

Table des suffixes

Thierry Lecroq

Thierry.Lecroq@univ-rouen.fr

Laboratoire d'Informatique, Traitement de l'Information, Systèmes.



Plan

- 1 Introduction
- 2 Alphabet de taille bornée
- 3 Table des LCP
- 4 Bibliographie

Plan

- 1 Introduction
- 2 Alphabet de taille bornée
- 3 Table des LCP
- 4 Bibliographie

Table des suffixes

- **Texte** $y \in A^*$ de longueur n

- **Permutation des suffixes**

$p : \{0, 1, \dots, n-1\} \mapsto \{0, 1, \dots, n-1\}$ telle que

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

- **LCP (Longest Common Prefixes)**

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

Table des suffixes

Exemple pour $y = \text{aabaabaabba}$

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 Alphabet de taille bornée**
- 3 Table des LCP
- 4 Bibliographie

Algorithme

- 1 Deux tiers des positions i de y sont triées selon $\text{prem}_3(y[i..n-1])$: les positions de la forme $i = 3k$ ou $i = 3k + 1$ avec k entier. Soit $t[i]$ le rang de i dans la liste triée.
- 2 Les suffixes du mot $z = t[0]t[3] \cdots t[3k] \cdots t[1]t[4] \cdots t[3k+1] \cdots$ sont triés récursivement. Soit $s[i]$ le rang du suffixe à la position i sur y dans la liste triée ($i = 3k$ or $i = 3k + 1$) dérivée de la liste des suffixes triés de z .
- 3 Les suffixes $y[j..n-1]$, pour j de la forme $3k + 2$, sont triés en utilisant la table s .
- 4 L'étape finale consiste à fusionner les listes obtenues aux étapes 2 et 3.

Détails des étapes

Étape 1

Peut être exécutée en temps linéaire en utilisant un radix sort

Étape 3

Puisque l'ordre des suffixes $y[j + 1..n - 1]$ est connue par s , l'étape 3 peut être réalisée en temps linéaire en utilisant un radix sort sur les couples $(y[j], s[j + 1])$.

Détails des étapes

Étape 4

Comparer les suffixes aux positions i ($i = 3k$ ou $i = 3k + 1$ pour la première liste) et j ($j = 3k + 2$ pour la seconde liste) revient à comparer des couples de la forme $(y[i], s[i + 1])$ et $(y[j], s[j + 1])$ si $i = 3k$ ou de la forme $(y[i]y[i + 1], s[i + 2])$ et $(y[j]y[j + 1], s[j + 2])$ si $i = 3k + 1$.

La fusion peut donc être réalisée en temps linéaire.

Exemple

Le mot

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

Les 2 ensembles de positions

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

Étape 1

	$i \bmod 3 = 0$				$i \bmod 3 = 1$			
i	0	3	6	9	1	4	7	10
$prem_3(y[i..n-1])$	aab	aab	aab	ba	aba	aba	abb	a
$t[i]$	1	1	1	4	2	2	3	0

Exemple

Étape 2

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

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

Exemple

Étape 3

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

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

Exemple

Étape 4

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

La table des suffixes

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

Tri-des-suffixes(y, n)

```
1  si  $n \leq 3$  alors
2    retourner permutation des suffixes triés de  $y$ 
3  sinon  $P_{01} \leftarrow \{i \mid 0 \leq i < n \text{ et } (i \bmod 3 = 0 \text{ ou } i \bmod 3 = 1)\}$ 
4    si  $n \bmod 3 = 0$  alors
5       $P_{01} \leftarrow P_{01} \cup \{n\}$ 
6       $t \leftarrow$  table des rangs des positions  $i$  de  $P_{01}$  selon  $prem_3(y[i..n-1])$ 
7       $z \leftarrow t[0]t[3] \cdots t[3k] \cdots t[1]t[4] \cdots t[3k+1] \cdots$ 
8       $q \leftarrow \text{TRI-DES-SUFFIXES}(z, \lfloor 2n/3 \rfloor + 1)$ 
9       $L_{01} \leftarrow (3q[j] \text{ si } 0 \leq q[j] \leq \lfloor n/3 \rfloor + 1, 3q[j] + 1 \text{ sinon}$ 
avec  $j = 0, 1, \dots, |z| - 1$ )
10      $s \leftarrow$  table des rangs des positions de  $L_{01}$ 
11      $(s[n], s[n+1]) \leftarrow (-1, -1)$ 
12      $L_2 \leftarrow$  liste des positions  $j = 3k + 2, 3k + 2 < n$ 
triées selon  $(y[j], s[j+1])$ 
13      $L \leftarrow$  fusion de  $L_{01}$  et  $L_2$  en utilisant COMP
14      $p \leftarrow$  permutation des positions sur  $y$  correspondant à  $L$ 
```

Pseudo-code

Comp(i, j)

```
1  si  $i \bmod 3 = 0$  alors  
2    si  $(y[i], s[i + 1]) < (y[j], s[j + 1])$  alors  
3      retourner  $-1$   
4    sinon retourner  $1$   
5  sinon si  $(y[i..i + 1], s[i + 2]) < (y[j..j + 1], s[j + 2])$  alors  
6    retourner  $-1$   
7    sinon retourner  $1$ 
```

Proposition 2.1

Le temps d'exécution de l'algorithme TRI-DES-SUFFIXES appliqué à un mot de longueur n est $O(n)$.

