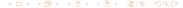
Suffix Array and Longest Common Prefix

Ulises ² (Tirado Zatarain | Mendez Martinez) (ulises.tirado@cimat.mx,ulisesmdzmtz@gmail.com)

²Algorists Group

July, 2016



Outline

- Introduction
 - Definitions

Solving the problem

Suffix Array

Definition (Suffix Array)

A suffix array is an array that contains the indexes of all lexicographically sorted string suffixes. Formally, let α be the suffix array of the string $S = s_1 s_2 \cdots s_n$ and denoting the substring of S ranginf from i-th to j-th character by S[i..j], then for all two integers such that i < j, we have that $S[a_i..n] < S[a_j..n]$.

Lets see an example

Suffix Array: S = ababaac						
Suffixes	Sorted	Array				
1: ababaac	5 : aac	$a_1 = 5$				
2:babaac	3:abaac	$a_2 = 3$				
3:abaac	1:ababaac	$a_3 = 1$				
4:baac	6:ac	$a_4 = 6$				
5 : aac	4:baac	$a_5 = 4$				
6 : ac	2:babaac	$a_6 = 2$				
7 : c	7 : c	$a_7 = 7$				

Longest Common Prefix Array

Definition (Longest Common Prefix Array)

A longest common prefix array is an auxiliary data structure to the suffix array. It 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.

Let lcp(v,w) denote the length of the longest common prefix between two strings v and w. Then thr LCP array h is an integer array of size n such that h_1 is undefined and $h_i = lcp(S[a_{i-1}..n], S[a_i..n])$ for every $1 < i \le n$.

Lets see an example

Consider the string $S = banana@$.							
i	1	2	3	4	5	6	7
si	b	а	n	a	n	α	0

The corresponding suffix array $\mathfrak a.$							
i	1	2	3	4	5	6	7
$\mathfrak{a}_{\mathfrak{i}}$	7	6	4	2	1	5	3
1	0	а	а	а	b	n	n
2		0	n	n	a	a	a
3			a	a	n	@	n
4			0	n	a		a
5				a	n		0
6				@	a		
7					0		

Lets see an example

The LCP h is constructed by comparing lexicographically consecutive suffixes.

Longest Common Prefix Array							
i	1	2	3	4	5	6	7
hi	_	0	1	3	0	0	2

Suffix Array: Naive solution

- In principle, a non-empty string have n+1 suffixes (including itself and the empty string) and the sum of all lengths of suffixes is $\frac{n(n+1)}{2}$, then we can read all these characters in $\mathcal{O}\left(n^2\right)$ time.
- So, using a quasi-linear sorting algorithm we can solve the problem in $\mathcal{O}(n^2 \log n)$ time.

```
\label{eq:function_build_suffix_array} \begin{split} & \text{function } \text{build-suffix-array}(\text{char }*\text{str}): \text{int }*\text{begin} \\ & | & \text{function } \text{cmp}(\text{int } i, \text{int } j): \text{bool} \\ & | & \text{return } \text{strcmp}(\text{str}+i, \text{str}+j) < 1 \;; \\ & \text{int } n \leftarrow \text{strlen}(\text{str}); \\ & \text{int } *\alpha \leftarrow \text{new int } [n]; \\ & \text{for } i = 0 \text{ to } n - 1 \text{ do } \alpha[i] \leftarrow i \;; \\ & \text{std::sort}(\alpha, \alpha + n, \text{cmp}); \\ & \text{return } \alpha; \end{split}
```

LCP: Naive solution

```
function lcp(char *v, char *w): int begin
    int k \leftarrow 0:
    while v[k] * w[k] \neq 0 \&\& v[k] = w[i] do k \leftarrow k+1;
    return k;
end
function build-lcp(char *str): int *begin
    int *a \leftarrow build-suffix-array(str);
    int n \leftarrow strlen(str):
    int *h \leftarrow new int [n]:
    for i = 1 to n - 1 do
       h[i] \leftarrow lcp(str[a[i-1]], str[a[i]]);
    end
    return h;
end
```

References |

- Standford University
- HackerRank
- Code Forces
- Code Chef
- Wikipedia