Assignment 6: Suffix Trees and Suffix Arrays

This problem set is all about suffix trees and suffix arrays, their applications, their properties, and related structures. We hope it solidifies your understanding of these amazing data structures!

Working in Pairs

We suggest working on this problem set in pairs. If you work in a pair, you should jointly submit a single assignment, which will be graded out of 9 points. If you work individually, we will grade out of 8 points, but will not award extra credit if you earn more than 8 points.

Due Wednesday, May 14 at 2:15PM at the start of lecture.

Problem One: Counting Substrings (2 Points)

Design an O(m)-time algorithm that, given a string T, outputs the number of distinct substrings of T.

Problem Two: *k*-Approximate Matching (4 Points)

The normal string matching problem is too restrictive in many genomics applications where, due either to sequencing errors or random mutations, a pattern string P might not exactly match anywhere in T even though it *almost* matches. The *k-approximate matching problem* is the following: given a string T of length m, a string P of length n, and a value $k \geq 0$, determine whether there is a substring of T of length n that matches P in all but at most k places. For example, given the strings

T = `thatthatisisthatthatisnotisnotisthatitititis` P = `matisse`

then P can be placed here relative to T such that it matches in all but 3 places:

```
thatthatisisthatthatisnotisnotisthatitititis
      matisse
```

However, there is no way to place P such that it matches in all but 2 places.

Design an O(km + n)-time algorithm for solving the k-approximate matching problem and prove that your algorithm is correct. As a quick aside, if $k \ll n$, this can be significantly faster than the naïve algorithm.

Problem Three: Variations on DC3 (2 Points)

Consider a variation on DC3 called DCk. In this variation, instead of computing the relative order of the suffixes at indices of T that are nonzero mod 3, we compute the relative order of the suffixes at indices that are nonzero mod k using an appropriate generalization of the approach used by DC3. We then use that ordering to sort the suffixes at positions that are zero mod k, then merge together the suffixes using a generalization of the DC3 approach

i. **(1 Point)** Will DC2 correctly compute a suffix array in time O(m)? If it does, why isn't it used instead of DC3? If it doesn't, why not? Justify your answer.

ii. **(1 Point)** Will DC4 correctly compute a suffix array in time O(m)? If it does, why isn't it used instead of DC3? If it doesn't, why not? Justify your answer.

Problem Four: Course Feedback (1 Point)

We want this course to be as good as it can be and would really appreciate your feedback on how we're doing. For a free point, please take a few minutes to answer the course feedback questions available at https://docs.google.com/forms/d/1a-RZsnCF7Qjz-wKbfii_p2BC8SLFccQnlvbsM0CO2V0/viewform. If you're submitting this problem set in a group, **please have each group member fill this out individually**.