Démonstration.

La récursivité de l'algorithme (ligne 9) donne la relation de récurrence $T(n) = T(2n/3) + O(n)$ avec $T(n) = O(1)$ pour $n \leq 3$ puisque les autres lignes s'exécutent en temps constant ou en temps $O(n)$. Cette récurrence a pour solution $T(n) = O(n)$. \square

Lemma 2.2

En utilisant les notations de l'algorithme, soient z_0 et z_1 tels que $z = z_0 z_1$ avec $z_0 = t[0]t[3] \cdots t[3k] \cdots$ et $z_1 = t[1]t[4] \cdots t[3k+1] \cdots$. Soient i'_0 et i'_1 2 positions sur z . Soit

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

et soit

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

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

Démonstration.

Remarquons que la lettre $z[|z_0| - 1]$ est unique puisqu'elle correspond à l'unique mot de longueur 1 ou 2 quand $n \bmod 3 \neq 0$, et au mot vide sinon à cause de la ligne 6.

On suppose que $z[i'_0..|z| - 1] < z[i'_1..|z| - 1]$ et on considère la longueur ℓ de leur plus long préfixe commun. L'unicité de $z[|z_0|]$ implique que cette lettre ne peut apparaître que dans l'un de ces 2 mots $z[i'_0 + \ell]$ et $z[i'_1 + \ell]$ et uniquement en dernière position. Donc chaque mot est un facteur soit de z_0 ou de z_1 (aucun ne chevauche la frontière entre z_0 et z_1 dans z).

Donc ils correspondent tous les 2 à des facteurs de y et l'inégalité entre $z[i'_0 + \ell]$ et $z[i'_1 + \ell]$ se transpose aux facteurs correspondant de y et donc aux suffixes $y[i_0..n - 1] < y[i_1..n - 1]$. \square

Plan

- 1 Introduction
- 2 Alphabet de taille bornée
- 3 Table des LCP**
- 4 Bibliographie

Lemma 3.1

Soient i, j, i' des positions sur y pour lesquels $j = p[i]$ et $j - 1 = p[i']$. Alors $LCP[i'] - 1 \leq LCP[i]$.

Démonstration.

Soit u le plus long préfixe commun à $y[j - 1..n - 1]$ et à son prédécesseur dans l'ordre lexicographique $y[k..n - 1]$. Par définition : $LCP[i'] = |u|$.

Si u est le mot vide le résultat est vrai car $LCP[i] \geq 0$. Sinon, u peut s'écrire cv avec $c = y[j - 1]$ et $v \in A^*$. Le mot $y[j - 1..n - 1]$ admet alors le préfixe cvb pour une lettre b , et son prédécesseur admet pour préfixe cva pour une lettre a telle que $a < b$ sauf s'il est égal à cv . □

Démonstration.

Donc, v est un préfixe commun à $y[j..n - 1]$ et $y[k + 1..n - 1]$. De plus, $y[k + 1..n - 1]$, qui commence par va ou est égal à v , est inférieur à $y[j..n - 1]$ qui commence par vb .

Donc $LCP[i]$ qui est la longueur maximale des préfixes communs à $y[j..n - 1]$ et à son prédécesseur dans l'ordre lexicographique ne peut pas être plus petit que $|u|$.

Nous avons donc $LCP[i] \geq |u| = LCP[i'] - 1$. □

Algorithm

Def-LCP(y, n, p)

```
1  pour  $i \leftarrow 0$  à  $n - 1$  faire
2      Rank[p[i]]  $\leftarrow i$ 
3   $\ell \leftarrow 0$ 
4  pour  $j \leftarrow 0$  à  $n - 1$  faire
5       $\ell \leftarrow \max\{0, \ell - 1\}$ 
6       $i \leftarrow \text{Rank}[j]$ 
7      si  $i \neq 0$  alors
8           $j' \leftarrow p[i - 1]$ 
9          tantque  $j + \ell < n$  et  $j' + \ell < n$  et
                 $y[j + \ell] = y[j' + \ell]$  faire
10              $\ell \leftarrow \ell + 1$ 
11         sinon  $\ell \leftarrow 0$ 
12         LCP[i]  $\leftarrow \ell$ 
13 LCP[n]  $\leftarrow 0$ 
14 retourner LCP
```

Plan

- 1 Introduction
- 2 Alphabet de taille bornée
- 3 Table des LCP
- 4 **Bibliographie**

Bibliographie



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.