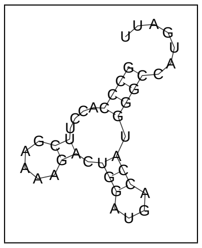
CSE 417
Algorithms & Computational Complexity
Assignment #7 (rev. a)
Due: Wednesday, 3/6/19

Turnin: Gradescope again; @uw email and gradescope password as before. On-time turn-in deadline is 11PM.

These problems relate to RNA secondary structure prediction (aka "RNA folding"), described in lecture and section 6.5 of the text. In short, you will implement Nussinov's algorithm and its associated "traceback" routine.

Structure: RNA (secondary) structure is often diagrammed as shown at right, which nicely depicts paired and unpaired regions. However, these pictures are somewhat awkward to generate. "Dot-bracket" notation is completely equivalent and simpler to generate. This is a string of parens and dots, of the same length as the RNA string. A dot 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, (a consequence of the "no pseudoknots" rule) so specific paired positions are marked by "matching" left/right parens. The sequence and structure shown below are the same as in the figure at right.



```
GCCCACCUUCGAAAAGACUGGAUGACCAUGGGCCAUGAUU                  [1]
((((....((.....)).(((....))).)))).......                  [2]
Length = 40, Pairs = 9, Time = 0.00006 sec                [3]
                                                          [4]
```

Core Algorithm: You should have a subroutine named "Nussinov" whose single parameter is a string of letters $x_1x_2\dots x_n$ from the 4 letter alphabet $\{$A,G,C,U$\}$ (all uppercase), as in line [1]. It should calculate the Nussinov OPT table, call your traceback routine, and print the outputs specified below, including the time it took to do all this.

Traceback: Also provide a traceback routine generating one of the optimal structures corresponding to your OPT table. 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. E.g., there are 14 different optimal structures for the sequence on line [1]. (The structure shown on line [2] is *not* one of them; it just serves to illustrate the format).

Output: For each "Nussinov" call, print to standard out one line containing the input, like [1] above, a second line containing (one of) its optimal structure(s) (i.e., output of your traceback), formatted as in [2] above and vertically aligned with the RNA sequence, plus a third line giving (i) the length of the input, (ii) the total number of pairs in that structure, and (iii) the time, in seconds (or fractions thereof) as in [3]. Additionally, for a length n input, if $n \leq 25$, print the $n \times n$ OPT matrix calculated by Nussinov's algorithm; print one line per row with n white-space-separated integer values per line, preferably keeping columns vertically aligned. Follow all of this by one blank line, as in [4].

Input: Your main program should read a sequence of lines from "standard input" each containing one such string, and call "Nussinov" on each. Different lines may be of different lengths (different "n"). For timing purposes, you should also generate *random* sequences of length, say, $n = 2^k$ for $4 \leq k \leq 12$ or more, with A,C,G,U independently equally likely in each position. Generate at least one sequence of each length, preferably several. Call "Nussinov" on each. Generating and processing these random sequences should be *optional*, and by default this option should be *"off"* so that our hard-working TAs don't have to endure the wait time, but it should be obvious how to enable it. E.g., your main program might look like:

```
while(!end_of_file(STDIN)){
  seq = read_a_line();
  Nussinov(seq);
}
if(FALSE){ // True to execute timing tests, False to skip them
  for(k=4; k<=12; k++){
    Nussinov(random_seq(2^k))
    Nussinov(random_seq(2^k))
    Nussinov(random_seq(2^k))
  }
}
```

What You Need To Do:

1. [18 points] Implement the Nussinov algorithm for calculating $\textsc{Opt}[i, j]$.
2. [2 points] The conventions used to index the $\textsc{Opt}$ table differ between the book and the slides; *clearly state which convention you are using in a comment near the top of your Nussinov code* .
3. [20 points] Devise and implement a traceback algorithm to construct the structure string (i.e., the string of parens and dots). I strongly recommend that you look for a recursive algorithm to do this, but it is not required. If you'd like, you may create auxiliary data structures while you're building $\textsc{Opt}$ to facilitate the traceback, but I recommend against this approach.
4. [20 points] As stated above, arrange to read strings from STDIN, optionally (but by default do *not*) generate random strings, process each, including printing the input as in [1], with the structure aligned vertically below it as in [2], and also the print the simple summary statistics as in [3]. Additionally, print the $\textsc{Opt}$ matrix if $n \leq 25$, and finally the blank separator line as in [4]. All output should go to standard out. For the specified test cases below, capture this output in a text file named `out.txt` and include it with your turn-in.
5. [20 points] Write a description of your traceback algorithm, explaining how it works/why it is correct.
6. [10 points] Analyze (separately and collectively) the (big-O) run time of the algorithms in steps 1 and 3.
7. [10 points] Measure the actual run time of your algorithm (total time for both parts) on random RNA sequences of length 16–4096, 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 6. For some tips on how to do the timing, see the FAQ page.

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

```
1: AGCUCAUAUGGC
2: GCCCACCUUCGAAAAGACUGGAUGACCAUGGGCCAUGAUU
3: GCUCCAGUGGCCUAAUGGAUAUGGCUUUGGACUUCUAAUCCAAAGGUUGCGGGUUCGAGUCCCGUCUGGAGUA
```

As stated above, for sequence 1 (but not the others), print out your $\textsc{Opt}$ matrix.

FYI, Sequence 3 is a naturally occurring 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).

Language: You may use C, C++, C#, Haskell, Java, Lisp, ML, Perl, Python, R, or Ruby; talk to me before beginning if you prefer something else.

What/How To Turn In: There will be a two-part turn-in via Gradescope again:

- Turn in your code and `out.txt` for the specified test cases (steps 1–4) to Gradescope hw7code.
- Turn in a single .pdf file with your written answers to steps 5–7 to Gradescope hw7written.

Revision History:

Rev a: Reorganized, added details, removed "draft" label. — 3/3/19.