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 involved 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 (that is, the amount gained or lost relative to your starting $100) when the Astros have i wins and the Yankees have j wins. For example, $p(4, 1) = 100$, because if the Astros win, you should win $100, while $p(1, 4) = -100$ since if the Yankees win you should lose $100. 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?

Question 2. Suppose we have a set $S = \{x_1, x_2, \ldots, x_n\}$ of n positive integers, and let M be a positive integer. The goal is to find the subset of S with the largest possible sum, subject to the constraint that the sum must be $\leq M$.

(a) Describe a simple brute-force algorithm for solving this problem. What is the running time of the algorithm?

(b) Using dynamic programming, describe an algorithm with running time $\Theta(nM)$. Be sure that you explain how to find not only the maximum possible sum, but also the actual subset which has that sum. Justify the correctness of your algorithm.

2

(c) This is known as a *psuedopolynomial-time* algorithm: the running time is a polynomial in the *value* of M, but actually exponential in terms of the *size* of the input (*i.e.* the number of bits needed to represent M).

Give one example of a set of inputs for which your dynamic programming solution would be faster, and one example of a set of inputs for which the brute force algorithm would be faster.

Question 3 (K&T 6.3)**.** Let $G = (V, E)$ be a directed, unweighted graph with nodes $v_1, \ldots, v_n$. We say that G is an *ordered graph* if it has the following properties:

(i) Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.

(ii) Every node other than v_n has at least one outgoing edge.

Given an ordered graph G, we want to find the *longest* path from v_1 to v_n.

(a) Consider the following greedy algorithm.

Algorithm 1: X

```
w ← v_1
L ← 0
while w ≠ v_n do
    Choose the outgoing edge (w, v_j) with the smallest j
    w ← v_j
    Increment L
end while
return L
```

Show that this algorithm does *not* correctly solve the problem, by giving an example of an ordered graph for which it does not return the correct answer. Be sure to explain what the correct answer is and what incorrect answer is returned by the algorithm.

(b) Give an efficient algorithm that takes an ordered graph G and returns the length of the longest path from v_1 to v_n. Justify its correctness and analyze its time complexity.

(c) Explain how to modify your algorithm so that it can also be used to recover the longest path itself, rather than only its length.

Question 4 (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". 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, so we just need to make sure that the number of characters on each line is no more than L.

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 remaining at the right margin. For example, suppose $L = 10$. Then

```
Call me Ishmael.
```

is not valid, since it has length $(4+1)+(2+1)+8$ which is greater than 10. On the other hand,

```
Call me
```

is valid, and has a slack of 3, since it has length only 7, leaving 3 remaining spaces at the end.

We will say that a formatting is optimal when the sum of the *squares* of the slacks of all lines (including the last line) is minimized.

4

Medium hint

(a) Describe a greedy algorithm to find a formatting of a list of words, and give an example where your greedy algorithm does *not* produce an optimal solution.

(b) Using dynamic programming, design and analyze an efficient algorithm to find an optimal formatting of a set of words W into valid lines for a given line length L. (As usual, "analyze" means to prove it is correct, and analyze its asymptotic running time.)

4

Big hint

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

Question 5. (**Optional/Extra Credit**)

If you want some practice actually implementing a dynamic programming solution to a problem, write a program that implements your algorithm from Question 4. Your program should take two command-line arguments: (1) an integer representing the maximum line length; and (2) a file name. It should then output a justified version of the file to `stdout` using the algorithm above.

For example, suppose `lorem.txt` contains the text:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque
rhoncus interdum odio, mattis finibus eros imperdiet non. Praesent egestas lectus.
```

Then running your program with the arguments `25` and `lorem.txt` should print

```
Lorem ipsum dolor sit
amet, consectetur
adipiscing elit. Quisque
rhoncus interdum
odio, mattis finibus
eros imperdiet non.
Praesent egestas lectus.
```

which is an optimal formatting of the text into lines of length at most 25.

Also available is a file `neruda.tgz` which contains a Pablo Neruda poem together with two outputs: `neruda.50.out` and `neruda.30.out` are the results of running my solution on `neruda` using a line length of 50 and 30, respectively. Note that in both cases, there are multiple correct solutions with the same minimum score. Your program may not produce exactly the same output as mine, but you should ensure that it produces a solution with the same score.