CS41 Lab 9

November 2, 2020

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.
- **Note:** To complete problem 1 you will need to access CS Department lab machines. See the Remote Tools guide for instructions on how to remotely log on to department machines.

1. **Shortest Paths with Negative Edge Weights.** In lecture today, we saw the Shortest Paths with Negative Edge Weights (SP-NEW) problem. Here, the inputs are a directed graph $G = (V, E)$ with edge costs $\{c_e\}$, as well as start/end vertices $s, t \in V$. Your goal is to output the cost of the minimum $s \rightsquigarrow t$ path. We assume that G has no negative cycles, but otherwise edges can have negative cost.

   In lecture today, we saw that when edge costs are negative, Dijkstra's algorithm doesn't always compute the minimum-cost path.

   Here is an interesting idea for solving SP-NEW:

   - Given $G = (V, E)$, scan the edges and compute the minimum edge cost: $M := \max_{e \in E} c_e$.
   - For each edge e, let $c'_e := c_e + M$.
   - Run Dijkstra's algortihm on G with the new edge costs $\{c'_e\}$, which are all now nonnegative.
   - Return the distance returned by Dijkstra's algorithm.

   This algorithm doesn't quite work, because there are graphs G such that the minimum-cost path using edge costs $\{c_e\}$ is **not** the minimum-cost path using edge costs $\{c'_e\}$.

   Provide such an example. Your goal is to give an input for SP-NEW such that the minimum-cost $s \rightsquigarrow t$ path is different using edge costs $\{c_e\}$ vs $\{c'_e\}$

2. **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 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.

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 scs^R are both palindromes.

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

   (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.

4. **Gerrymandering (K& T 6.24)** *Gerrymandering* is the practice of carving up electoral districts in very careful ways so as to lead to outcomes that favor a particular political party. Recent court challenges to the practice have argued that through this calculated redistricting, large numbers of voters are being effectively (and *intentionally*) disenfranchised.

   Suppose we have a set of n precincts $P_1, \ldots, P_n$, each containing m registered voters. We're supposed to divide these precincts in to two *districts*, each consisting of $n/2$ precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. Say that the set of precincts is *susceptible* to gerrymandering if it is possible to perform the division into two districts in such a way that the same party holds a majority in both districts.

   Give an algorithm to determine whether a given set of precincts is susceptible to gerrymandering. The running time of your algorithm should be polynomial in n and m.

   For example, suppose there are four precincts, and two political parties A and B. Letting A_i and B_i be the number of voters in precinct i of each political party, Suppose we have

$$A_1 = 55, A_2 = 43, A_3 = 60, A_4 = 47 \textit{ and}$$

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

$$B_1 = 45, B_2 = 57, B_3 = 40, B_4 = 53 \; .$$

This set of precincts is susceptible to gerrymandering since pairing precincts 1 and 4 together and $2, 3$ together gives party A a 102 - 98 majority in the first district and a 103 - 97 majority in the second.

Hint: Focus on the choice you need to make as you're building up a partial solution to this problem (in this case, an assignment of precincts to districts). You will likely need to maintain extra information about the partial solution.