CS41 Lab 7

October 2017

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 *implementing* dynamic programming. This lab includes a coding portion! Feel free to skip between the theoretical (problem 1) and coding (problem 2) parts. Please focus on problems 1–2 (steel rods), and solve them completely before moving on to problems 3–4.

1. **Steel rods problem: theory.** Suppose you own a factory that produces long steel rods. Each rod you produce is n meters long, but it is often more profitable to cut the rod and sell the (shorter) pieces individually. The problem confronting you is: how do you cut the n-meter steel rod into pieces to maximize your revenue?

   - **Input:** n (the length of one rod) and a list (of length n) of prices P where $P[i]$ is the revenue from selling a rod of length i.
   - **Output:** Maximum possible revenue.

   (a) Consider a greedy approach which always chooses the cut k that maximizes the revenue $P[k]$. The idea is to make this cut, sell that rod, and then repeat with the remaining $(n-k)$-meter rod.

   Come up with a counterexample demonstrating that this greedy approach is not optimal.

   (b) Consider a greedy approach which always chooses the cut k that maximizes *revenue per meter* $\frac{P[k]}{k}$. The idea is to make this cut, sell that rod, then repeat with the remaining $(n-k)$-meter rod.

   Come up with a counterexample demonstrating that this greedy approach is not optimal.

   (c) Let's try to use dynamic programming for this problem.

   i. What is the value of the optimal solution?

   ii. What structure of this problem makes dynamic programming a promising approach? Write an expression for the value of the optimal solution in terms of sub-problems. (Hint: try phrasing it in terms of the best place to make the left-most cut.)

   iii. What will you store in an array for a dynamic programming solution to this problem? Make a plan and sketch the algorithm for filling in values of this array.

2. **Steel rods problem: practice.** Instead of writing pseudocode, let's write *actual* code to solve this problem!

   (a) Retrieve the code from github for this lab.

   - `geninput.cpp` generates an input to the steel rods problem. Look over it if you're interested.
   - `brute-force.cpp` is a straightforward brute-force implementation. It iterates over all possible sets of cuts, computes how much revenue is gained, and keeps track of the maximum revenue.

- `cutrod.cpp` is brute-force-ish implementation which uses naive recursive calls: to compute how much revenue can be obtained by cutting at k-feet, make a recursive call on the $(n-k)$-foot rod and add in $P[k]$.

(b) Open `brute-force.cpp`.

i. Look over the code to make sure you understand what it does.

ii. Then, compile the code. e.g. with

```
g++ -o bf brute-force.cpp
```

iii. Run the code on a couple of input sets. Sample usage:

```
./bf < inputs/20.in
```

iv. Now, time this implementation using the UNIX `time` command:

```
time ./bf < inputs/20.in
```

How long does it take to run on a twenty-foot steel rod? ten feet? Try to find an input size which makes the brute-force algorithm take roughly one minute.

(c) Repeat the same steps for `cutrod.cpp`.

How large does the input have to be for this program to take one minute? Which program runs faster?

(d) Implement your own dynamic programming solution to the steel rod problem in the file `dp.cpp`.

i. Compile and run your program on some of the same inputs that you ran the brute force program(s) on. Does your program give the same output? (If not, you have bugs.)

ii. Time your program on different input sizes. How large does the input have to be for your program to take one minute?

3. **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).

4. **SANNdwiches again.** Ann is the proprietor of the wildly successful "Ann's SANNdwich Shop". Her customers are regular and pushy – they order sandwiches at the same time and expect them to be made immediately. She used to have several employees, but they all recently quit to prepare for a pterodactyl hunt. Ann actually doesn't care much about her customers' satisfaction or wait times; she primarily cares about making money, and in her free time she secondarily cares about coming up with terrible puns to rename her shop.

Design and analyze a polynomial-time dynamic programming algorithm to help Ann maximize her profits. The input to your algorithm is a list of n sandwich orders. Each order has:

- a *start time* s_i.
- a *finish time* f_i.
- a *profit* p_i.

Your algorithm should return the maximum amount of profit Ann can make by making sandwiches one at a time.

Hints:

- Focus on the first two steps of the dynamic programming process; don't stress about pseudocode.
- Focus on the **choice** you might make to construct an optimal solution. For example, with the Steel Rod Problem, our choice was where to make the left-most cut.
- Try not to make any sandwich wordplay.