Advanced Algorithms, Fall 2011

Test #3: Solutions

Question 1. Since we want a sequence of indices in increasing order, we can think of scanning down the indices, from 1 to n. As we encounter the next index, i, the only question is whether to include $A[i]$ in the sum so far. Obviously, if including it reduces the best sum found so far, then we do not want to include it; but we also need to take into account the constraint that, if we use $A[i]$, then we could not have used $A[i-1]$ and so must rely on the best sum found 2 iterations back. Still, it appears that we can make a local decision based on just the one index i and its predecessors. Thus we set up an array S with one entry for each n, plus a zero position. The recurrence is simply

$$S[i] = \max\{S[i-1], S[i-2] + A[i]\}$$

Note that the choice is to use $A[i]$, in which case it gets added to the best sum we had two steps back, i.e., $S[i-2]$ (because we cannot use adjacent entries), or not to use it, in which case the best sum we had found at the previous step is also the best sum we can get at this step. We fill up the table position by position, starting with position 2 (because the recurrence needs two initial conditions) and moving on to n. The two base conditions are (i) $S[0] = 0$; and (ii) $S[1] = A[1]$ if $A[1]$ is positive, $S[1] = 0$ otherwise. The DP obviously runs in $\Theta(n)$ time. It returns an optimal solution because (i) it ensures that it never uses consecutive indices in the solution, so that its solution is legal; and (ii) it explores both choices (including or excluding $A[i]$) and selects the better, so that, at step i, it has computed the best ascending, non-consecutive subsequence of elements up to i from which to form the sum.

Question 2. The hint is very important. The entire solution of that homework problem is devoted to a randomized incremental algorithm that runs in expected linear time and decides whether the intersection of a collection of halfplanes is empty. So one should look for a solution to the current problem that involves a collection of halfplanes.

Now, if there is a line that intersects every vertical segment, that line is not vertical and so has an equation of the form $y = mx + k$, for some suitable slope m and intercept k. Let s_i be the ith segment and let its endpoints be (x_i, b_i) and (x_i, t_i), with $t_i > b_i$. Since the line cuts this segment, we must have $b_i \leq mx_i + k$ and $t_i \geq mx_i + k$ (with one of the two inequalities strict)—that is, the line passes above b_i and below t_i, although it might pass through one of the two. This must be true for every segment s_i, so we get a total of $2n$ inequalities. There is a solution (the line exists) if and only if all inequalities can be satisfied by one choice of m and k. But each inequality describes a halfplane in a new space where the two variables are m and k—the unknowns. So there exists a solution (suitable values for m and k) if and only if the intersection of these $2n$ halfplanes is nonempty.

Thus we are done: we need to decide whether the intersection of $2n$ halfplanes is empty by using a randomized incremental algorithm that runs in expected linear time, but that is precisely what the solution of the homework problem gives us.

Question 3. A function such as $f(x_1, x_2, x_3) = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot x_2 + x_2 \cdot x_3$ involves three operations: addition, multiplication, and complement. In the case of binary values (computation over Z_2), we have $\overline{x} = (1 - x) \mod 2$, so that the complement is just a form of subtraction. Thus the formula is literally written in terms of arithmetic, even though, because we expect it to be computed within Z_2, the field of integers modulo 2, we associate it with a Boolean function. But we could carry the same three operations in *any* reasonable arithmetic system—for instance, we could use Z_p, the field of integers modulo some prime p, for any choice of a prime p.

The fingerprinting process as described in the statement of the problem consists of picking some random values for the variables, using these values in each dag to obtain a single output value for each dag, and then comparing these values. So first consider using 0-1 values. The effect is to pick a random truth assignment for the variables; the obvious way to use it in each dag is to compute the value of the function represented by that dag. What we then get is the

value of the function represented by that dag for the chosen truth assignment. Using the same assignment for both dags yields the value of each of the two functions for the assignment; it is thus clear that different values indicate that the dags represent different functions. If the values are the same, however, we have not gained much knowledge: of the 2^n possible truth assignments, we have tested only one and retained only one bit of information. Thus, if we assert on the basis of such a test that the two dags represent the same function, the probability of error can be as large as $1 - 2^{-n}$, and it cannot be improved much by using a polynomial number of tests.

So what went wrong with this simple idea? The problem is loss of information at every step of the evaluation. Consider an internal node with three parents (three incoming arcs); if the values on these incoming arcs are 0, 1, 0, then the value we push farther downward to our children is the sum (the logical or), 1; but then our children can no longer distinguish between the three arcs we actually have and the values they provide, 010, and any other configuration that also results in a 1, such as 111, 001, 100, 110, etc., or even configurations with different numbers of arcs. Information we have computed, and at one point "knew," got lost because we were forced to summarize every computation with a single bit. If, on the other hand, we compute within a larger setting, some Z_p for a sufficiently large prime p, then we retain far more of the information computed during the evaluation on the dag.

