$

i	0	1	2	3	4	5	6	7	8	9	10	11
$y[i]$	a	a	b	a	a	b	a	a	b	b	a	
$SUF[i]$	10	0	3	6	1	4	7	9	2	5	8	
$LCP[i]$	0	1	6	3	1	5	2	0	2	4	1	0

j	$r[j]$		j	$r[j]$	
0	1	a a b a a b a a b b a	1	4	a b a a b a a b b a
3	2	a a b a a b b a	4	5	a b a a b b a

Lemma 2 *Let* $j \in (1, 2, \ldots, n-1)$ *with* $r[j] > 0$.
Then $LCP[r[j-1]] - 1 \le LCP[r[j]]$.

Example

i	$SUF[i]$	$LCP[i]$	
0	10	0	a
1	0	1	a a b a a b a a b b a
2	3	6	a a b a a b b a
3	6	3	a a b b a
4	1	1	a b a a b a a b b a
5	4	5	a b a a b b a
6	7	2	a b b a
7	9	0	b a
8	2	2	b a a b a a b b a
9	5	4	b a a b b a
10	8	1	b b a

i $r[j]$	$SUF[i]$ j	$LCP[i]$ $LCP[r[j]]$	
1	0	1	a a b a a b a a b b a
4	1	1	a b a a b a a b b a
8	2	2	b a a b a a b b a
2	3	6	a a b a a b b a
5	4	5	a b a a b b a
9	5	4	b a a b b a
3	6	3	a a b b a
6	7	2	a b b a
10	8	1	b b a
7	9	0	b a
0	10	0	a

LCP algorithm

- **Rank** r: $r[j]$ = rank of suffix at position j $\quad (r = SUF^{-1})$
- **Permutation** SUF: $SUF[k]$ = position of suffix of rank k

```
Lcp(y, n, SUF, r)
ℓ ← 0
for j ← 0 to n − 1 do
    ℓ ← max{0, ℓ − 1}
    if r[j] > 0 then
        i ← SUF[r[j] − 1]
        while y[i + ℓ] = y[j + ℓ] do
            ℓ ← ℓ + 1
        LCP[r[j]] ← ℓ
LCP[0] ← 0
LCP[n] ← 0
return LCP
```

- **Running time**: $O(n)$

Complexity of LCP computation

- **Overall LCP computation**
  in time $O(n)$ for suffixes in lexicographic order, *i.e.* for the $n+1$ external nodes of the binary tree
  in time $O(n)$ for the other n nodes of the binary tree by Lemma 1

Proposition 2 *The computation of LCP's of pairs of suffixes used in the binary search can be done in time $O(n)$ with $O(n)$ memory space after suffixes are sorted.*