A question for which you'd never guess the correct answer

Question 1 (Micah's cafeteria problem)**.** *The Yankees have made it to the World Series against your favorite team the Houston Astros. The World Series is a best of 7 series which means that the first team to win 4 total games is declared the winner. Thus, the series can be as short as 4 games or as long as 7 games. As an amateur gambler, you plan to place bets on each of the games in the series. Unfortunately, your gambling exploits from the Academy Awards have left you with only $100 in your pocket. While your love for the Astros is unbounded, so too is your enmity for the Yankees. This acrimony has led you to the following decision* If the Yankees win, you want to lose all $100, but if the Astros win, you want to double your money. *What should your strategy be? In particular, how much money should you bet on the first game? Note: There is no probability in this question – your strategy is based purely on the wins and loses of the two teams in the series.*

(a) Start by letting $p(i, j)$ be your current winnings or losings when the Astros have i wins and the Yankees have j wins. For example $p(4, 1) = 100$ because if the Astros you should win $100 dollars while $p(1, 4) = -100$ since if the Astros lose you should lose $100. l In general, what are the base cases for p?

(b) Write a recursive definition for $p(i, j)$.

(c) Now, $p(0, 0)$ should be 0, so $p(1, 0)$ should reveal your bet. What is it?

Greedy Failure

Question 2. *Recall the* matrix-chain order *problem which asks,* given n matrices — $A_1, \ldots, A_n$ with corresponding dimensions $p_0, p_1, \ldots, p_n$ — in what order should we perform the matrix multiplications so as to minimize the number of scalar multiplications? *It is tempting to consider greedy strategies to solve this problem For example, consider the following two greedy algorithms:*

Algorithm 1

> *Given an interval of matrices $A_1, \ldots, A_n$, choose the pair of matrices A_i, A_{i+1} requiring the fewest scalar multiplications. That is, choose i such that $p_{i-1}p_ip_{i+1}$ is minimal. Multiplying these two matrices leaves us with $n - 1$ matrices so recursively apply the strategy on the remaining matrices.*

Algorithm 2

> *Given an interval of matrices $A_i, \ldots, A_j$, choose the split point t such that $p_{i-1}p_tp_j$ is minimal. Use this strategy recursively on the intervals $A_i, \ldots, A_t$ and $A_{t+1}, \ldots, A_j$. The top-level recursion begins with the interval $A_1, \ldots, A_n$.*

For each algorithm, give an example where applying the strategy to the example yields a sub-optimal solution.

Algorithms in the Wild

Question 3 (Derived from K&T 6.6)**.** *In a word processor, the goal of loose justification is to take text with a ragged right margin, like this,*

```
Call me Ishmael.
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text whose right margin is as "even" as possible, like this.

```
Call me Ishmael. Some years ago, never
mind how long precisely, having little
or no money in my purse, and nothing
particular to interest me on shore, I
thought I would sail about a little
and see the watery part of the world.
```

To make this precise enough for us to start thinking about how to write a justifier for text, we need to figure out what it means for the right margins to be "even." So suppose our text consists of a sequence of words*, $W = \{w_1, w_2, \ldots, w_n\}$ where w_i consists of c_i characters. We have a maximum line length of L. We will assume we have a fixed-width font.*

A formatting of W *consists of a partition of the words in W into* lines*. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have*

$$c_k + \sum_{i=j}^{k-1} (c_i + 1) \leq L.$$

We will call an assignment of words to a line valid *if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the* slack *of the line—that is, the number of spaces left at the right margin.*

(a) Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the squares *of the slacks of all lines (including the last line) is minimized.*

(b) Why did we use the sum of the squares *instead of just, say, the sum above? That is, what sort of bias does this optimization function create?*

Question 4. *I found this homework:*

(a) Winning! I am down with DP. It's my favorite problem-solving technique ever.

(b) Good. The problems were fun and I liked the coding. I spent under 10 hours on this assignment.

(c) Okay. I see how DP might be useful. But I spent way too much time on this problem set. Cut me a little slack.

(d) Tepid. Spring break here I come.

(e) Awful. C'mon Brent, this class sucks like Sheen. Teach me something useful.

Extra Credit

Question 5 (Implementing Justification)**.** *Write a program Python that takes two arguments: (1) an integer representing the maximum line length; and (2) a file name, and outputs a pretty-printed version to* `stdout` *using the algorithm above. Your program should be called* `justify` *so if* `sonnets.txt` *is a file of my personal poetry, then*

```
$ justify 20 sonnets.txt
```

should pretty-print out the poetry to the screen. PYTHON *is a great language and useful for many quick and dirty programming tasks. I have prepared a skeleton program from which you can start. It is available on the course website. Use* `turnin` *to turn in your single-file source code. Please ask for help if you need it.*

Question 6 (Generalized Huffman Codes)**.** *Suppose you are given an alphabet $\Sigma = \{a_1, \ldots, a_t\}$ of t symbols. A word $w = u_1 u_2 \cdots u_l$ is a finite sequence of (possibly repeated) symbols from Σ. A code is a set of words $C = \{w_1, w_1, \ldots, w_n\}$. A code is prefix-free if no word in C is a prefix of another word in C. Any code of this form can be expressed as a tree T where a root-to-leaf path in T yields a word in C. If the cost of character a_i is c_i then the cost of a word $w = a_{j_1} a_{j_2} a_{j_m}$ is*

$$c(w) = \sum_{i=1}^{m} c_{j_i}.$$

If codeword w_i has associated probability p_i, then the cost of a code C is

$$\sum_{w_i \in C} c(w_i) p_i.$$

In the standard Huffman coding problem you are given a discrete probability distribution with n values $\mathcal{P} = p_1, \ldots, p_n$ and asked to find a minimum cost prefix-free code for $\mathcal{P}$ over the alphabet $\Sigma = \{0, 1\}$ where $c(0) = c(1) = 1$. In this case the algorithm that greedily builds a binary tree by always combining the pair of values with lowest probability yields an optimal solution. However, the greedy algorithm does not work when the costs of the encoding symbols are not equal. The generalized Huffman coding problem with unequal, integer symbols costs *asks for a minimum-cost prefix free code for $\mathcal{P}$ over an alphabet of size t where $c(a_i) \in \mathbb{Z}^+$ for all $a_i \in \Sigma$. Develop a dynamic programming algorithm for this problem that runs in $O(n^{\alpha+2})$ time where $\alpha = \max\{c(a_i) \mid a_i \in \Sigma\}$.* **Hint: think about associating a length with each edge in the tree. Can you grow a tree and express its cost in terms of the number of leaves it currently has, as well as the *height* of every non-leaf path?**