1 Linear time construction of suffix array

We now focus on the problem of building the suffix array (sometime also called the POS array to distinguish from the LCP array) in $O(n)$ time. This problem has puzzled people for ten years, and three groups of people solved it (a little different methods) in 2003. All the three methods use the same high-level idea: that is, first sort part of string (called a sample), and then sort the rest of string from the sorted sample. The three methods differ by the way of choosing samples and how to merge.

The main idea is divide and conquer. But how are we going to divide? One may say do first half and second half, but this does not seem to do much. Another is to say odd position or even position. Why is this useful? Sorting suffix is different from sorting a list of numbers. suffix $i$ and $i+1$ is not independent (unlike sorting an array of numbers)! Look at two positions: $i$ and $j$. If we know which one is (lexicographically) bigger, then we can decide in constant time which of suffix $i+1$, $j+1$ bigger? Why? Compare the first char, and if equal then $(i+1,j+1)$ pair tells us the result. This is roughly the high-level idea this approach takes, but with an important twist: instead of doing odd-even, we do tri-partition: $0,1,2 \mod 3$. Let us give an example to see how this works.

Here we start numbering position from 0. As an example, consider $T= y\ a\ b\ b\ a\ d\ a\ b\ b\ a\ d\ o$. For this string, $\text{POS}= [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0]$. We list suffix for 1 mod 3 (in an array called $B_1$), and list suffix for 2 mod 3 (in an array called $B_2$). For this example, $B_1=[1,4,7,10]$, $B_2=[2,5,8,11]$.

**Step 1: Sort suffix in $B_1$ and $B_2$**

For this purpose, we will sort this string: $[\text{abb}\][\text{ada}\][\text{bba}\][\text{do}$ | $\text{bba}\][\text{dab}\][\text{bad}\][\text{o}$$. Note each triple (encapsulated in []) becomes a new character. This string is longer than the original string, but we just replace the triple with the ranks of these triples in the matrix by their ranks. This is done by radix sort of the letters present in the samples. This turns to:

- Abb 1
- Ada 2
- Bba 4
- Dos 6
- Bba 4
- Dab 5
- Bad 3
- O$$ 7

So we have the new string (by plugging in the ranks) as $R’= [1,2,4,6,4,5,3,7]$. Then we sort this recursively (and note $R’$ is shorter than $T$ by $1/3$). And suppose we get $\text{SA}(R’)= [8, 0, 1, 6, 4, 2, 5, 3, 7]$ (recall suffix start from 0). Use radix sort is the simple way of converting the combo alphabet back to integer alphabet, so that we can use recursion!

Also, create a rank for every suffix for the sampled positions. $\text{Rank}(R’)= x, 1, 4, x, 2, 6, x, 5, 3, x, 7, 8, x, 0, 0$ (note use 0 for the last two positions) Be careful: how do we know this? $\text{SA}(R’)= [8, 0, 1, 6, 4, 2, 5, 3, 7]$. Also, we can define $\text{rank}(R’)= 1, 2, 5, 7, 4, 6, 3, 8, 0$ (simply put the current rank in the corresponding position and increase the current rank by 1). This rank helps us
to determine which suffix of two suffixes is smaller. These suffixies have rank starting from 0 to 8 (i.e. suffix 8 is ranked 0, suffix 0 is ranked 1, and so on). We also note that these suffixes correspond to the suffixes in the original $T$: $[1, 4, 8, 2, 7, 5, 10, 11]$. Thus the original suffix 1 corresponds to suffix 1 in the new string, with rank 1 (second smallest). The original suffix 4 corresponds to suffix 2 in the new string, with rank 4. And so on. YW: I did not explain clearly here in the class. You should think about this when reading this part.

Step 2. Sort positions not sampled (i.e. mod 3 == 0)

Now this becomes easy: For a pair for each of these position ($S[i]$, rank($S[i + 1]$)), Then do radix sort of this pairs, and it takes only $O(n)$ time! Just line them up like before, and sort it. In our example, we have: (y,1), (b,2), (a,5), (a,7), which gives (0,0) < (a,5) < (a,7) < (b,2) < (y,1).

Step 3: Merge

Standard comparison merge! In more detail: we are merging one list contains 0 mod 3 suffix (called $M_0$), and the other list contains 1 or 2 mod 3 suffix (called $M_1$ and $M_2$). So, we have two cases: compare $M_0$ with $M_1$, and $M_0$ with $M_2$. If we compare $M_0$ with $M_1$, (say $s_0$ and $s_4$) we first see if $S[0] \neq S[4]$. If so, we are done (which suffix is larger is already known). Otherwise, we then look to see if $s[1]$ and $s[5]$, which one is larger (and this is already known in one of the list $B_1$ and $B_2$). If we compare $M_0$ with $M_2$ (say $s_0$ with $s_5$), we compare two characters first and then the case reduce to the original situation. Now we are done!

Finally, time analysis. We are doing recursion over $\frac{2n}{3}$, so we have: $T(n) = T(\frac{2n}{3}) + O(n)$. This gives $O(n)$ running time.