UC Berkeley, CS 194-1: Algorithms for Computational Biology (Spring 2012)

Problem Set 2

Instructor: Yun S. Song

Out: February 2, 2012 Due: February 16, 2012 (in class)

1. **(Implementation)** As mentioned in class, to construct the Burrows-Wheeler transform (BWT) of a string S, one usually constructs the suffix array for S first. In this exercise, you are asked to implement a linear-time algorithm for direct suffix sorting.

   Download `ps2_1_instruction.zip` from the course webpage (see Problem Sets tab), and read "`KS_algorithm.pdf`" (2 pages) and "`specification.txt`" included in the zip archive. The archive also contains example input and output files.

   You may use any of the following programming languages: C, C++, Java, Python, Perl, Ruby. Please provide a document detailing how one can compile and run your code. Your program should be able to do the following:

   (a) Read in an input string S over the English Alphabet from a file in FASTA format (at most 80 characters per line); see `http://en.wikipedia.org/wiki/FASTA_format` for further details. Your implementation should be case sensitive, with upper-case letters lexicographically smaller than lower-case letters. Assume that the last character of S is the terminating symbol \$.

   (b) Construct the suffix array for S using the KS algorithm.

   (c) Construct the Burrows-Wheeler transform L of S.

   (d) Output L in a file in FASTA format.

   (e) Implement a linear-time BWT decoding of L and output the result in another file in FASTA format. (If you have done everything correctly, this string should be identical to the original string S.)

   Note: This part of the problem set should be submitted via e-mail (to both the instructor and the GSI). You may pair up with someone for this programming assignment. Each group needs to turn in only one source code. If you form a group, you should include both of your names in the submission.

2. **(FM-index and Exact Pattern Matching)** Given a string S of size $|S| = n$, let $\Pi^{\text{sorted}}(S)$ denote the matrix containing the sorted cyclic permutations of S as rows, and let L denote the last column of that matrix (i.e., the BWT of S). The first column of $\Pi^{\text{sorted}}(S)$ is denoted by F. For σ a non-empty substring of S, we use $sp(\sigma)$ (respectively, $ep(\sigma)$) to denote the index of the first (respectively, last) row in $\Pi^{\text{sorted}}(S)$ such that σ occurs as a prefix of that row. (Recall that, in our convention, the row index takes value in $\{1, \ldots, n\}$.) As discussed in class, for a character c and $k \in \{1, \ldots, n\}$, $M[c]$ denotes the index of the first row where c appears in column F, while $OCC(c, k)$ denotes the number of occurrences of c in $L[1..k]$. Prove the following statements:

   (a) For all character c, if the concatenated string $c\sigma$ is a substring of S, then the following recursions hold:

   $$\begin{aligned} sp(c\sigma) &= M[c] + OCC(c, sp(\sigma) - 1), \\ ep(c\sigma) &= M[c] + OCC(c, ep(\sigma)) - 1. \end{aligned}$$

   (b) If σ is a substring of S, and $sp(c\sigma)$ and $ep(c\sigma)$ are computed using the above recursions, then $sp(c\sigma) \leq ep(c\sigma)$ if and only if $c\sigma$ is a substring of S.

3. **(Counting Co-optimal Alignments)** In the global alignment problem involving two sequences of lengths n and m, devise an $O(nm)$-time algorithm to compute the number of distinct co-optimal alignments.

4. (**Shortest Common Supersequence**) Let X and Y denote two strings of lengths n and m, respectively. A string Z is said to be a *common supersequence* for X and Y if both X and Y are subsequences of Z. (Recall that $X = x_1, \ldots, x_n$ is a subsequence of $Z = z_1, \ldots, z_k$, where $n \leq k$, if there exist $1 \leq i_1 < i_2 < \cdots < i_n \leq k$ such that $x_1, x_2, \ldots, x_n = z_{i_1}, \ldots, z_{i_n}$.) Devise an $O(nm)$-time algorithm for finding the *shortest common supersequence* of X and Y.

5. **(Global Alignment of Circular DNA)** Bacterial DNA is often organized into circular molecules. This motivates the following problem: Given two linear strings of lengths n and m, there are n circular shifts of the first string and m circular shifts of the second string, resulting in nm pairs of strings. We want to compute the global alignment of each of these nm pairs of strings. It turns out that this problem can be solved faster than the naive time bound $O(n^2m^2)$, which follows from applying the $O(nm)$-time dynamic programming algorithm to each pair of strings. Devise an $O(nm^2 + n^2m)$-time algorithm for finding an optimal global alignment for every pair of circular shifts.