Introduction to Algorithms	Problem Set 3
CS 482 Spring 2004	Due February 13, 2004

Please hand in each problem on a separate sheet with your name and NetID on each.

Prelim 1 is scheduled for Thursday February 26th, starting at 7:30 p.m. The prelim will roughly cover the material from the first 3 problem sets (Chapters 1, 2, and 3).

Reading: Sections 3.1-3.4, 3.6, and 3.7.

Some Comments on Dynamic Programming Problems. 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 without these explanations is not a valid answer.

(1) *(This is Problem 25 at the end of Chapter 3.)* As we all know, there are many sunny days in Ithaca, NY; but this year, as it happens, the spring ROTC picnic at Cornell has fallen on a 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.

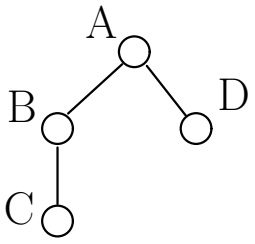

Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

(2) *(This is Problem 26 at the end of Chapter 3.)* 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 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.

(3) *Gerrymandering*, which has been in the news again lately, 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 re-districting, large numbers of voters are being effectively (and intentionally) disenfranchised.

Computers, it turns out, have been implicated as some of the main "villains" in much of the news coverage on this topic: it is only thanks to powerful software that gerrymandering grew from an activity carried out by a bunch of people with maps, pencil, and paper into the industrial-strength process that it is today. Why is gerrymandering a computational problem? Partly it's the database issues involved in tracking voter demographics down to the level of individual streets and houses; and partly it's the algorithmic issues involved in grouping voters into districts. Let's think a bit about what these latter issues look like.

Suppose we have a set of n *precincts* $P_1, P_2, \ldots, P_n$, each containing m registered voters. We're supposed to divide these precincts into two *districts*, each consisting of $n/2$ of the precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. (Suppose for simplicity that every voter is registered to one of these two.) We'll 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.

Example: Suppose we have $n = 4$ precincts, and the following information on registered voters.

precinct	Number registered for Party A	Number registered for Party B
1	55	45
2	43	57
3	60	40
4	47	53

This set of precincts is susceptible, since if we grouped precincts 1 and 4 into one district, and precincts 2 and 3 into the other, then party A would have a majority in both districts. (Presumably, the "we" who are doing the grouping here are members of Party A.) This example is a quick illustration of the basic unfairness in gerrymandering: although party A holds only a slim majority in the overall population (205 to 195), it ends up with a majority in not one but both districts.