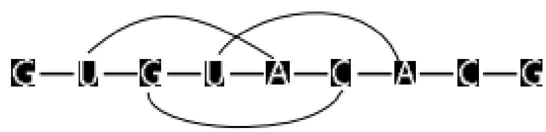
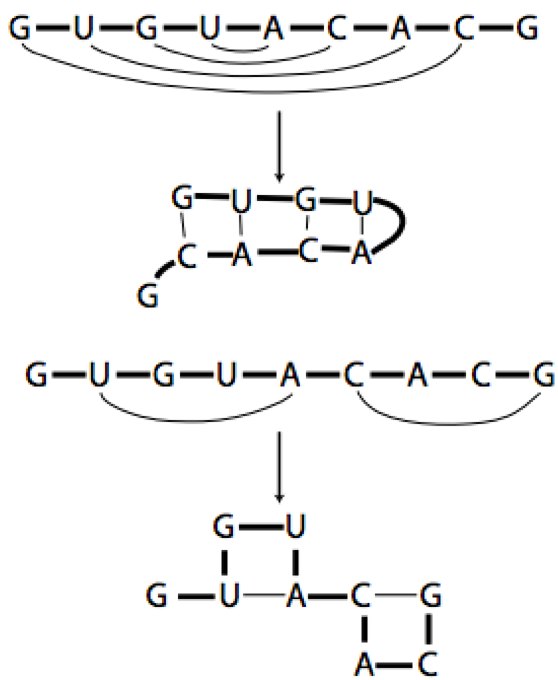
BCB 5300

Homework 3:
More fun with dynamic programming

1. RNAs adopt complex 3d structures that are important for biological functions. Pairs of positions in RNA with complimentary nucleotides can form *bonds*. Bonds between locations (i, j) and (i', j') are interleaving if $i < i' < j < j'$ and are noninterleaving otherwise; see the figure below.



(a) Interleaving bonds



(b) Non-interleaving bonds

Every set of noninterleaving bonds corresponds to a potential RNA structure. In a very naive formulation of RNA folding, one simply tries to find a maximum set of non-interleaving bonds. Design a dynamic programming algorithm for finding the largest set of noninterleaving bonds, given an RNA sequence as input.

(Note: There are more adequate folding models, which attempt to minimize energy - these are quite a bit more difficult!)

2. (a) Consider a model of virus infection where a virus infects a bacterium, and modifies a replication process by inserting:

    - at every A, between 1 and 5 additional A's
    - at every C, a run of 1 to 10 additional C's
    - at every G, a run of of G's of arbitrary length ≥ 1
    - at every T, a run of of T's of arbitrary length ≥ 1

    The gaps or insertions are allowed for in this virally modified final DNA sequence. For example, the sequence AAATTAAAGGGGCCCCCTTTTTTTTTCC is an infected version of ATAGCTC; however, AAAAAAAATTAAAGCCCCCTTTTTTTTTCC would not be, since it inserts too many A's in the first slot and did not insert any extra G's.

    Given two sequences v and w, give an efficient algorithm (including run time and space) that will determine if v could be an invected version of w.

   (b) Now consider a version where the virus will either delete a letter or will insert a run of arbitrary length, for each A,G,T,C it encounters in the original DNA. Give an efficient algorithm to decide if v could be an infected version of w under these circumstances.

3. Recall the dynamic programming solution to local alignment that we covered in class (or go re-read it in the textbook); this algorithm required O(mn) time and space. Adapt the divide and conquer framework from chapter 7 to get a linear space solution. (Your running time can still stay higher, though.)