

Suffix Arrays

Thierry Lacroq

Thierry.Lacroq@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 Bounded Size Alphabet
- 3 LCP Table
- 4 Bibliography

Plan

- 1 Introduction
- 2 Bounded Size Alphabet
- 3 LCP Table
- 4 Bibliography

Suffix Array

- **Text** $y \in A^*$ of length n

- **Suffix Permutation**

$p : \{0, 1, \dots, n - 1\} \mapsto \{0, 1, \dots, n - 1\}$ such that

$$y[p[0]..n - 1] < y[p[1]..n - 1] < \dots < y[p[n - 1]..n - 1]$$

- **LCP (Longest Common Prefixes)**

$$LCP[i] = |lcp(y[p[i - 1]..n - 1], y[p[i]..n - 1])|$$

Suffix Array

Example for $y = \text{aabaabaabba}$

i	$p[i]$	$LCP[i]$											
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								

Plan

- 1 Introduction
- 2 Bounded Size Alphabet**
- 3 LCP Table
- 4 Bibliography

Algorithm

- 1 Two thirds of the positions i of y are sorted according $first_3(y[i..n-1])$: positions of the form $i = 3k$ or $i = 3k + 1$ with k integer. Let $t[i]$ be the rank of i in the sorted list.
- 2 Suffixes of the string $z = t[0]t[3] \cdots t[3k] \cdots t[1]t[4] \cdots t[3k+1] \cdots$ are sorted recursively. Let $s[i]$ be the rank of suffix at position i on y in the sorted list ($i = 3k$ or $i = 3k + 1$) derived from the sorted list of the suffixes of z .
- 3 The suffixes $y[j..n-1]$, for j of the form $3k + 2$, are sorted using table s .
- 4 The final step consists in merging lists obtained at steps 2 and 3.

Details of the different steps

Step 1

Can be executed in linear time by using a radix sort

Step 3

Since the order of the suffixes $y[j + 1..n - 1]$ is known by s , step 3 can be realized in linear time by using a radix sort on the pairs $(y[j], s[j + 1])$.

Details of the different steps

Step 4

Comparing the suffixes at positions i ($i = 3k$ or $i = 3k + 1$ for the first list) and j ($j = 3k + 2$ for the second list) amounts to compare pairs of the form $(y[i], s[i + 1])$ and $(y[j], s[j + 1])$ if $i = 3k$ or of the form $(y[i]y[i + 1], s[i + 2])$ and $(y[j]y[j + 1], s[j + 2])$ if $i = 3k + 1$.

The merging operation can thus be done in linear time.

Example

The string

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

The 2 position sets

$$P_{01} = \{0, 1, 3, 4, 6, 7, 9, 10\} \quad P_2 = \{2, 5, 8\}$$

Step 1

	$i \bmod 3 = 0$				$i \bmod 3 = 1$			
i	0	3	6	9	1	4	7	10
$first_3(y[i..n-1])$	aab	aab	aab	ba	aba	aba	abb	a
$t[i]$	1	1	1	4	2	2	3	0

Example

Step 2

$z = 11142230$ $L_{01} = (10, 0, 3, 6, 1, 4, 7, 9)$

i	0	3	6	9	1	4	7	10
$s[i]$	1	2	3	7	4	5	6	0

Example

Step 3

	$i \bmod 3 = 2$		
i	2	5	8
$(y[i], s[i + 1])$	(b, 2)	(b, 3)	(b, 7)

$$L_2 = (2, 5, 8)$$

Example

Step 4

i	$i \bmod 3 = 0 \text{ or } 1$								$i \bmod 3 = 2$		
$(y[i]y[i+1], s[i+1])$	10	0	3	6	1	4	7	9	2	5	8
$(y[i], s[i+1])$	(a, -1)			(a, 4)(a, 5)(a, 6)			(ab, 2)(ab, 3)(ab, 7)		(b, 0)	(ba, 5)(ba, 6)(bb, 0)	(b, 2)(b, 3)(b, 7)

