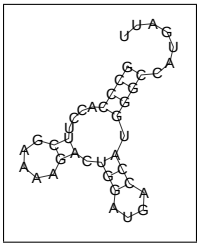
CSE 417
Algorithms & Computational Complexity
Assignment #8 (rev. a)
Due: Friday, 3/4/22

This is the final version; re-read it carefully if you've looked at draft versions.

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



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 logically 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 the matching right paren. (Parens must be properly balanced/nested as a consequence of the "no pseudoknots" rule.) The sequence and structure shown below are the same as in the figure at right.

```
GCCCACCUUCGAAAAGACUGGAUGACCAUGGGCCAUGAUU                [1]
((((....((.....)).(((....))).))))........               [2]
Summary: 9, 40,  0.06                                   [3]
                                                        [4]
```

Input: Your main program should read a sequence of lines from "standard input" each containing one such string from the 4 letter alphabet {A,G,C,U} (all uppercase), as in line [1], and call "Nussinov" (below) on each. Different lines may be of different lengths (different "n").

Core Algorithm: You should have a subroutine named "Nussinov" whose single argument is a string of letters as specified above. It should calculate the Nussinov OPT table, call your traceback routine, and print the outputs specified below. The conventions used to index the OPT table differ between the book and the slides; *follow the conventions in my slides*.

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. Give any one of them. 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 echoing 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.
- Additionally, for a length n input, if $n \leq 15$, print the $n \times n$ OPT matrix calculated by Nussinov's algorithm. Print one line per row (smallest index first for both rows and columns) with n white-space-separated integer values per line, preferably keeping columns vertically aligned; do not have blank lines above/below/interspersed with this.
- Also print a "Summary" line, as shown in [3] above, giving (comma-separated) (i) the total number of pairs in that structure, (ii) the length of the input, and (iii) the time taken for this calculation, in milliseconds (or fractions thereof).
- Finally, follow all of this by one blank line, as in [4]. (One blank lines per input in total; no other blank lines should appear.)

(The bracketed numbers, "[1]"..., should not be part of your output; they're just to allow me to reference the specific lines in this writeup. The examples linked at the end of this writeup illustrate the reqjuired format.)

Turnin: You are writing one program, but your turnin (and our grading) will be split into three parts:

1. *Traceback:* Give a traceback algorithm to construct a dot-bracket structure string. I strongly recommend that you look for a recursive algorithm for this, but it is not required. As usual, describe it in English, explain how it works/why it is correct, and analyze its run-time, and compare to the run-time of Nussinov OPT table construction.

2. *Timing:* For timing purposes, your program 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; more intermediate lengths and larger maximum lengths are fine, but will take more time. Call "Nussinov" on each. Generating and processing these random sequences should be *optional* in your program, 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++){
    for(reps=1; reps <=3; reps++){
      Nussinov(random_seq(2^k))
    }
  }
}
```

   Graph measured time versus sequence length. Overlay on the plot an interpolated trend line for the expected asymptotic growth rate. For comparison, fit a quadratic trend line to the same data. (Excel might be convenient for these steps, but many other tools can easily do it, too.) Discuss how this compares to the theoretical performance predicted in Q 1. I am expecting a brief write-up, say 1-2 pages, summarizing and discussing these findings, with the requested graph. For some tips on how to do the timing, see the FAQ page.

3. *Code:* Turn in your code. Again, you may use Java or Python. Name it hw8.java or hw8.py. In summary, you should have a subroutine called Nussinov with a single string parameter. It should print its argument to stdout (i.e., output line [1]), implement the Nussinov algorithm for calculating $\text{OPT}[i, j]$ (following the indexing conventions shown in my slides), call your traceback subroutine to calculate the dot-bracket structure, print it ([2]), print OPT if $n \leq 15$, print the summary line ([3]), and a blank line ([4]). Your main program should read strings from STDIN and process each by calling Nussinov. Optionally (but by default do *not*) generate random strings, and process each as above. All output should go to standard out.

The example presented earlier, plus a few others (including a biologically functional RNA), together with expected outputs on these examples in the specified format (almost; see below), are available here:

- `hw8-trna.txt`
- `hw8-trna-out.txt`

The output is in the format specified in the assignment, except: (1) my summary line adds a fourth field (which you should *not* do), giving the total number of distinct optimal structures, and (2) it shows all of the optimal structure lines, not just one. You only need to generate *one* dot-bracket line; this way you can see if your one structure is among the ones my program found on these examples. Also, my code echoes input lines starting with "#" to stdout, so you should remove these comments from `hw8-trna.txt` before using it as input (or add this feature to your code if you prefer). Our autograder tests will *not* contain comments. As in hw5, you should read input from stdin; use a command line redirect (e.g., "`hw8.py < hw8-trna.txt`") to read from a file, rather than explicitly dealing with file names in your code.

Revision History:

Rev a: removed "draft" label. — 2/26/22.