CSE 584A Algorithms for Biosequence Comparison Spring Semester 2018

Homework 2

Assigned: 2/19/2018 *Due Date: 3/7/2018*

This homework must be completed and submitted electronically. Formatting standards, submission procedures, and (optional) document templates for homeworks may be found at

`https://classes.engineering.wustl.edu/cse584/ehomework/ehomework-guide.html`

Advice on how to compose homeworks electronically, with links to relevant documentation for several different composition tools, may be found at

`https://classes.engineering.wustl.edu/cse584/ehomework/composing-tips.html`

Please remember to

- **create a separate PDF file (typeset or scanned) for each problem;**
- **include any figures (typeset or hand-drawn) inline or as floats;**
- **upload and submit your PDFs to Blackboard before class time on the due date.**

Always show your work.

1. (**30%**) A *traversal* of a suffix tree visits all of its nodes in some order. To visit a node x, we call a procedure VISIT$(x.L, x.R, x.D, C)$, where

   - $x.L$ and $x.R$ are pointers to the leftmost and rightmost leaves that are descendants of x;
   - $x.D$ is the *depth* of x, that is, the length of the string labeling the path from the root to x;
   - C is a list of x's immediate children.

   A *postorder*, or "bottom-up," traversal of a tree has the property that all descendants of node x are visited prior to x. Our repeat-finding algorithms are examples of postorder traversal, as are related algorithms for, e.g., the longest common substring problem.

   Suppose you are given string S along with its suffix array A and adjacent LCP array L. Give an $O(|S|)$-time algorithm to simulate a postorder traversal of S's suffix tree *without* actually building the tree. Your algorithm should call VISIT once for every node x of this tree. The pointers $x.L$ and $x.R$ in each call should be replaced by indices in A of the leftmost and rightmost suffixes under x, while the child list C should now list the index in A of the leftmost (i.e. lexicographically least) suffix under each direct child of x.

   Hint: as you traverse the suffix array from left to right, maintain a stack that keeps track of the nodes on the path from the root of the tree to the current suffix.

2. (**20%**) A repeat α in a string S is said to be *supermaximal* if no instance of α occurs as substrings of an instance of some larger repeat. For example, in the string $S = acgtcacgtgacgtc$, the string "acgt" is *not* a supermaximal repeat, since one of its instances appears as substrings of an instance of the longer repeat "acgtc". However, *acgtc* itself *is* a supermaximal repeat, since its sole instance is maximal.

   Show how to extend the suffix tree-based repeat finding algorithm we discussed in class to enumerate all instances of supermaximal repeats in a string S in time proportional to their number, plus at most linear extra work in $|S|$.

3. (**50%**) An improved version of the Manber-Myers algorithm for suffix array construction is due to Larsson and Sadakane. They give a good description of their improvements in the following paper: `https://doi.org/10.1016/j.tcs.2007.07.017`.

   For this problem, you must read this paper, understand the proposed improvements, and implement the Larsson-Sadakane construction algorithm.

   To demonstrate that your algorithm works, you should build a suffix array for the human chromosome 1 string given in Homework 1. More specifically, if this input string is S, you should:

   (a) Append a terminal "$" character to S;

   (b) Compute its suffix array A;

   (c) Compute the *Burrows-Wheeler transform* (BWT) of S, i.e. a string whose ith character is $S[A[i]-1]$. (If $A[i] = 1$, the ith character of the BWT should be "$".)

   (d) Emit an output file as follows. The first line should be an integer indicating the offset of the "$" character in the BWT. After this line, emit the first 10 million and last 10 million characters of the BWT, with no line breaks or other spaces separating them.

   Where to Turn In Your Solution: each student will have a Git repository through GitHub Classroom.

   To turn in your solution, please check the following items into the assignment repo before the due date and time:

   - your code
   - the requested output
   - a README file describing any interesting properties of your implementation (including bugs, if known)

   Note that GitHub will not let you check the entire BWT of the input string into your repo, because it's too big.

   Advice on Implementation: This is a big file and will need a big chunk of memory to process. You can get away with storing the suffix array using a single 32-bit integer per position of the string, so hopefully you won't need more than 9 bytes of space per character, and the sorting step is done with a modified QuickSort that (unlike radix sort) can be done for each group in-place. You may use randomized pivot choice.

   If you are unable to process the full string due to memory limitations, let me know. You'll need a couple of gigabytes to build the full suffix array.