

# Suffix Array

Ulises Méndez Martínez

Ulises Tirado Zatarain

# Suffix Array

- **Definición:** Un **suffix array** es un arreglo que contiene los índices de los sufijos ordenados alfabéticamente.
- **Ejemplo:** sea  $SA$  un **suffix array** y dos enteros  $i, j$ . Siempre que  $i < j$  tendremos que  $S[SA[i]..N] < S[SA[j]..N]$ .

**Ej:** Sea  $S = \text{"ababaac"}$

Sus sufijos:

0:ababaac

1:babaac

2:abaac

3:baac

4:aac

5:ac

6:c

Después de ordenar:

4:aac

2:abaac

0:ababaac

5:ac

3:baac

1:babaac

6:c

El suffix array es:

$sa[0] = 4$

$sa[1] = 2$

$sa[2] = 0$

$sa[3] = 5$

$sa[4] = 3$

$sa[5] = 1$

$sa[6] = 6$

## Solución Naive (Done is better than none)

- En principio, una cadena de longitud  $n$  tiene  $n + 1$  sufijos (contando la cadena completa completa y el sufijo vacío), y la suma de la cantidad de caracteres de todos ellos es  $O(n^2)$ , por lo que tan solo leer los sufijos tardaría al menos  $O(n^2)$ .
- Si tomamos todos los sufijos y los ordenamos lexicográficamente con la función sort, obtendremos un algoritmo  $O(n^2 * \log(n))$ .

Source  $O(N^2 \cdot \log N)$ : <http://ideone.com/aiTl5x>

```
#include <bits/stdc++.h>


using namespace std;

char *str = "ababaac";

bool cmp(int a, int b) {
    return strcmp(str+a, str+b) < 1;
}

void build_suffix( ) {
    int n = strlen(str);
    int sa[n];
    for(int i=0; i<n; i++) sa[i] = i;
    sort(sa,sa+n, cmp);
    for(int i=0; i<n; i++){
        printf("sa[%d] = %d\n",i,sa[i]);
    }
}

int main() {
    build_suffix();
    return 0;
}
```

 stdin

Standard input is empty

 stdout

sa[0] = 4

sa[1] = 2

sa[2] = 0

sa[3] = 5

sa[4] = 3

sa[5] = 1

sa[6] = 6

Lets move to  $O(N \cdot \log^2(N))$

<https://discuss.codechef.com/questions/21385/a-tutorial-on-suffix-arrays>

<http://www.geeksforgeeks.org/suffix-array-set-2-a-nlognlogn-algorithm/>

<https://web.stanford.edu/class/cs97si/suffix-array.pdf>

<https://www.hackerrank.com/challenges/string-similarity/topics/suffix-array>

<http://codeforces.com/blog/entry/4025?#comment-81522>

# En resumen....

- Vamos a apoyarnos en el hecho de que los sufijos a ordenar no son cadenas al azar, todas pertenecen en realidad a una misma cadena.
- Para mejorar el performance podemos comparar los sufijos, inicialmente por la primera letra, a continuación por las 2 primeras letras, por las 4, y así sucesivamente por las  $2^k$  letras.
- Si ya tenemos los sufijos ordenados por sus  $2^i$  primeras letras, podemos ordenarlos por sus  $2^{(i+1)}$  utilizando algún algoritmo como merge sort en  $O(N \log N)$ , esto se logra porque la comparación entre cada sufijo es hecha en  $O(1)$ . ... (Wait, what?)

# Banana: Paso a pasito

Index	Suffix	Rank
0	banana	1
1	anana	0
2	nana	13
3	ana	0
4	na	13
5	a	0



Index	Suffix	Rank	Next Rank
0	banana	1	0
1	anana	0	13
2	nana	13	0
3	ana	0	13
4	na	13	0
5	a	0	-1

Index	Suffix	Rank	Next Rank
5	a	0	-1
1	anana	0	13
3	ana	0	13
0	banana	1	0
2	nana	13	0
4	na	13	0

# Asignación de posiciones


Index	Suffix	Rank
5	a	0 [Assign 0 to first]
1	anana	1 (0, 13) is different from previous
3	ana	1 (0, 13) is same as previous
0	banana	2 (1, 0) is different from previous
2	nana	3 (13, 0) is different from previous
4	na	3 (13, 0) is same as previous

Index	Suffix	Rank	Next	Rank
5	a	0		-1
1	anana	1		1
3	ana	1		0
0	banana	2		3
2	nana	3		3
4	na	3		-1

Index	Suffix	Rank	Next	Rank
5	a	0		-1
3	ana	1		0
1	anana	1		1
0	banana	2		3
4	na	3		-1
2	nana	3		3

