CS 222 HW 4 Due Thursday Feb 3 (extension to Tuesday Feb. 8) Because of the midterm on Feb. 3 you have a lot of time for this homework, but don't leave it to the last minute. In particular, it is good to start on problem 4 early so you have time to let it percolate in your brain.

New comments (Feb. 2) are marked with stars.

1. In this problem we are interested in computing the number of optimal sequence alignments between two strings X and Y, where the definition of optimal is the same as in the version of sequence alignment we studied in class. However, we explicitly impose the requirement that in an optimal alignment we never place a space opposite another space (since this might be viewed as a match).

*** Some explanation of the requirement that a space opposite a space is not allowed. It is allowed in the original definition of a sequence alignment but we never said what its penalty should be. Clearly, the only reasonable costs are zero (if you view that as a match), or a positive number. In either case, even if allowed, it is not a good idea to have one, and the method we used to find a min cost alignment is not even capable of putting a space opposite a space. But now that we are asking for the number of optimal alignments, if someone were to say that a space opposite a space is actually a match, and should have penalty zero, then the number of optimal alignments would be infinite. That is clearly a useless answer, so we simply do not allow spaces opposite spaces.

1a. Set up recurrences that can be used to compute the number of distinct optimal sequence alignments between X and Y, and show how to calculate that number efficiently using the recurrences. Be sure to analyze the running time completely.

1b. Now suppose that instead of using your new recurrences to compute the number of optimal alignments, someone has just used the old recurrences to fill in the full DP table to compute the value of the optimal alignment. Explain how you can use this table to compute the number of optimal alignments. What is the time used for that computation?

2. Can the linear space sequence alignment method be modified to compute the number of optimal alignments? Why is that not easy? (I know why it is not easy, but I don't know if it can be done.)

3. Can the $O(n^2 \log n)$ method for circular string alignment be modified to run in $O(n)$ space? Why is that not as easy as the case of sequence alignment between two strings? (I know why it is not as easy, but I don't

know if it can be done in $O(n)$ space. I have a seen a claim that it can be done.)

***Actually (Feb. 2) I looked back at the claim, and it is that the computation can be done in $O(n \log n)$ space. So if you have time, consider that possibility.

4. A "Wagnerian String" 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 non-intersecting walk, without deforming the string. See the figure below.

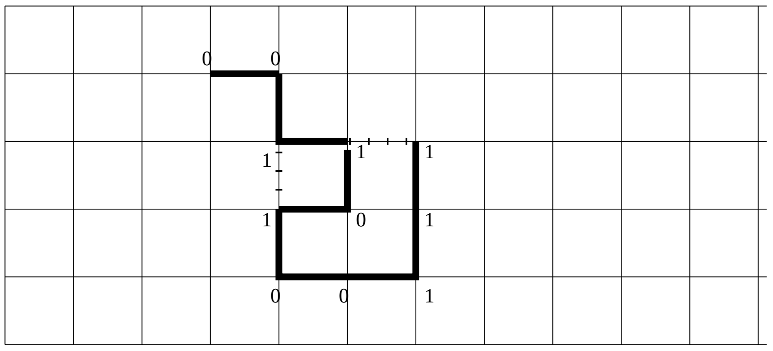

Figure 1: One possible embedding of the string 00110100111. The hatched lines show the contacts, two in this embedding. Can you find an embedding with more contacts?

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 $2(C(S) + 1)$.

c) 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. So you will have to discover an idea, rather than just playing around.

d) Prove that, over all possible strings and embeddings of those strings, the highest possible ratio of contacts to 1's is at most 7/6. Hint: the handshake lemma from graph theory is helpful. The rest is just case analysis.

***The handshake lemma is that the sum of node degrees in a graph is an even number. The degree of a node is the number of edges that touch it. The reason that this is called the handshake lemma is that if you go to a gathering and ask everyone there how many handshake they took part in, the sum will be an even number since every handshake is counted twice, once by each of the two people involved.