Discrete Math in CS	**Problem Set 2**
CS 280 Fall 2005	**Due September 16, 2005**

Reading: Rosen, Sections 3.3-3.5.

Note: The questions below present claims and ask you to provide proofs. Each of these can be proved by induction. For each induction proof, you should clearly state:

(1) What you're trying to prove in the form $\forall n : P(n)$. You should say explicitly what $P(n)$ is. (Also, note that when the problem statement uses phrases like "for all $n \geq 1$," it's assumed that n is an integer unless otherwise specified.)

(2) A proof of the basis, specifying what $P(1)$ is and how you're proving it. (Also note any additional basis statements you choose to prove directly, like $P(2)$, $P(3)$, and so forth.)

(3) The induction hypothesis.

(4) A proof of the induction step, starting with the induction hypothesis and showing all the steps you use. This part of the proof should include an explicit statement of where you use the induction hypothesis. (If you find that you're not using the induction hypothesis at all, it's generally a warning that something is going wrong with the proof itself.)

In addition to correctness, solutions involving proofs or explanations will be graded on style and clarity.

---

(1) Many of you have the seen the function $n!$ (n factorial) from calculus or intro computer science; it is the product of all the natural numbers from 1 through n. Alternately, we can view it as having the following recursive definition:

(i) $1! = 1$.

(ii) For $n \geq 1$, $(n+1)! = (n+1) \cdot n!$.

The factorial function shows up in many identities involving summations, and here we discuss one of these that can be proved directly using induction.

For each natural number $n \geq 1$, define

$$f(n) = \sum_{i=1}^{n} \frac{i}{(i+1)!} = \frac{1}{2!} + \frac{2}{3!} + \cdots + \frac{n}{(n+1)!}.$$

Prove that for all $n \geq 1$, we have $f(n) = 1 - \dfrac{1}{(n+1)!}$.

(2) If you examine a sequence consisting of the first few powers of 6, you notice that they all end in the digit '6' — namely, $6, 36, 216, 1296, ...$

Prove that this pattern persists indefinitely. That is, prove that for all $n \geq 1$, the last digit of 6^n is a '6.'

(*Note:* If we unwind what it means for a number to end in '6,' we see that the problem is asking you to show that for all $n \geq 1$, the number 6^n can be written in the form $10k + 6$ for some integer k. So your induction proof, somewhere in it, should show that 6^n can be written in this form, for a suitably chosen value of k.)

(3) Suppose we have a function $g(n)$ defined on the natural numbers that has the following two properties.

(i) $g(1) = 2$; and

(ii) For all natural numbers $n \geq 1$, we have $g(n + 1) = 2g(n) - 1$.

If we try out the first few values of $g(\cdot)$, we see that $g(1), g(2), g(3), g(4), g(5)$ are $2, 3, 5, 9, 17$ respectively. Notice that this sequence consists of the powers of two, with one added to each: $1 + 1, 2 + 1, 4 + 1, 8 + 1, 16 + 1$. This too is a pattern that we can try verifying for all n.

Prove that for all $n \geq 1$, we have $g(n) = 2^{n-1} + 1$.

---

(4) Computing applications involving graphics and image processing rely on methods for *hidden-surface removal*: given a description of a scene with many objects, and a particular "vantage point" from which you're viewing the scene, the application needs to determine which parts of the scene are visible to you, and which are occluded by other objects. For example, if you're controlling a character in a video game, then as you move around, the software needs to keep track of what's coming into view and what's moving out of view at all times.

This type of problem is an important topic of study in the field of *computational geometry*, and a lot of geometric reasoning has gone into the design of fast algorithms for these problems. Here we explore a very basic example of the kind of analysis that gets used here.

Suppose we are given a collection of n horizontal line segments in the plane. We'll assume that the y-coordinates of all the segments are distinct from one another (i.e. they're all at different "heights"). For simplicity, let's suppose that each line segment has a different *color*. We define the *upper envelope* of the collection to be the sequence of colors that you see as you look vertically downward from above all of them, scanning from left to right. Note that a color can occur more than once in the sequence. For example, consider the set of line segments with "colors" A, B, C, D, and E in Figure 1. The upper envelope would be $A, B, C, D, E, D, C, B, A$.

We define the *complexity* of the upper envelope to be its length as a sequence. So the upper envelope in Figure 1 has complexity 9.

