Suffix Array and Longest Common Prefix

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

[2]Algorists Group

July, 2016

Outline

Suffix Array

Definition (Suffix Array)

A **suffix array** is an array that contains the indexes of all lexicographically sorted string suffixes. Formally, let a 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 $\mathtt{lcp}(v, w)$ 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 $h_i = \mathtt{lcp}(S[a_{i-1}..n], S[a_i..n])$ for every $1 < i \leqslant n$.

Lets see an example

Consider the string $S = banana@$.

i	1	2	3	4	5	6	7
s_i	b	a	n	a	n	a	@

The corresponding suffix array a.

i	1	2	3	4	5	6	7
a_i	7	6	4	2	1	5	3
1	@	a	a	a	b	n	n
2		@	n	n	a	a	a
3			a	a	n	@	n
4			@	n	a		a
5				a	n		@
6				@	a		
7					@		

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
h_i	–	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 $\mathcal{O}(n^2)$ time.
- So, using a quasi-linear sorting algorithm we can solve the problem in $\mathcal{O}(n^2 \log n)$ time.

```
function build-suffix-array(char *str): int *begin
    function cmp(int i,int j): bool
        return strcmp(str+i,str+j) < 1 ;
    int n ← strlen(str);
    int *a ← new int [n];
    for i = 0 to n − 1 do a[i] ← i ;
    std::sort(a,a+n,cmp);
    return a;
end
```

LCP: Naive solution

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

References I

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