Suffix Array (SA)

Manber & Myers '90

I. The Problem

1. Review of the Problem solved by KMP (How many have tried KMP?)

Input: 2 strings, say, str1 and str2

Output: All the positions in str1 that str2 appears

2. Problem solved by SA

Input: a single string, say, str1

M queries, $(pos1, pos1')$ $(pos2, pos2')\cdots(posM, posM')$

Output: For each query (pos_i, pos_i'), the maximum length of matching substring of str1 starting from (pos_i, pos_i')

3. Example

3.1

str1	a	b	a	b
pos	0	1	2	3

queries: (0,2) => 2 (e.g., ab)

(0,3) => 0

(1,3) => 1 (e.g., b)

3.2 Problem Reduction

SA can solve the problems of KMP, and more.

For example, search aba in abab:

step1: build suffix array for abababa (0 1 2 3 4 5 6)

step2: issue queries (0,4) (1,4) (2,4) (3,4)

II. The Algorithm

best known: $O(N+M)$ (aka. suffix tree)

this talk: $O(N \log N + M)$

II. The Algorithm

There are 3 steps in this algorithm: step1: compute the suffix array
step2: compute the height array
step3: compute the RMQ

1. Overview

Input: banana (012345) ⇒ all the suffixes ⇒ the result of step1 aka. suffix array! ⇒ the result of step2 aka. height array! ⇒ RMQ

all the suffixes		the result of step1 aka. suffix array!		the result of step2 aka. height array!	
		0	a	0	X
a		1	ana	1	1
na		2	anana	2	3
ana		3	banana	3	0
nana		4	na	4	0
anana		5	nana	5	2
banana					

RMQ queries ⇒ (1,5)

position 1 ⇒ rank 2

position 5 ⇒ rank 0

The minimum of (0,2] in height array is 1, so that the query result is 1

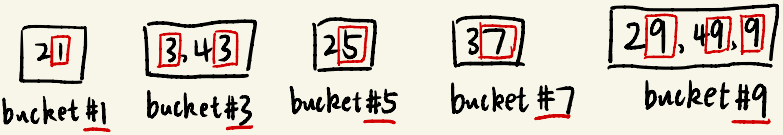
2. step1 - computing the suffix array

2.1 example of Bucket Sort (more details on Wikipedia)

sort the numbers: 29, 25, 3, 49, 9, 37, 21, 43

Iteration 1

input array: 29, 25, 3, 49, 9, 37, 21, 43



result of iteration 1: 21, 3, 43, 25, 37, 29, 49, 9

→ the last digit is in increasing order

Iteration 2

result of iteration 1: 21, 3, 43, 25, 37, 29, 49, 9 → the last digit is in increasing order

bucket #0	bucket #2	bucket #3	bucket #4
3, 9	21, 25, 29	37	43, 49

result of iteration 2: 3, 9, 21, 25, 29, 37, 43, 49 → what we want!

2.2 Summary of Bucket Sort

- each iteration considers a more significant digit
- for each iteration, each item is split into 2 parts
  - part 1: more significant, unsorted, defines the bucket number
  - part 2: less significant, already sorted, defines order of entering the buckets

2.3 From Numbers to Strings

a simple example. compute the suffix array for "abab"
0 1 2 3

iteration 1

substring of length 1

dictionary order ↓

a	pos = 0
a	pos = 2
b	pos = 1
b	pos = 3

iteration 2

substring of length 2

part 1	part 2	
a	b	pos = 0
a	b	pos = 2
b	\0	pos = 3
b	a	pos = 1

iteration 3

substring of length 4

part 1	part 2	
ab	\0 \0	pos = 2
ab	ab	pos = 0
b \0	\0 \0	pos = 3
b a	b \0	pos = 1

note: \0 is the end of string

iteration 3

substring of length 4

ab \0 \0 pos=2

abab pos=0

b \0 \0 \0 pos=3

bab \0 pos=1

This is the Suffix Array (SA)!

it is a rank-to-position mapping

SA[0] = 2 → the 1st suffix after sorting is ab

SA[1] = 0 → the 2nd suffix after sorting is abab

SA[2] = 3 → the 3rd suffix after sorting is b

SA[3] = 1 → the 4th suffix after sorting is bab

2.4 Exercise "banana"

We have constructed the suffix array for "abab". The construction for "banana" is left for you as an exercise. The result is:

string	b	a	n	a	n	a
position	0	1	2	3	4	5

SA[0] = 5 → for "a"

SA[1] = 3 → for "ana"

SA[2] = 1 → for "anana"

SA[3] = 0 → for "banana"

SA[4] = 4 → for "na"

SA[5] = 2 → for "nana"

3 step2 - compute the height array

3.1 The meaning of the height array. Recall the height array for "banana":

height[0] = NULL

height[1] = 1 "a", "ana" have matching prefix of length 1

height[2] = 3 "ana", "anana" have matching prefix of length 3

height[3] = height[4] = 0 "anana", "banana", "banana", "na" have no matching prefix

height[5] = 2 "na", "nana" have matching prefix of length 2

3.2 Constructing Algorithm (Pseudo code)

```
int match=0
for ( int pos=0; pos < len(str); pos++ ){
    // iterate all positions in str one-by-one
    int prev = SA[ position_to_rank[pos] -1 ]
    // height is comparing 2 substrings next to each other in SA
    while (str[ pos + match ] == str[ prev + match ])
        match++
    height [ position_to_rank [pos]] = match
    // the index of height array is the rank
    if (match >0) match--
    // prepare for the substring starting from pos+1
}
```

3.3 Key Lemmas for understanding the code

Lemma1. if ① str1 is the previous one of str2 in the dictionary
and ② str1 and str2 has matching prefix of length >0
then removing the 1st char of str1 and str2 (i.e., str1[1:] str2[1:])
str1[1:] appears before str2[1:] in the dictionary

Lemma2. if str1, str2, str3 appear in the dictionary in this order
then length of (str1, str3) matching prefix
$\leq$ length of (str2, str3) matching prefix

4. step3 – compute RMQ (Regional Minimum Query)

compute the RMQ over the height array costs O(nlogn) for pre-processing and O(1) for each query.

III An Interview Question

Given a string, find the longest substring that is a palindrome.

For example, the string abcddcalevelab has substring "cddc" and "alevela" which are palindrome, and the later is the longest.