A basic problem is to determine how large the complexity of the upper envelope can get to be, given n horizontal line segments. If we consider a more general version of the example

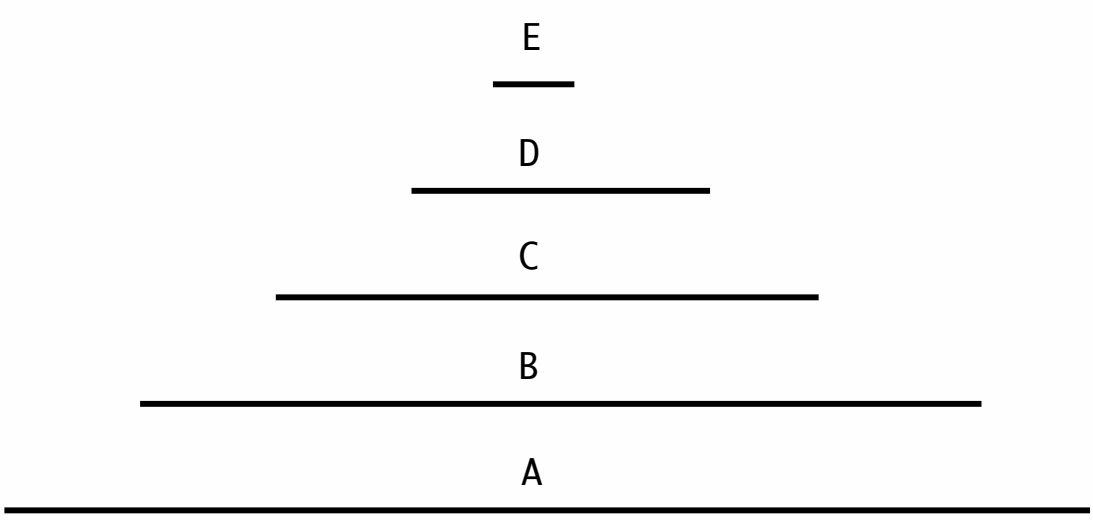

Figure 1: A collection of five line segments with upper envelope $A, B, C, D, E, D, C, B, A$.

in Figure 1, in which we have n line segments, each nested atop the one below like layers on a cake, then it's not hard to convince yourself that the upper envelope has complexity $2n - 1$. (Note that Figure 1 shows the case of $n = 5$, where $2n - 1 = 9$.)

(a) One way to try giving a bound on the complexity of the upper envelope would be to bound how many times a single color can appear. Consider, for example, the following claim.

> *Claim: There is a natural number c, so that for all collections of horizontal line segments (each at a distinct y-coordinate), each color appears at most c times.*

(Note the quantifiers in this claim: it asserts that a single value of c works for all collections of line segments.)

Show that this claim is false, by giving a proof of its negation.

(b) Despite the failure of the claim in (a), it is possible to give a good bound on the complexity of the upper envelope of an arbitrary collection of n line segments.

Prove that for all $n \geq 1$, the upper envelope of any collection of n horizontal line segments, each at a distinct y-coordinate, has complexity at most $2n - 1$.

(Note that this shows the example above to have the maximum complexity possible.)

Final comment: While it's far beyond the scope of this question, it's interesting to ask what happens if we make the problem a bit more general. It turns out that for a collection of n arbitrary line segments, which need not be horizontal and which may cross one another, determining the maximum possible complexity of the upper envelope is a monstrously difficult problem. The short answer is that after years of concerted effort by researchers in

computational geometry, the maximum possible complexity for n segments was shown to be proportional to $n \cdot \alpha(n)$, where $\alpha(n)$ is the inverse of the extremely rapidly-growing function $A(n, n)$ defined on page 273 of the book. (The book doesn't discuss this problem, just the function $A(n, n)$.)

