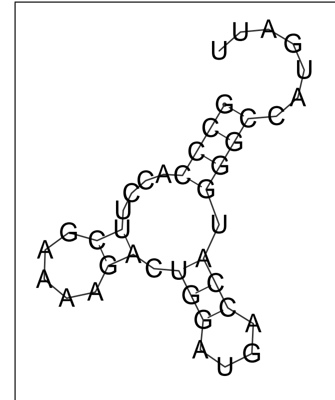
CSE 417
HW#3: RNA "folding"

Due: Monday, 1/31/2005 W.L.Ruzzo



To briefly summarize my lecture on RNA structure prediction, aka RNA folding, RNA molecules often fold back on themselves, forming stable double-helix structures akin to the famous DNA double helix, with G-C and A-U pairs forming. (We'll ignore other pairs, which sometimes form, too.) A commonly used set of rules for what structures may form is: $r_i * r_j$ allowed only if $i < j - 4$ (it can't bend too sharply) and if $r_i * r_j$ and $r_{i'} * r_{j'}$ are two pairs with $i \leq i'$ then either

1. $i = i'$ and $j = j'$ (r_i can't pair with two different bases)
2. $j < i'$ (one pair completely precedes the other), or
3. $j' < j$ (one is completely nested within the other).

Two pairs violating conditions 2-3 are called a "pseudoknot". Forbidding pseudoknots makes algorithms both simpler and faster, and means that there is a very simple linear representation of the structure using nested parentheses. For example, the structure depicted above is:

```
GCCCACCUUCGAAAAGACUGGAUGACCAUGGGCCAUGAUU
((((....((.....)).(((....))).)))).......
9 Pairs.
```

For a given RNA sequence, there will usually be a huge number of structures satisfying the above restrictions. Nature usually favors the "most stable" structure. One approximation to "most stable" is the structure having the maximum possible number of pairs (subject to the above restrictions). The goal of this homework is an algorithm for this problem.

Input: a single line containing a string of letters $x_1 x_2 \ldots x_n$ from the 4 letter alphabet {A,G,C,U} (all uppercase, for simplicity).

Output: one line containing the input, followed by a second line containing (one of) its optimal structure(s). (I say "one of" since there may be different structures with equal numbers of pairs; often slight variants of each other. Giving any one of them is OK.) This line will be a string of parens and dots, vertically aligned with the input string. A dot in the structure line means that the corresponding position in the RNA is unpaired; a left paren means it is paired with a position to its right, marked by a right paren. Furthermore, parens must be properly balanced/nested, so specific paired positions are marked by "matching" left/right parens.

Method: The Nussinov, Jacobsen algorithm, based on the following recurrence, computes, for each $1 \leq i < j \leq n$ the quantity $B[i, j]$ which is the maximum number of pairs in any folding of the substring $x_i x_{i+1} \ldots x_j$ of the input:

$$B[i,j] = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max \left(\begin{array}{l} B[i+1, j-1] + p_{i,j}, \\ \max\{B[i,k] + B[k+1,j] \mid i \leq k < j\} \end{array} \right) & \text{if } i < j-4, \end{cases}$$

where $p_{i,j} = 1$ if x_i could pair with x_j (i.e., G-C or A-U pairs), and $p_{i,j} = 0$ otherwise.

$B[i, j]$ is most easily represented as an n by n array, of which the lower-left triangle will be all zero. Since $B[i, j]$ depends on entries to its left in the same row and entries below it in the same

column, the convenient order in which to compute entries is to fill columns in left to right order, each column filled from the diagonal upwards. $B[1, n]$ will hold the maximum number of pairs.

What You Need To Do:

1. Implement the above algorithm for calculating $B[., .]$. For $n \leq 25$, print out B. (You'll probably find this useful for debugging.)

2. Devise and implement an algorithm to construct and print the structure (i.e. the string of parens and dots). This is a "traceback," similar to ones we've seen with other dynamic programming algorithms. You may (or may not) find it convenient to create auxilliary data structures while you're building B to facilitate this. I strongly recommend that you look for a recursive algorithm to do this, but it is not required.

   Print the input, with the structure aligned vertically below it. Also print the number of pairs.

3. Write a description of your traceback algorithm, explaining how it works/why it is correct.

4. Analyze (separately and collectively) the (big-O) run time of both parts 1 and 2.

5. Measure the actual run time of your algorithm (total time for both parts) on random RNA sequences of length 20–2000, say, plot them on a graph (e.g. Excel might be convenient, but is not required), and discuss how this compares to the theoretical performance predicted in step 4. For some tips on how to do the timing, see:
   `http://www.cs.washington.edu/education/courses/cse417/05wi/faq.html#timers`

Test Cases: Please show you output on the following two sequences.

```
1: AGCUCAUAUGGC
2: GCUCCAGUGGCCUAAUGGAUAUGGCUUUGGACUUCUAAUCCAAAGGUUGCGGGUUCGAGUCCCGUCUGGAGUA
```

As stated above, for sequence 1 (but not sequence 2), print out your B matrix. FYI, Sequence 2 is an arginine tRNA from *Trypanosoma brucei,* the African sleeping sickness parasite; cf. Mottram,J.C.; Eier,W.; Sloof,P.; Bell,S.; Nelson,R.G.; Barry,J.D.; tRNAs of Trypanosoma brucei J. Biol. Chem. 266:1 (1991)

What To Turn In: Email your code (and only that) to `ruzzo@cs.washington.edu`; print out and turn in a listing of it, its output on the tests above, and your write-ups of steps 3–5 above.

Language: C/C++ or Java; Talk to me before beginning if you prefer something else.

Appendix - Correctness of the Method: For completeness, here's a synopsis of the correctness proof of the Nussinov-Jacobsen method (also given in lecture). For substring $x_i, \ldots, x_j$, position i is either paired or not, and if paired is either paired to position j, or to some position $i < k < j$. It's certainly not paired if it's within 4 positions of j. If it is paired with j, then the best pairing overall couples that single pair with the best possible pairing of the substring between them, which is the first term in the "max." Similarly, if both i and j are unpaired, the same term in the recurrence applies (with $p_{i,j} = 0$). Alternatively, if i is paired with some $k \neq j$, then the best overall pairing consists of the best pairing between i and k plus the best one between $k + 1$ and j. The second term of the "max" checks all possible k for the best choice of k. (Note that this is where the lack of pseudoknots is critical: if $r_i * r_k$ or $r_{k+1} * r_j$ are paired, then we can treat substring i through k completely independently from $k + 1$ through j.)

Just for fun: This is *not* required, but if you're curious to see pretty diagrams of the structures your algorithm predicts, paste a sequence/structure (two separate lines of exactly equal length, ≤ 200) into `http://abstract.cs.washington.edu/~ruzzo/fold.pl` and click the button. (It's held together with duct tape; don't panic if it doesn't work...)