CS 222a Winter2015, HW 1 Due Tuesday Jan. 13 Unless stated otherwise, homeworks are to be done individually. Typed (and typeset) solutions are appreciated, and much easier to read. Read the policy on homeworks in the syllabus.

These problems are just to get you warmed up. They are not based on any material that has been taught yet.

1. Given two strings L and S that have an equal number of occurrences of each specific character, we define $D(S_1, S_2)$ as the minimum number of *transpositions* of adjacent characters needed to convert S_1 into S_2. For example, $S_1 = CBADA$ can be converted into $S_2 = ABDCA$ using exactly four transpositions. Notice that all the transpositions are done on S_1.

1a. Assume that S_1 and S_2 each have exactly one occurrence of each character, for example $S_1 = ACBD$ and $S_2 = DCAB$. Develop an efficient algorithm to compute the number $D(S_1, S_2)$ given any input strings S_1 and S_2 that obey the stated assumption. Argue that your algorithm is correct (try to find a rigorous yet simple argument) and discuss how efficient it is in terms of the number of operations it does. (What are the primitive operations in your algorithm?)

1b. Develop an efficient algorithm to actually transform S_1 into S_2 using exactly $D(S_1, S_2)$ transpositions. Argue that your algorithm is correct and discuss how efficient it is in terms of the number of operations it does. The operations are the operations done by the algorithm, not the number of transpositions needed.

1c. Now explain how to handle the case of computing $D(S_1, S_2)$ when those strings may have more than one occurrence of any character. Assume that both strings have the same number of occurrences of any particular character. Hint: One approach is to find a simple reduction of this problem to the previous one. Of course, give a proof of correctness and an analysis of the number of operations needed.

2. The following algorithmic problem arose in the field of Sociology. You are given two strings, for example $L = ABCCBCD$ and $S = AQCBAD$. For each character X in the alphabet, define $M(X)$ as the minimum number of times X appears in either L or S. For example, $M(A) = 1$ and $M(Q) = 0$ in the example above.

The problem requires us to remove characters from L and S so that for each character X, the number of remaining occurrence of X in each of L and

S is exactly $M(X)$. So far there is no real problem since we know exactly how many of each character must be removed from each string.

However, for some character(s) X there may be choices for which specific occurrences of X should be removed. Hence there are choices for what the resulting strings (call them S_1 and S_2) will be.

Now comes the problem: Among all possible ways to choose the required removals, we want to create resulting strings S_1 and S_2 to minimize $D(S_1, S_2)$. We call this the *MinDistRem problem*. For example, with L and S above, we can create $S_1 = ABCD$ and $S_2 = CBAD$ with $D(S_1, S_2) = 3$, or we could create $S_1 = S_2 = ACBD$ with $D(S_1, S_2) = 0$, and so we choose the latter as the solution to the MinDistRem problem.

2a. (10) Solve the MinDistRem problem for $L = ACDQDCGFDERAE$ and $S = EECDACWERGARF$.

2b. We would like to find an efficient algorithm for the MinDistRem problem. The following algorithm was proposed:

1. Let L be the longer string and S the shorter string. If the two strings are the same length, choose L and S arbitrarily.

2. For string L make a list (called LIST, duh!) of the characters (but not their positions) that will be removed. For each such character X include on LIST a number of copies of X equal to the number of occurrences of X that must be removed.

3. Set pointer p to 1 and pointers q and q' to 0.

4. Let X be the character in position p of S.

5. Scan L for the first occurrence (if any) of character X from position p forward in L. If such an X is found, set q' to the position of that found X.

6. If such an X was found in L, scan the characters from $q + 1$ to $q' - 1$ (inclusive) for any characters on the LIST. As each such character is found (if any are) remove it from L and remove one copy of it from the LIST.

7. Set q to q'.

8. Increment p by one.

9. Repeat steps 5 through 8 until p is at the end of S or until LIST is empty.

10. If LIST is not empty, scan L from its right end to find occurrences of characters on LIST. As each such character is found (if any are), remove it from L and remove one copy of it from LIST.

