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

3. Example
   3.1 str1 abab str2 queries: (0, 2) => 2 (e.g., ab)
   pos 0 1 2 3
   (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
   step2: issue queries (0, 4) (1, 4) (2, 4) (3, 4)
II. The Algorithm

The 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: step 1: compute the suffix array

step 2: compute the height array

step 3: compute the \( \text{RNIQ} \)

1. Overview

<table>
<thead>
<tr>
<th>all the suffixes</th>
<th>the result of step 1 aka. suffix array</th>
<th>the result of step 2 aka. height array</th>
<th>( \text{RNIQ} ) queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0 a</td>
<td>0 X</td>
<td>(1, 5)</td>
</tr>
<tr>
<td>ana</td>
<td>1 ana</td>
<td>1 1</td>
<td>position 1 ( \Rightarrow ) rank 2</td>
</tr>
<tr>
<td>nana</td>
<td>2 ana</td>
<td>2 3</td>
<td>position 5 ( \Rightarrow ) rank 0</td>
</tr>
<tr>
<td>anana</td>
<td>3 banana</td>
<td>3 0</td>
<td>The minimum of list ( [0, 2] ) in height array is 1, so that the query result is 1</td>
</tr>
<tr>
<td>banana</td>
<td>4 na</td>
<td>4 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 nana</td>
<td>5 2</td>
<td></td>
</tr>
</tbody>
</table>

2. step 1 - 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

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 “a b a b”

iteration 1
substring of length 1
a pos = 0
a pos = 2
b pos = 1
b pos = 3

dictionary order

iteration 2
substring of length 2
a b pos = 0
a b pos = 2
b a pos = 0
b a pos = 1

iteration 3
substring of length 4
ab ab pos = 0
ab ab pos = 2
b a b a pos = 0
b a b a pos = 1

note: \0 is the end of string
This is the Suffix Array (SA)!

It is a rank-to-position mapping

<table>
<thead>
<tr>
<th>Index</th>
<th>SA String</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ab</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>abab</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>bab</td>
<td>1</td>
</tr>
</tbody>
</table>

Exercise "banana"

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

<table>
<thead>
<tr>
<th>String</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>banana</td>
<td>0 2 3 4 5</td>
</tr>
</tbody>
</table>

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. step 2 - Compute the height array

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

"a", "ana" have matching prefix of length 1
"ana", "anana" have matching prefix of length 3
"anana", "banana" have no matching prefix
"banana", "na" have matching prefix of length 2
"na", "nana" have matching prefix of length 2
3.2 Constructing Algorithm (Pseudo code)

```c
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

**Lemma 1.** 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

**Lemma 2.** if str1, str2, str3 appear in the dictionary in this order

then length of (str1, str3) matching prefix

≤ length of (str2, str3) matching prefix
4. step 3 – compute RMQ (Regional Minimum Query)
compute the RMQ over the height array costs $O(n \log n)$ 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 "cdcc" and "alevela" which are palindrome, and the later is the longest.