02-714: Homework #2

Due: Oct. 3 at the start of class

Please write your answers neatly or typeset them. You may discuss the problems with your current classmates, but you must write your own solutions entirely independently. If you need to make any assumptions in order to solve a problem, state them explicitly. If you consult any sources, cite them.

1. (Adapted from JáJá.) A string x is a *peroid* of y if $y = x^k x_{1\ldots,j}$ where $k \geq 1$ and j is some integer ≥ 0. In other words, x is a period of y if y consists of repeated copies of x, where the last copy might only be partial.

   Let $pe(x)$ be the length of the smallest period of x. Prove the following:

   (a) If y has periods of length p and q, then it has one of size $\gcd(p, q)$ (gcd = greatest common divisor).

   (b) If y has a period of length $q \leq |y| - pe(y)$ then q is a multiple of $pe(y)$.

   (c) If y occurs in x at positions i and j, then $|j - i| \geq pe(y)$.

   (d) If y occurs in x at positions i and $i + d$, where $d \leq m - pe(y)$, then d is a multiple of $pe(y)$.

2. Give a $O(nm)$-time algorithm to *count* the number of optimal alignments between two strings x and y of length n and m, respectively.

3. Describe a simple modification to the edit-distance dynamic programming algorithm we saw in class that allows you to more easily use many processors at once to solve the problem faster.

4. Let x be a string of length n and y be a string of length m, and let $g > 0$ be an integer. Give an $O(nmg)$-time algorithm to compute the optimal alignment between x and y that uses at most g gaps, where a run of consecutive "-" characters counts as a single gap. Can you do it faster if you make some additional realistic assumptions?

5. Let X be a string of length n. Describe how to pre-process X so that given an index i, you can quickly find the shortest string u that starts at position i but occurs nowhere else. How fast can your algorithm solve this problem?

6. (Gusfield) Suppose you have a set $S = \{s_1, \ldots, s_r\}$ of r strings of total length m, give an $O(rm)$-time algorithm to compute the longest common substrings between all pairs of strings in S.

7. (Gusfield) (a) Given two strings X and Y and an integer k, give a linear-time algorithm to find a set of substrings of X, each of length $\geq k$, such that Y is a concatenation of these strings or determine that no such set exists. (b) Suppose we add the condition that the chosen substrings of X cannot overlap. Give some approach to solve this problem.