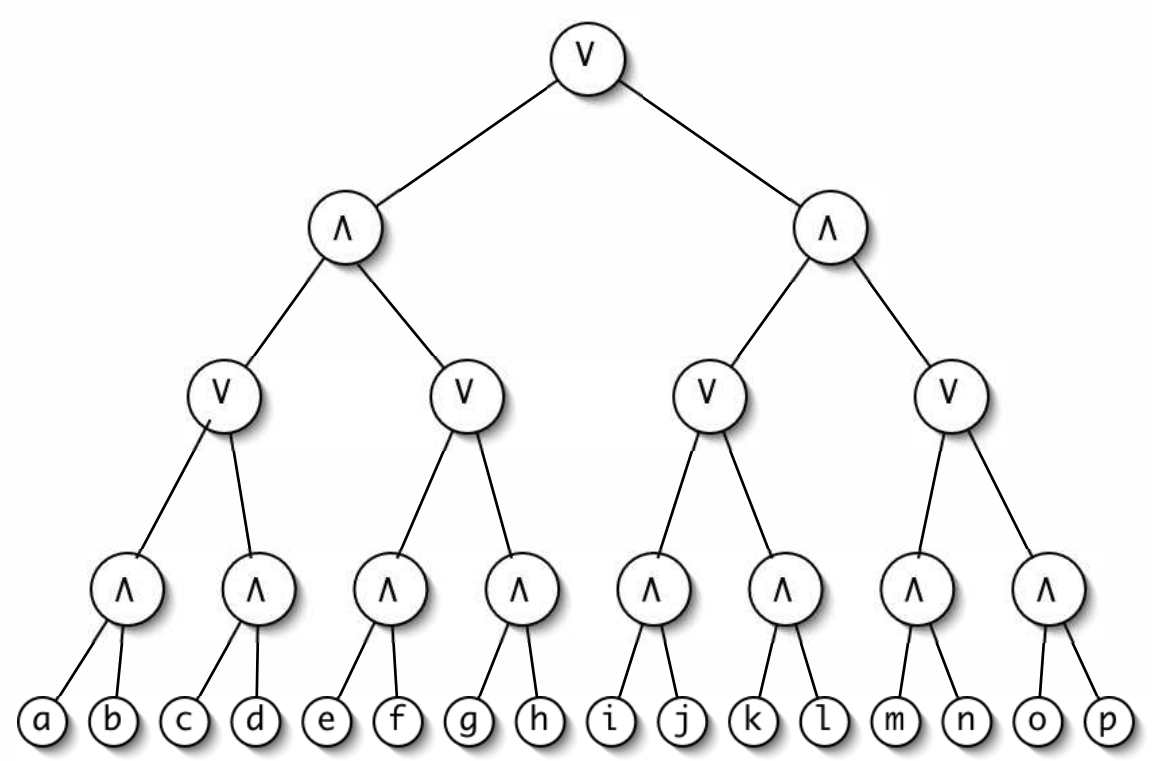

Figure 2: The Boolean formula F_2: a complete binary tree with alternating $\vee$ and $\wedge$ at each level, and $4^2 = 16$ leaves containing variables.

(5) Consider a Boolean formula defined as follows. We pick a natural number k, and we first construct a complete binary tree with $2k$ levels below the root. That is, we start with a root, construct two children of the root, construct two children for each of these children, and so on for $2k$ levels in total below the root. Notice that such a tree has $2^{2k} = 4^k$ leaves.

We now turn this into a Boolean formula as follows. We label the root with $\vee$, label its children $\wedge$, label their children $\vee$, and continue alternating between $\vee$ and $\wedge$ as we move down the tree. Finally, we label each leaf with a different variable. (So there are 4^k variables.)

We call the resulting formula F_k. Figure 2 shows an example of F_2. (Note that F_0 is a formula consisting just of a single variable, with no connectives $\vee$ or $\wedge$.)

The formulas F_k play an important role in the study of game trees, since they represent one of the simplest classes of games: F_k is the tree corresponding to a 2-player game with k moves by each player, and with two options for each player at each move.

An equivalent, recursive, definition of the formulas $F_0, F_1, F_2, \ldots$ is as follows.

- F_0 is a formula consisting just of a single variable.
- For all $k > 0$, the formula F_k can be written $(A \wedge B) \vee (C \wedge D)$, where each of A, B, C, and D is a copy of the formula F_{k-1} on distinct collections of variables. (Notice, for example, how the formula F_2 in Figure 2 is built up from four copies of F_1.)

Here we investigate a basic property of these formulas: the smallest number of variables you need to specify in order to force the value T.

Let $h(k)$ denote the smallest number of specified variables in any partial setting for F_k that forces the value T. That is, there should a partial setting for F_k that forces the value T while specifying $h(k)$ variables, and there should be no partial setting that specifies fewer variables and also forces the value T.

The problem is: Give a formula for $h(k)$ in terms of k, and prove by induction on k that it is correct.

One thing to notice about this problem is that we're not giving you the formula for $h(k)$ and asking you to prove it's correct; instead, we're asking you to come up with the formula. There are a few ways to go about this. One way is to think about F_k for some small values of k and look for a pattern (although F_3 is already quite large). Another way, at least as promising, is to think about what an induction proof would look like, and see if you can discover what formula would be needed to make the induction step work out.