Introduction to Algorithms	Problem Set 3
CS 482 Spring 2000	Due February 18, 2000

Some Comments. The following problems can all be solved by a dynamic programming approach. In writing up dynamic programming algorithms, all the guidelines from earlier in the course apply; but there are some additional things worth keeping in mind as well. If your solution consists of a dynamic programming algorithm, you must clearly specify the set of sub-problems you are using, and the recurrence you are using — describing what they mean in English as well as any notation you define. You must also explain why your recurrence leads to the correct solution of the sub-problems — this is the heart of a correctness proof for a dynamic programming algorithm. Finally, you should describe the complete algorithm that makes use of the recurrence and sub-problems.

A description consisting of a piece of pseudo-code with these explanations is not a valid answer.

(1) As we all know, there are many sunny days in Ithaca; but this year, as it happens, the spring ROTC picnic has fallen on rainy day. The ranking officer decides to postpone the picnic, and must notify everyone by phone. Here is the mechanism she uses to do this.

Each ROTC person on campus except the ranking officer reports to a unique *superior officer*. Thus, the reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has as a parent node u equal to his or her superior officer. Conversely, we will call v a *direct subordinate* of u. See Figure 1, in which A is the ranking officer, B and D are the direct subordinates of A, and C is the direct subordinate of B.

To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates one at a time. The process continues this way, until everyone has been notified. Note that each person in this process can only call direct subordinates on the phone; for example, in Figure 1, A would not be allowed to call C.

Now, we can picture this process as being divided into *rounds*: In one *round*, each person who has already learned of the postponement can call one of his or her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates. For example, in Figure 1, it will take only two rounds if A starts by calling B, but it will take three rounds if A starts by calling D.

Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

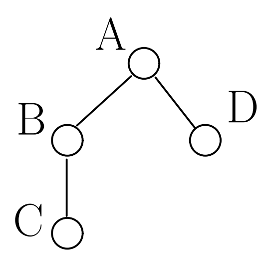

Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

(2) In a word processor, the goal of "pretty-printing" 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 pretty-printer 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, and ignore issues of punctuation or hyphenation.

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 but the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

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

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 — it's the number of spaces left at the right margin.

Give an efficient 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.

(3) You're consulting for a group of people whose jobs consist of monitoring and analyzing electronic signals coming from ships in coastal Atlantic waters. They want a fast algorithm for a basic primitive that arises frequently: "untangling" a superposition of two known signals. Specifically, they're picturing a situation in which each of two ships is emitting a short sequence of 0's and 1's over and over, and they want to make sure that the signal they're hearing is simply an *interleaving* of these two emissions, with nothing extra added in.

This describes the whole problem; we can make it a little more explicit as follows. Given a string x consisting of 0's and 1's, we write x^k to denote k copies of x concatenated together. We say that a string x' is a *repetition* of x if it is a prefix of x^k for some number k. So $x' = 10110110110$ is a prefix of $x = 101$.

We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) subsequences s' and s'', so that s' is a repetition of x and s'' is a repetition of y. (So each symbol in s must belong to exactly one of s' or s''.) For example, if $x = 101$ and $y = 00$, then $s = 100010101$ is an interleaving of x and y, since characters 1,2,5,7,8,9 form 101101 — a repetition of x — and the remaining characters 3,4,6 form 000 — a repetition of y.

In terms of our application, x and y are the repeating sequences from the two ships, and s is the signal we're listening to: we want to make sure it "unravels" into simple repetitions of x and y. Give an efficient algorithm that takes strings s, x, and y, and decides if s is an interleaving of x and y.