MIDTERM, WINTER 1999 — COMPUTER SCIENCE 308-251

Examiner: Luc Devroye
Date: February 18, 1999, 2:30 pm.

Instructions: Calculators and computers are not allowed.
Answer each question on a separate sheet.
Do your scratchwork elsewhere.
Each answer sheet should have your name/ID.

Time limit: 90 minutes.
Weight: Q1, Q2: 4 credits
Q3, Q4, Q5, Q6: 3 credits

Question 1. Consider the triple Fibonacci sequence defined by $F_{n+3} = F_{n+2} + F_{n+1} + F_n$, with $F_1 = F_2 = F_3 = 1$. Just using integer additions and multiplications, show how you can compute the n-th term F_n when $n = 2^k$, in $\Theta(k)$ (and thus $\Theta(\log n)$) time.

Answer. Well, the first thing to try is to get a matrix recurrence as for ordinary Fibonacci sequences (lecture 1). In particular, note that

$$\begin{vmatrix} F_{n+3} \\ F_{n+2} \\ F_{n+1} \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} \times \begin{vmatrix} F_{n+2} \\ F_{n+1} \\ F_{n+0} \end{vmatrix}, \begin{vmatrix} F_3 \\ F_2 \\ F_1 \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix}.$$

Call the multiplicative matrix A, and note that the vector $(F_{n+3}, F_{n+2}, F_{n+1})'$ is A^n times $(1,1,1)'$. Now A^n is obtained by computing A^2, A^4, ..., A^{2^k} in k matrix multiplications. Finally, $F_n = F_{n+3} - F_{n+2} - F_{n+1}$.

Question 2. Suppose that you are given a sorted sequence of distinct integers $\{a_1, \ldots, a_n\}$. Give an $O(\log n)$ algorithm to determine whether there exists an index i such that $a_i = i$. For example, in $\{-7, 10, 15\}$, there is no such i, but in $\{0, 2, 4\}$, we have $a_2 = 2$.

Answer. Let match (i, j) return true if $a_i = i$ or $a_{i+1} = i + 1$, ..., or $a_j = j$. We call this function with match $(1, n)$.

```
match (i,j)
  if i > j then return false
  if i ≤ j then set m = ⌊(i+j)/2⌋
    if a_m = m then return true
    if a_m > m then return match (i, m−1)
    if a_m < m then return match (m+1, j)
```

Note that the time T_n follows a recurrence as for binary search, and thus $T_n = O(\log n)$.

Question 3. A binary expression tree represents a mathematical expression in the following manner: each internal node has two children and contains a binary operator (such as "*", "/", "+", or "-"); each leaf contains an operand (such as "-15" or "199"). We would like to send the expression tree over a network to someone else, using (close to) the minimal number of bits. Assume that each operator requires 3 bits of storage, and each operand (leaf value) 31 bits. The operands, the operators, and the shape of the tree must be sent. There are n nodes in the tree.

A. Why is n odd?

B. Tell me how you would store the tree as one long string of bits (group the bits, and tell us what they represent, and please make a drawing to explain). Remember that the person at the receiving end must be able to reconstruct the tree.

C. How many bits would this take? (Be careful!)

Answer. If there are k internal nodes (clearly, with two children each), then there must be $k+1$ leaves, and thus $n = 2k+1$ nodes. There are many ways to store the tree. First we list the nodes in preorder and preface every node by a bit, 1 for an internal node, and 0 for a leaf, so that a person at the receiving end knows that a 1 is followed by 3 bits for an operator, and a 0 is followed by 31 bits for a 1. So, there is no boundary or separation problem. Also, the receiver can reconstruct the tree from the preorder sequence with the extra bit of information. The total number of bits sent is $4k + 32(k+1) = 36k + 32$ bits. As $k = \lfloor n/2 \rfloor$, this amounts to about $18n$ bits.

Question 4. Consider a binary game tree with n leaf nodes (where n is an appropriate power of 2). The leaf nodes are all at the same bottom level, and have values that increase from left to right. In the standard alpha-beta search, let T_n denote the time taken by the algorithm on a tree of size n (size being measured by the number of leaves in it). It is unimportant whether we start and/or end with max or min nodes. By considering two levels at a time, derive a recurrence inequality for T_n and solve it. Compare T_n with what you would get with pure minimax search (without the alpha-beta improvement).

Answer. Assume that the root is a min node. Then its four grandchildren have values that increase from left to right. Alpha-beta search needs to visit the three leftmost grandchildren, but will surely not need to visit the fourth one. Thus, $T_n \leq 3T_{n/4} + 1$, where 1 counts the local work done, and $T_{n/4}$ is the work done to compute the value of a grandchild. The master theorem tells us that $T_n = O(n^{\log_4 3})$. Of course, for ordinary minimax, the entire tree is visited. We have $T_n = 2T_{n/2} + 1$, which has solution $T_n = \Theta(n)$. Little tricky point here: n was the number of leaves, not the number of nodes in the tree.

Question 5. Assume that we are performing quicksort on a randomly permuted collection of n different numbers. How many pivots are there on the average that split badly to the right (such a split occurs if the pivot is the smallest element in the set it is supposed to split, and that set is of size at least 2)? Answer must be correct to within 1.

Answer. Note the one-to-one relationship with random binary search trees we explained in class. A node is a pivot if it is not a leaf in the corresponding (random) binary search tree. We saw that the expected number of leaves in a tree with n nodes is $(n+1)/3$, so the expected number of nodes with 2 children is $(n-2)/3$ and the expected number of nodes with 1 child is $(n+1)/3$. On the average, half of those split badly to the right (have only a right child), so the answer is $(n+1)/6$. Answers in the range $[n/6 - 5/6, n/6 + 7/6]$ were accepted.

Question 6. Consider an ordered rooted tree with at most 3 children per node. How would you store it in $O(n)$ space if the only operation needed in the future is "colinear", where colinear (a,b,c) returns true if there is a path starting at node a that visits node b and then ends up at node c without visiting an edge twice (so, you can trace a path with a pencil from a to b to c without using any edge twice). It will be called repeatedly with different values for a, b and c, and its worst-case complexity must be $O(1)$. You may assume that the nodes are pairwise unequal. Do not worry about the time needed to compute whatever it is that you are storing. Discuss your answer.

Answer. Only two situations are good: one of a or c is a descendant of b, or both are descendants of b. In the latter case, the children of b on the paths (ba) and (bc) respectively must be different. So, we need a structure that allows us to answer these questions quickly. Assume that the nodes are numbered 1 through n. We propose to store the preorder and postorder numbers for all nodes in two arrays of size n (preorder(i)=7 means that node i is 7th in preorder traversal). In addition, we keep pointers (possibly nil) to first, second and third children. The total storage requirement is thus

5n. The test "is a an ancestor of b" can be carried out in O(1) time by verifying that preorder(a) $<$ preorder(b) and postorder(a) $>$ postorder(b). Convince yourself that the condition described at the top of this page can be checked in $O(1)$ time.