The suffix array

i	0	1	2	3	4	5	6	7	8	9	10
$p[i]$	10	0	3	6	1	4	7	9	2	5	8

Pseudo-code

Skew-suffix-sort(y, n)

```
1  if  $n \leq 3$  then
2    return permutation of the sorted suffixes of  $y$ 
3  else  $P_{01} \leftarrow \{i \mid 0 \leq i < n \text{ and } (i \bmod 3 = 0 \text{ or } i \bmod 3 = 1)\}$ 
4    if  $n \bmod 3 = 0$  then
5       $P_{01} \leftarrow P_{01} \cup \{n\}$ 
6     $t \leftarrow$  table of ranks of positions  $i$  of  $P_{01}$  according to  $first_3(y[i..n-1])$ 
7     $z \leftarrow t[0]t[3] \cdots t[3k] \cdots t[1]t[4] \cdots t[3k+1] \cdots$ 
8     $q \leftarrow$  SKEW-SUFFIX-SORT( $z, \lfloor 2n/3 \rfloor + 1$ )
9     $L_{01} \leftarrow (3q[j]$  if  $0 \leq q[j] \leq \lfloor n/3 \rfloor + 1, 3q[j] + 1$  else
with  $j = 0, 1, \dots, |z| - 1$ )
10    $s \leftarrow$  table of ranks of positions in  $L_{01}$ 
11    $(s[n], s[n+1]) \leftarrow (-1, -1)$ 
12    $L_2 \leftarrow$  list of positions  $j = 3k + 2, 3k + 2 < n$ 
sorted according to  $(y[j], s[j+1])$ 
13    $L \leftarrow$  merge of  $L_{01}$  and  $L_2$  using COMP
14    $p \leftarrow$  permutation of positions on  $y$  corresponding to  $L$ 
```

Pseudo-code

Comp(i, j)

```
1 if  $i \bmod 3 = 0$  then  
2   if  $(y[i], s[i + 1]) < (y[j], s[j + 1])$  then  
3     return  $-1$   
4   else return  $1$   
5 else if  $(y[i..i + 1], s[i + 2]) < (y[j..j + 1], s[j + 2])$  then  
6   return  $-1$   
7   else return  $1$ 
```

Complexity

Proposition 2.1

The execution time of the algorithm SKEW-SUFFIX-SORT applied on a string of length n is $O(n)$.

Proof.

The recursivity of the algorithm (line 9) give the recurrence relation $T(n) = T(2n/3) + O(n)$ with $T(n) = O(1)$ for $n \leq 3$ since the other lines execute in constant time or in time $O(n)$. This recurrence has solution $T(n) = O(n)$. □

Lemma 2.2

Using the algorithm notation, let z_0 and z_1 be such that $z = z_0z_1$ with $z_0 = t[0]t[3] \cdots t[3k] \cdots$ and $z_1 = t[1]t[4] \cdots t[3k+1] \cdots$. Let i'_0 and i'_1 be 2 positions on z . Let

$$i_0 = \begin{cases} 3 \times i'_0 & \text{if } i'_0 < \lfloor n/3 \rfloor + 1, \\ 3 \times i'_0 + 1 & \text{else,} \end{cases}$$

and let

$$i_1 = \begin{cases} 3 \times i'_1 & \text{if } i'_1 < \lfloor n/3 \rfloor + 1, \\ 3 \times i'_1 + 1 & \text{else.} \end{cases}$$

If $z[i'_0..|z| - 1] < z[i'_1..|z| - 1]$ then $y[i_0..n - 1] < y[i_1..n - 1]$.

Correction

Proof.

Note that the letter $z[|z_0| - 1]$ is unique since it corresponds to the unique string of length 1 or 2 when $n \bmod 3 \neq 0$, and to the empty string otherwise because of line 6.

