CS 124 Homework 1. Due October 6, but you can probably don't need that much time, and I will probably will release the next homework before then, so the sooner you get to it the better.

This homework is intended to "get you going" and does not rely on anything we have talked about in the class. It does assume that you already are familiar with Dynamic Programming. It may be that you can find an answer to the problem(s) on the web or in a text. Please don't look for them there or copy from there. You will learn more when you think the problems through on your own, and when I detect web-found answers to homework problems, I tend to get unpleasant.

The following problems involve dynamic programming. If you are not comfortable with DP and want to see a discussion on RNA folding, there are several books that discuss it. For one, see the algorithm text by Kleinberg and Tardos. Also, for some figures illustrating the basic RNA folding recurrences, see the powerpoint for the WABI 2009 talk by Yelena Frid posted at http://wwwcsif.cs.ucdavis.edu/~gusfield/talks.html. Pages 6-15 of the powerpoint describe the recurrences and DPs.

1 Basic RNA folding

The input to the basic RNA-folding problem consists of a string K of length n over the four-letter alphabet {A,U,C,G}, and an optional integer d. Each letter in the alphabet represents an RNA *nucleotide*. Nucleotides A and U are called *complimentary* as are the nucleotides C and G. A *matching* consists of a set M of *disjoint* pairs of sites in K. If pair (i, j) is in M, then the nucleotide at site i is said to *match* the nucleotide at site j. It is also common to require a fixed minimum distance, d, between the two sites in any match. A match is a *permitted match* if the nucleotides at sites i and j are complimentary, and $|i - j| > d$. [1] A matching M is *non-crossing* or *nested* if and only if it does not contain any four sites $i < i' < j < j'$ where (i, j) and (i', j') are matches in M. Graphically, if we place the sites in K in order on a circle, and draw a straight line between the sites in each pair in M, then M is non-crossing if and only if no two such straight lines cross.

Finally, a *permitted matching* M is a matching that is non-crossing, where each match in M is a permitted match. The basic RNA-folding problem is to find a permitted matching of *maximum cardinality*.

[1] We let d=1 here, for simplicity, but in general, d can be any value from 0 to n.

The original $O(n^3)$ time dynamic programming solution

Let $S(i,j)$ represent the score for the optimal solution that is possible for the subproblem consisting of the sites in K between i and j inclusive (where $j > i$). $B(i,j) = 1$ if the nucleotides in positions i and j are complementary, and is 0 otherwise.

Then the following recurrences hold:

$S(i,j) = \max\{$
$\quad S(i+1, j-1) + B(i,j)$ **rule a**,
$\quad S(i, j-1)$ **rule b** ,
$\quad S(i+1, j)$ **rule c**,
$\quad Max_{i<k<j} S(i,k) + S(k+1, j)$ **rule d**
$\}$

Rule a covers all matchings that contain an (i,j) match; Rule b covers all matchings when site j is not in any match; Rule c covers all matchings when site i is not in any match; Rule d covers all matchings that can be decomposed into two non-crossing matchings in the interval $i..k$, and the interval $k+1..j$.

These recurrences can be evaluated in different ordering of the variables i,j,k. A common suggestion is to evaluate the recurrences in order of increasing distance between i and j. That is, the solution to the RNA folding problem is found for all substrings of K of length two, followed by all substrings of length three, etc. up to length n.

An alternative $O(n^3)$-time dynamic programming solution

for j =2 to n **do**
  for i =1 to $j-1$ **do** {rules a and b}
    S(i,j)=max(S(i+1,j-1)+B(i,j) ,S(i,j-1))

  for i =$j-1$ to 1 **do**
    S(i,j)=max(S(i+1,j) , S(i,j)) (Rule c)

    for $k = j-1$ to i+1 **do** {The loop is called the Rule d loop}
      S(i,j)=max(S(i,j), S(i,k-1)+S(k,j)) (Rule d)

The recurrences used in this algorithm are the same as before, but the order of evaluation of S(i,j) is different.

Problem 1: Argue that this algorithm correctly solves the RNA folding problem and runs in $O(n^3)$ time.

