CS41 Lab 8

November 5 2018

The lab this week focuses on dynamic programming. The purpose of this lab is to gain practice using this technique to solve problems. This includes getting some hands-on experience with dynamic programming implementations.

General Hints:

- 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.
- Focus on the **choice** you might make to construct an optimal solution.

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/brody/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/brody/public/cs41/rna-A /home/brody/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 the 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? $O(n^3)$ or $O(n^4)$? $O(2^n)$? Do the same for `rna-B`.

2. **Subset Sum.** Given an integer *weight threshold* $W > 0$ and a list of n items $\{1, \ldots, n\}$ each with nonnegative weight w_i, output a set of items $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i$ is as large as possible, subject to $\sum_{i \in S} w_i \leq W$.

3. **Shortest Paths with Negative Cycles.** Let $G = (V, E)$ be a directed graph with edge costs $\{c_e : e \in E\}$. Earlier in the semester we saw that Dijkstra's Algorithm can be used to compute the length of the shortest path between nodes s and t when the path costs are nonnegative.

Design a dynamic program that produces the minimum cost path between s and t when the edge costs can be negative. You can assume that there are no cycles in G with negative total cost.

4. **Longest Palindrome.** Let Σ be a finite set called an *alphabet*. (For example, Σ can be $\{0, 1\}$ or $\{a, b, c, \ldots, z\}$.)

   A *palindrome* is a string which reads the same backwards and forwards. Let s be a string of characters from Σ and let $c \in \Sigma$ be some character. The reversal of s is denoted s^R. Then the strings ss^R (that is, s concatenated with s^R) and scs^R are both palindromes.

   (a) Give a dynamic programming algorithm that takes a string x of characters from Σ, of length $|x| = n$, and returns the *length* of the longest palindrome contained in x (the longest palindrome that is a substring of x). Your algorithm should run in time asymptotically better than $O(n^3)$.

   (b) Modify your algorithm so that it also returns the longest palindrome in x (not just its length).