$O(N \cdot \log^2(N))$ : <http://ideone.com/fujyna>

```
20. void build_suffix( ) {
21.     int N = strlen(str), prev = 0, now = 1;
22.
23.     int P[2][N];
24.     int L[N];
25.
26.     for(int i=0; i<N; ++i) {
27.         P[0][i] = str[i] - 'a';
28.     }
29.
30.     for(int cnt=1; cnt<N; cnt<=1)
31.     {
32.         for(int i=0; i < N; ++i)
33.         {
34.             L[i].F.F = P[prev][i];
35.             L[i].F.S = (i + cnt < N) ? P[prev][i+cnt] : -1;
36.             L[i].S = i;
37.         }
38.         sort(L, L+N, cmp);
39.         for(int i=0; i < N; ++i) {
40.             P[now][L[i].S] = (i>0 && L[i].F==L[i-1].F) ? P[now][L[i-1].S] : i;
41.         }
42.         swap(now, prev);
43.     }
44.     // Print solution
45.     for(int i=0; i<N; i++){
46.         printf("sa[%d] = %d\n",i,L[i].S);
47.     }
48. }
```

 stdin

Standard input is empty

 stdout

sa[0] = 4

sa[1] = 2

sa[2] = 0

sa[3] = 5

sa[4] = 3

sa[5] = 1

sa[6] = 6

# Ahora $O(N \log N)$

<https://apps.topcoder.com/forums/?module=RevisionHistory&messageID=1171511>

<http://codeforces.com/blog/entry/18480?#comment-234539>

Tarea para el espectador, puntos claves:

- Ordenamiento en  $O(N)$  usando bucket sort.
- Hash?

# Referencias

\* Argentina Training Camp 2012, 2014. <https://sites.google.com/site/trainingcampargentina2014/>

\* Question on @Quora: What are some of the good sources to understand suffix a <http://qr.ae/1Uy2FT>



# Longest Common Prefix Array

- Is an auxiliary data structure to the suffix array.
- Stores the length of the longest common prefixes between pairs of consecutive suffixes in the sorted suffix array.
- Examples:
  - LCP of a and aabba is 1.
  - LCP of abaabba and abba is 2.

# Example

Consider the string  $S = \text{banana\$}$ :

<b>i</b>	1	2	3	4	5	6	7
<b>S[i]</b>	b	a	n	a	n	a	\$

and its corresponding suffix array  $A$  :

<b>i</b>	1	2	3	4	5	6	7
<b>A[i]</b>	7	6	4	2	1	5	3

Complete suffix array with suffixes itself:

<b>i</b>	1	2	3	4	5	6	7
<b>A[i]</b>	7	6	4	2	1	5	3
<b>1</b>	\$	a	a	a	b	n	n
<b>2</b>		\$	n	n	a	a	a
<b>3</b>			a	a	n	\$	n
<b>4</b>			\$	n	a		a
<b>5</b>				a	n		\$
<b>6</b>				\$	a		
<b>7</b>					\$		

Then the LCP array  $H$  is constructed by comparing lexicographically consecutive suffixes to determine their longest common prefix:

<b>i</b>	1	2	3	4	5	6	7
<b>H[i]</b>	$\perp$	0	1	3	0	0	2

# Algoritmo eficiente de construcción

- Kasai et al. (2001) presentó el primer algoritmo  $O(N)$  que construye el arreglo LCP dado el texto dado y su suffix array.
- Miremos un par de sufijos continuos en el suffix array y sean sus índices  $i1$  y  $i1+1$ . Si su  $lcp > 0$ , entonces si nosotros borramos la primera letra de ambos, podemos ver fácilmente que las cadenas resultantes tendrán el mismo orden relativo. También podemos notar que el lcp de estas nuevas cadenas será exactamente  $lcp-1$ .
- Ahora cambiemos al sufijo obtenido de  $i$  una vez que borramos la primer letra, sea su índice  $i2$ , ahora veamos el lcp de los sufijos  $i2$  y  $i2+1$ . Podemos ver que este lcp será al menos el ya mencionado  $lcp-1$ .
- $lcp(i,j)=\min(lcp_i, lcp_{i+1}, \dots, lcp_{j-1})$

# Implementación: $O(N)$ : <http://ideone.com/Bgxwxr>

```
vector<int> kasai(string s, vector<int> sa)
{
    int n=s.size(),k=0;
    vector<int> lcp(n,0);
    vector<int> rank(n,0);

    for(int i=0; i<n; i++) rank[sa[i]]=i;

    for(int i=0; i<n; i++, k?k--:0)
    {
        if(rank[i]==n-1) {k=0; continue;}
        int j=sa[rank[i]+1];
        while(i+k<n && j+k<n && s[i+k]==s[j+k]) k++;
        lcp[rank[i]]=k;
    }
    return lcp;
}
```

 stdin

Standard input is empty

 stdout

lcp[0] 1

lcp[1] 3

lcp[2] 1

lcp[3] 0

lcp[4] 2

lcp[5] 0

lcp[6] 0

# Referencias

<https://www.hackerrank.com/challenges/string-similarity/topics/lcp-array>

<http://codeforces.com/blog/entry/12796#comment-175287>

[https://en.wikipedia.org/wiki/LCP\\_array](https://en.wikipedia.org/wiki/LCP_array)