CSC 284/484 Advanced Algorithms - homework 5
applied homework due: Apr 16th, 11:59pm EST
theoretical homework due: Apr 16th, 11:59pm EST

Do not search for a solution online, do not use any written material when writing any part of the code (for example, no copy-paste, no open textbook when writing code, no reediting of an old source file from an old project, etc). You can discuss your solution(s) or problems/issues you are facing with your classmates. Do not share code.

PROBLEM 11 (applied)—implement one of the suffix array algorithms

- Test your algorithm on the attached test data.
- Write a little note indicating:
  - Whether your code passed all the tests (if not where is the problem).
  - Timing information (how long your code runs on each test).
- Your note should be typeset (LaTeX, Word, etc); send your code and note to the instructor and the TA (`breber@cs.rochester.edu`) (subject line: CSC 284/484 HOMEWORK 5).
- Allowed languages: C, C++, Java, Python (I discourage you from using Python—it is better for you to learn how to code algorithms in more efficient languages).

The objective is to implement construction of suffix arrays. An $O(n \log n)$ or $O(n(\log n)^2$ algorithm is fine (you can also implement the $O(n)$ DC3 algorithm if you want but it is not required).

INPUT FORMAT: The first line contains n the number of test cases. Then n lines follow, each line containing a string over alphabet {a,b,...,z} (lower case letters) of length at most 100,000.

OUTPUT FORMAT:

The output contains a line for each test case. The line contains the suffix array for the string (with a special \$ symbol attached (where we assume that \$ is smaller than any of the letters)).

EXAMPLE INPUT:

```
5
aaaaa
abcde
edcba
ababa
aaaab
```

EXAMPLE OUTPUT:

```
5 4 3 2 1 0
5 0 1 2 3 4
5 4 3 2 1 0
5 4 2 0 3 1
5 0 1 2 3 4
```

PROBLEM 12 (theoretical, bonus problem)

Typeset your solution (LaTeX, Word, etc) and send pdf to the instructor by email (subject line: CSC 284/484 HOMEWORK 5).

We have k strings $u_1, \dots, u_k \in \{a,b\}^*$. Let $n = |u_1| + \dots + |u_k|$. We want to get an $O(n)$ algorithm that outputs $\ell_1, \dots, \ell_k$ where ℓ_i is the length of the longest string that has at least 2 occurrences in at least i of the strings $u_1, \dots, u_k$.

For example, if $u_1 =$ababa, $u_2 =$cabaca, $u_3 =$cabcabca then $\ell_1 = 5$ (string "cabca" occurs twice in u_3), $\ell_2 = 2$ (string "ab" occurs twice in u_1 and u_3), and $\ell_3 = 1$ (string "a" occurs twice in u_1, u_2, u_3).