11. If sequence L is now shorter than sequence S, exchange the labels L and S, and repeat steps 2 to 10 on these two current strings L and S. Else terminate.

12. At termination, one the remaining string L is called S_1 and the remaining string S is called S_2.

2c. Execute the above algorithm on $L = ABCCBCD$ and $S = ACBAD$ and then on $L = ACDQDCGFDERAE$ and $S = EECDACWERGARF$.

2d. Argue that the algorithm always terminates within two executions of steps 2 to 10.

2e. Is it true that the algorithm always solves the MinDistRem problem? Justify.

3. If you think the above algorithm does not solve the MinDistRem problem, can you propose an efficient algorithm that does always solve it? (this may be a very hard problem)

Problem 4:

A "prototein" is a binary string that we embed on a two-dimensional grid. A legal embedding must satisfy the following rules:

1. Each character in the string gets placed on one point of the grid.

2. Each point of the grid gets at most one character of the string placed on it.

3. Two adjacnet characters in the string must be placed on two points that are neighbors on the grid in either the horizontal or vertical direction, but not both. That is, two points across a diagonal are not neighbors. (Note that an interior point on the grid has four neighbor on the grid, and that an outside point has three neighbors, and that a corner point has only two neighbors).

Rules 1,2,3 mean that we are embedding the string onto the grid as a self-avoiding walk, without deforming the string. See the figure below.

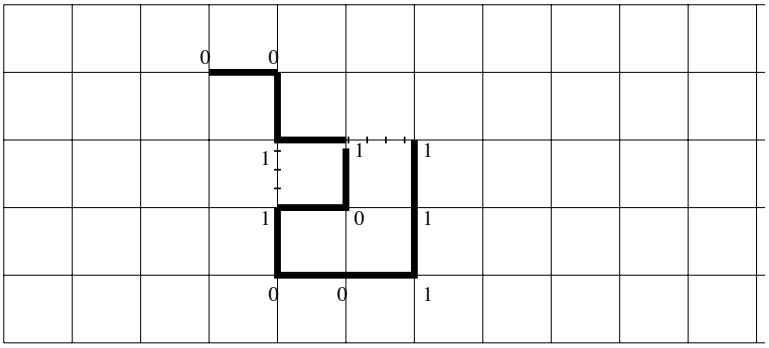

Figure 1: One possible embedding of the string 00110100111. The hatched lines show the contacts, two in this embedding. Can you find an embedding with more contacts?

A contact is formed for every pair of 1's that are not adjacent in the string, but are placed on neighboring points in the grid. Such a pair of 1's is called a "contact" in the embedding.

Given an input string, the general problem is to find an embedding of the string on the grid so as to maximize the number of contacts.

Your Problems:

4a) Prove (that is give a clear explanation for) the following claim: For any string, and any legal embedding of the string, the characters in positions i and j in the string can form a contact only if $|i - j|$ is odd. Try playing with some examples first to convince yourself that this is true, and then try to find a concise way to prove it.

4b) For any given string S, let E(S) be the number of 1's in even positions in the string, and let O(S) be the number of 1's in odd positions in the string. Let C(S) be the minimum of E(S) and O(S). Prove that the number of contacts, in any legal embedding of S, cannot exceed $2(C(S) + 1)$.

4c) If we can invent the string S, as well as decide on how to embed it, we could get alot of contacts, but that is cheating. Still if we can invent a string S, and embed it, so that the ratio of the number of contacts to the number of 1's in the string is high, that would be a good challenge. Find a

string S, and an embedding of it, that has a ratio of contacts to 1's of 7/6. Hint: you will need a string of length more than 25 (I think). So you will have to discover an idea, rather than just playing around.

4d) Prove that, over all possible strings and embeddings of those strings, the highest possible ratio of contacts to 1's is 7/6. Hint: the handshake lemma from graph theory (remember graphs from cs100 or cs20?) is helpful here. The rest is just case analysis.