A Boolean function is entirely defined by the paths that lead to the leaf labelled 1; each path is a logical "and" (a product); and multiple paths to a node correspond to a logical "or" (a sum). The paths that lead to the leaf labelled 1 correspond to the minterms of the function. (Those that use fewer than n variables are composed of merged minterms.) A Boolean function is uniquely defined by its minterms. But, as we just mentioned, a sum of minterms is thus just an arithmetic function. Computing that function over Z_p gives us the solution. So here is the test algorithm:

1. assign a randomly chosen value v_i in Z_p to variable x_i (how to choose p is discussed below) and the value $1 - v_i$ to the complement of x_i;
2. assign the value 1 to the source node;
3. assign to each arc the value of its associated variable (v_i for the arc labelled 1, $1 - v_i$ for the arc labelled 0);
4. traverse the dag using standard topological sorting order and assign to each node the sum, over all arcs entering the node, of the product of the value of each such arc times the value of the node at that arc's tail; and
5. return the value computed for the leaf labelled 1.

This takes time proportional to the size of the dag and so is very efficient. What is the probability of error—the probability that, if the the two functions are different, a given test still returns the same values for the two dags? We have a choice of p^n different assignments of values to the variables; we claim that, if the two dags represent different functions, at least $(p-1)^n$ different assignments of values will lead to different outputs from the two dags. If that is true, then the probability of error is bounded by $1 - \left(\frac{p-1}{p}\right)^n$ and, by picking $p > 2n$, we ensure that the probability of error is bounded by $1 - \left(1 - \frac{1}{2n}\right)^n < \frac{1}{2}$.

So it remains to show that, if the two dags represent two different functions, at least $(p-1)^n$ of the assignments will yield different outputs in our procedure. This is in fact a direct consequence of what we called in class the "principle of deferred decisions." Suppose the dag has variable x_n at the top and the other $n-1$ variables below; then the function computed by dag G_1 can be written as $G_1 = x_n \cdot G_{1a} + \neg x_n \cdot G_{1b}$, where G_{1a} and G_{1b} are dags on $n-1$ variables. Now, if G_1 and G_2 represent different functions of n variables, then G_{1a} and G_{2a} represent different functions or G_{1b} and G_{2b} represent different functions. Assume the first (the second is symmetric and if both hold, our bounds would be stronger). Then, by inductive hypothesis, we

have that at least $(p-1)^{n-1}$ choices of assignments give different results on G_{1a} and G_{2a}. Since the $n-1$ variables have fixed values, if we still get $G_1 = G_2$ for some value v_n of x_n, that value is unique—it is the solution of the linear equation in one unknown

$$v_n \cdot G_{1a} + (1 - v_n) \cdot G_{1b} = v_n \cdot G_{2a} + (1 - v_n) \cdot G_{2b}$$

So, of the p choices of value for x_n, only one can cause an error and $(p-1)$ of them are safe, which completes the proof.

Question 4. There are, as stated, $|V| + 1$ states, call them s_0 through s_n. s_0 and s_n correspond to the two valid 2-colorings: one is the complement of the other and so, from the perspective of one, every single vertex in the other has the "wrong" color. We do not move from s_0 nor from s_n: if we reach these states, we stop, as there will not remain any monochromatic edge. Elsewhere in the chain, changing the coloring of a vertex must either increase or decrease the number of correctly colored vertices. Moving from state s_i to state s_{i+1} or to state s_{i-1} occurs with equal probability of $\frac{1}{2}$ because, given a monochromatic edge, we know that exactly one of its endpoints is wrongly colored, so that we have a 50-50 chance of picking that vertex and fixing it, but also a 50-50 chance of picking the other, which was correct. Thus we have a symmetric random walk on the line, bounded by two absorbing states.

Now how do we analyze this chain? If both end states (s_0 and s_n) stop the algorithm, then the most distant state from the stopping point is $s_{n/2}$. Since the probability of transition to the left equals that of transition to the right, the expected value will not change as long as the moves remain reversible, that is, as long as we do not reach an end state. The expected number of moves (many back and forth ones) that are needed before our walk can reach an end state can thus be computed by considering the simpler infinite chain (no end state) and computing on this chain the expected number of moves made before deviating $n/2$ states from the mean. This is a typical tail estimate: what is the probability of deviating this far from the mean? Note that the standard deviation of our distribution is a constant and we need to deviate by $\frac{n}{2}$; using a Chebyshev bound thus immediately gives us a probability of $\frac{c}{n^2}$ for some constant c and thus also an expected number of steps to reach state s_n for the first time of n^2/c or $\Theta(n^2)$.

Alternately, we can go through a (more or less) exact computation: if we have come $n/2 - 1$ states away, we can either make one more move and reach the critical state, or we make one move backwards and then are forced to make 2 moves forward. This is a Markov chain—memoryless, so making 2 moves forward obeys the same distribution anywhere along an infinite chain. Thus we want to write a recurrence for the expected number of moves needed to move for the first time from state s_i to state s_{i+1}. Denote this quantity by $M(i)$; then we can write

$$M(i) = 1 + \frac{1}{2}\big(M(i-1) + M(i)\big)$$

because we always have to make one move, but, with probability $\frac{1}{2}$, that move takes us in the opposite direction to state s_{i-1} and then we need $M(i-1)$ additional steps to return to state s_i and another $M(i)$ steps to move to state s_{i+1}. (We can just add these quantities and multiply them by $\frac{1}{2}$ thanks to linearity of expectations.) This is an easy recurrence with solution $M(i) = 2i$. Now, starting from $s_{n/2}$, the expected number of steps we take to reach s_n for the first time is simply

$$\sum_{i=n/2}^{n-1} M(i) = \sum_{i=n/2}^{n-1} 2i$$

and that is $\Theta(n^2)$.