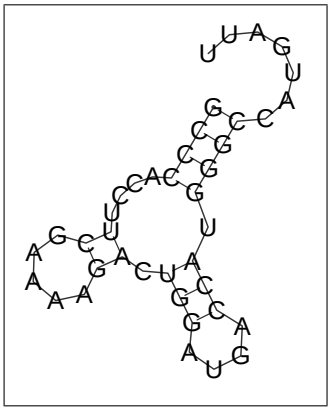
CSE 417
HW#4: RNA "folding"

Due: Friday, 2/17/2006 W.L.Ruzzo



To briefly summarize section 6.5 and 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                    [1]
((((....((.....)).(((....))).))))........                   [2]
9 Pairs.                                                    [3]
```

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), as in line [1] above.

Output: one line containing the input, a second line containing (one of) its optimal structure(s), as in [2] above, plus a third line giving the total number of pairs in that structure, as in [3]. 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. The structure 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: Use the Nussinov, Jacobsen algorithm, described in section 6.5 of the text.

What You Need To Do:

1. Implement the above algorithm for calculating OPT$[i, j]$. For $n \leq 25$, print out OPT. (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 OPT 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/cse417/06wi/faq.html#timers`

   As to comparing the measured to the theoretical performances, saying, e.g., "n^2 goes up and so does my graph" isn't very persuasive evidence. We'll talk in class about how you can do something better.

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

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

As stated above, for sequence 1 (but not sequence 2), print out your OPT matrix.

FYI, Sequence 2 is a naturally occuring example, specifically 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: (a) Electronically turn in your code (and only that) via the e-turnin form linked from the course web page. (b) Print out and turn in, in class, a listing of your code, its output on the tests above, and your write-ups of steps 3–5 above (about 2-3 pages).

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

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...)