Let us assume that $z[i'_0..|z| - 1] < z[i'_1..|z| - 1]$ and let us consider the length ℓ of their longest common prefix. The unicity of $z[|z_0|]$ implies that this letter can occur only in one of the 2 strings $z[i'_0 + \ell]$ and $z[i'_1 + \ell]$ and only in the last position.

Thus each string is a factor either of z_0 or of z_1 (none of them overlap the frontier between z_0 and z_1 in z).

Thus they both correspond to factors of y and the inequality between $z[i'_0 + \ell]$ and $z[i'_1 + \ell]$ transposes to the corresponding factors of y and thus to suffixes $y[i_0..n - 1] < y[i_1..n - 1]$. \square

Plan

- 1 Introduction
- 2 Bounded Size Alphabet
- 3 LCP Table**
- 4 Bibliography

Lemma 3.1

Let i, j, i' be positions on y such that $j = p[i]$ and $j - 1 = p[i']$.
Then $LCP[i'] - 1 \leq LCP[i]$.

LCP Table

Proof.

Let u be the longest common prefix to $y[j - 1..n - 1]$ and its predecessor in the lexicographic $y[k..n - 1]$. By definition :
 $LCP[i'] = |u|$.

If u is the empty string the result holds since $LCP[i] \geq 0$.
Otherwise, u can be written cv with $c = y[j - 1]$ and $v \in A^*$. The string $y[j - 1..n - 1]$ has for prefix cvb for a letter b , and its predecessor has for prefix cva for a letter a such that $a < b$ except if it is equal to cv . □

Proof.

Thus, v is a common prefix to $y[j..n - 1]$ and $y[k + 1..n - 1]$.
Moreover, $y[k + 1..n - 1]$, that starts with va or is equal to v , is inferior to $y[j..n - 1]$ that starts with vb .

Thus $LCP[i]$ which is the maximal length of common prefixes to $y[j..n - 1]$ and its predecessor in the lexicographic order cannot be smaller than $|u|$.

We thus have $LCP[i] \geq |u| = LCP[i'] - 1$. □

Algorithm

Def-LCP(y, n, p)

```
1 for  $i \leftarrow 0$  to  $n - 1$  do
2    $Rank[p[i]] \leftarrow i$ 
3  $\ell \leftarrow 0$ 
4 for  $j \leftarrow 0$  to  $n - 1$  do
5    $\ell \leftarrow \max\{0, \ell - 1\}$ 
6    $i \leftarrow Rank[j]$ 
7   if  $i \neq 0$  then
8      $j' \leftarrow p[i - 1]$ 
9     while  $j + \ell < n$  and  $j' + \ell < n$  and
10       $y[j + \ell] = y[j' + \ell]$  do
11        $\ell \leftarrow \ell + 1$ 
12   else  $\ell \leftarrow 0$ 
13    $LCP[i] \leftarrow \ell$ 
14 return  $LCP$ 
```

Plan

- 1 Introduction
- 2 Bounded Size Alphabet
- 3 LCP Table
- 4 Bibliography**

Bibliography



J. Kärkkäinen and P. Sanders.

Simple linear work suffix array construction.

In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, number 2719 in *Lecture Notes in Computer Science*, pages 943–955, Eindhoven, The Netherlands, 2003. Springer-Verlag, Berlin.



T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park.

Linear-time longest-common-prefix computation in suffix arrays and its application.

In A. Amir and G. M. Landau, editors, *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, number 2089 in *Lecture Notes in Computer Science*, pages 169–180, Jerusalem, Israel, 2001. Springer-Verlag, Berlin.



Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park.

Linear-time construction of suffix arrays.

In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 186–199, Morelia, Michocán, Mexico, 2003. Springer-Verlag, Berlin.



Pang Ko and Srinivas Aluru.

Space efficient linear time construction of suffix arrays.

In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, 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.