Problem 2: Now suppose that instead of wanting to find the the permitted matching of largest cardinality, we want to *count* the exact number of permitted matchings. That problem can also be solved in $O(n^3)$ time by DP. One is tempted to redefine $S(i,j)$ to be the number of permitted matchings inside the interval $[i, ..., j]$ inclusive, and then use the following recurrences:

$S(i,j) = \{$
$S(i+1, j-1) + B(i,j)$ **rule a**,
$+ S(i, j-1)$ **rule b** ,
$+ S(i+1, j)$ **rule c**,
$+ \sum i < k < j [S(i,k) \times S(k+1, j)]$ **rule d**
}

This will not work.

Problem 3: Explain why the recurrences above will not correctly count the number of permitted matchings. Will they overcount or undercount? Now give recurrences that can be used to correctly count the number of permitted matchings, and give a DP that computes the count in $O(n^3)$ time. This may be a bit challenging.

2 Simpler Nested Matching Problems

Let S be a string consisting of a total of n left and right parentheses. The number of left parens need not equal the number of right parens. Here is a recursive definition of what it means for a string of left and right parens to be *properly nested* i.e., balanced: a) the string "()" is properly nested b) If A is a properly nested string and B is a properly nested string, then A B is a properly nested string. c) If A is a properly nested string, then "(A)" is a properly nested string.

For example, "((() (())) ())" is properly nested, while "(()(" is not. Is "()()()" properly nested under the above definition? Hint: Yes, but you might be tempted to say No if you don't understand recursion.

Here is an algorithm that checks if the parens are properly nested i.e., balanced. Scan the string S left to right. When a "(" is encountered, push it on a stack. When a ")" is encountered, check if the stack is empty. If so, then S is not properly nested. If the stack is not empty, then pop the stack.

When the end of S is reached, check if the stack is empty. If yes, then S is properly nested; if not, then S is not properly nested.

Problem 4. Explain (prove) that the algorithm correctly determines if S is properly nested. What is the running time of the algorithm? Justify you answer.

Problem 5. Modify the algorithm to become an $O(n)$ time algorithm that finds the smallest possible subset of parentheses to *delete*, so that the remaining parens are properly nested. Equivalently, find the largest subset of parens that can be properly nested. The algorithm must find a subset, not just its size. Justify your answer.

Problem 6. Suppose we modify the RNA folding problem so that the input strings are just on an alphabet of size 2, i.e., on binary strings. In this case 0 and 1 are permitted to match each other, but 0 cannot match a 0 and 1 cannot match a 1, and no symbol can match a neighbor, i.e., d = 1. We want to find a permitted matching to maximize the number of matched pairs. It is easy to modify the DP for RNA folding so that this problem can be solved in $O(n^3)$ time.

However, suppose $d = 0$, so we permit a symbol to match a neighbor, provided of course that the type of the neighbor is the opposite of the type of the symbol. Give an algorithm for this that runs in $O(n)$ time. Argue that your algorithm is correct and that it does run in $O(n)$ time. The solution is not by DP in this case.

Note: This problem only relates to the binary case, i.e. when the string only has 0's and 1's. Several people asked about the case of four symbols and correctly found that getting an $O(n)$ time algorithm for that case is not easy. In fact, I don't know of an $O(n)$ time algorithm for that case.

Problem 6a. Does the $O(n)$ solution work if $d = 1$, i.e., neighbors are not permitted to match even if they are different? Can the RNA folding problem (with four characters) be solved in $O(n)$ time if $d = 0$? If not, does it seem odd that the binary case can be solved in $O(n)$ time, but the 4-character case takes so much more time. Is that just a fluke of nature, or do you think it is because we have not figured out yet how to solve the RNA problem in $O(n)$ time?

Problem 6b: The above problem of finding a largest nested matching of 0's and 1's (when a number can match its neighbor if they are different) is very similar to the problem of finding the largest subset of left and right

parens which properly nest, and yet the solutions are a bit different. In fact, in the case of parens, it can happen that some "("s and some ")"s remain unmatched, while in the case of 0's and 1's, either all 0's or all 1's (or both) are matched. Explain the fundamental difference between these two similar problems, that explains the difference between the solutions.

3 First problem on string matching

Problem 7. Suppose you have a string matching algorithm that can take in (linear) strings S and T and determine if S is a substring (contiguous) of T. However, you want to use it in the situation where S is a linear string but T is a *circular* string, so it has no beginning or ending position. You could break T at each character and solve the linear matching problem $|T|$ times, but that would be very inefficient. Show how to solve the problem by only one use of the string matching algorithm. This has a very simple, cute, solution when you see it.