CS41 Lab 8

In typical labs this semester, you'll be working on a number of problems in groups of 3-4 students. You will not be handing in solutions; the primary purpose of these labs is to have a low-pressure space to discuss algorithm design. However, it will be common to have some overlap between lab exercises and homework sets.

The learning goals of lab this week are (i) to understand how dynamic programming affects runtime of algorithms in practice, and (ii) to continue to practice building DP algorithm design skills. I encourage you to work on the first problem and then whichever problem looks interesting.

General Hints:

- Focus on the **choice** you might make to construct an optimal solution.
- Initially focus on the first two steps of the dynamic programming process. Don't stress about pseudocode until after you've solved all lab problems.

1. **Testing RNA Substructure Implementations.** Last week, we introduced the RNA Substructure problem and developed an efficient algorithm for RNA Substructure that uses dynamic programming. In this lab problem, you'll see this solution in practice.

   In /home/fontes/public/cs41/, you'll find two executables: `rna-A`, and `rna-B`. One uses dynamic programming to solve the RNA Substructure problem, and one solves it without storing solutions to overlapping subproblems in a table. Each implementation takes in the name of a file containing a single string representing an RNA molecule, and returns the size of the largest matching (following the RNA substructure rules discussed in class).

   For this exercise, you'll use the UNIX `time` command to examine the runtime of each implementation. For example, to measure how much time `rna-A` takes on input `rna_test_data/test1`, execute

   ```
   $ time /home/fontes/public/cs41/rna-A /home/fontes/public/cs41/rna_test_data/test1
   ```

   (a) Using the test files in `rna_test_data` and your own test files, determine which program uses dynamic programming and which does not.

   (b) **How large can inputs be?** For both `rna-A` and `rna-B`, create input files of different sizes and determine how large the input can be if the implementation must run in at most 30 seconds.

   (c) **How does the runtime scale?** Again for each implementation, create some test files of different lengths, and measure the execution time and how it scales with the size of the inputs. Use this to guess what the implementation's runtime is. Is `rna-A` an $O(n^2)$ algorithm? or $O(n^3)$ or $O(n^4)$? $O(2^n)$? Do the same for `rna-B`.

2. **Backpack heist.** You have been recruited as the expert algorithm designer for a team planning an elaborate heist. As part of the heist, the team has scoped out a very fancy mansion full of nefariously-obtained treasures. However, your team will only be able to sneak out a single backpack of items during the heist.

You know that the backpack can hold a maximum weight of $W > 0$, and you have already established the list of n items in the mansion $\{1, \ldots, n\}$ each with nonnegative weight $w_i \geq 0$. Your task is to output a subset of items $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i$ is as large as possible, subject to the overall weight limit $\sum_{i \in S} w_i \leq W$.

Design an analyze a dynamic program to solve this problem. Your algorithm should determine (1) the actual weight of the backpack in the optimal case, and (2) which items to put in the backpack. Your algorithm should run in $O(nW)$ time.

3. **Longest Palindrome.** Let Σ be a finite set called an *alphabet*.[1] A *palindrome* is a string which reads the same backwards and forwards. Let s be a string of characters from Σ and let $x \in \Sigma$ be some character. The reversal of s is denoted s^R. Then the strings ss^R (that is, s concatenated with s^R) and sxs^R are both palindromes.

   In the **Longest Palindrome Problem**, you're given a string s of n characters from Σ and must output the length of the longest palindrome that is a substring of s.

   (a) *Briefly* describe a simple $\Theta(n^3)$ algorithm that solves the longest palindrome problem. Why is your algorithm $\Theta(n^3)$?

   (b) Design an algorithm that uses dynamic programming to solve the longest palindrome problem in less than n^3 time.

[1] For example, Σ might be $\{0, 1\}$ or $\{a, b, c, \ldots, z\}$.