Written Qualifying Exam
Theory of Computation

Fall, 1996

Friday, September 27, 1996

This is nominally a *three hour* examination, however you will be allowed up to four hours. All questions carry the same weight. There are seven questions. You are to answer the following six questions: the first five questions **and** one of the last two questions.

• Please write your name on the outside envelope, but not on any if the exam booklets.
• Please answer each question in the numbered booklet provided for that question.

Read the questions carefully. Keep your answers brief. Assume standard results, except where asked to prove them.

Problem 1 **[10 points]**
Let U be an ordered collection of weighted items, i.e. $U = \{s_1, s_2, \cdots, s_n\}$, with $s_1 < s_2 < \cdots < s_n$, and where s_i has weight w_i (an integer), for $1 \leq i \leq n$. Describe at a high level a single data structure that supports **all four** of the following operations, all running in time $O(\log n)$.

(1) Insert item s with weight w.

(2) delete item s.

(3) Negate($low, high$): For all items s in the range $[low, high]$ (i.e. $low \leq s \leq high$), change the weight of s from its current value w to $-w$.

(4) Max($low, high$): Report the maximum weight of the items in the range $[low, high]$.

Problem 2 **[10 points]** The standard Tower of Hanoi Problem has three posts named A, B, and C, and a stack of n rings that are initially all on post A. The rings have decreasing diameters, with the smallest ring located at the top of A. The rules for moving the rings are:

a) Only one ring may be moved at a time.

b) A ring can only be moved from the top of one stack to the top of another.

c) A ring can only be moved to an empty post or a post that has rings of larger diameter, so that the the diameters are always increasing down a post.

Consider the problem of moving the n rings from A to B in a way that uses **as many moves as possible** without repeating any configurations (where two configurations are counted as repeated only if each of the n rings is in an identical location for both configurations).

i) 1 pt Compute by hand the largest number of moves for a slowest nonrepeating algorithm for $n = 1$, and $n = 2$.

ii) 4 pts Present the algorithm for n rings, and be sure that it achieves the correct number of configurations for $n = 1$ and $n = 2$.

iii) 2 pts Explain, in a sentence or two, why your recursive solution does not repeat any configurations.

iv) 3 pts Present the recurrence equation for the exact number of ring moves used by your algorithm, and show that your solution uses the maximum number of rings moves without repeating any configuration of the n rings.

Go to the next page

Problem 3 [10 points]

Let $G = (V, E)$ be a directed graph with n vertices named $1, 2, 3, \ldots, n$, and with edges that have the nonnegative weights $Cost(i, j)$, for (i, j) in E. Assume that $Cost(i, j) = \infty$ for (i, j) not in E. In the following path cost problems, it will be sufficient to show how to compute the costs; path recovery is not required.

i) 1 pt Present an $O(n^3)$ time algorithm to solve the All-Pairs-Shortest-Paths problem for such a graph.

ii) 1 pt Let $e_1 = (a, b)$ and $e_2 = (c, d)$ be additional edges that are not in E, but which have vertices in V. We now define a **legal** path from i to j to be any path built from edges in $E \cup \{(a, b)\} \cup \{(c, d)\}$ where either all of the path edges are in G (i.e., neither e_1 nor e_2 is on the path) or both e_1 and e_2 are on the path, and the edges are arranged so that a traversal along the path from i to j will visit edge e_1 before e_2. There is no requirement that a legal path be cycle-free (i.e. simple). Explain how to postprocess the solution to part (i) to solve the All-Pairs-Shortest-Paths problem for **legal** paths for G plus these two new edges. Assume that the costs of the two new edges are the nonnegative values $Xcost(a, b)$ and $Xcost(c, d)$.

iii) 1 pt Now solve the same problem as (ii) where there are the k pairs of additional edges $(a_1, b_1), (c_1, d_1)$; $(a_2, b_2), (c_2, d_2)$; $\ldots$; $(a_k, b_k), (c_k, d_k)$. For this part, a **legal** path either has none of the new edges or contains just one pair (a_h, b_h) and (c_h, d_h) for the same h (plus edges from E) in the order described in part (ii).

iv) 7 pts You are given the same graph G and set of k pairs of additional edges as in (iii). Now a **legal** path is one that has either none of the additional edges or has the pairs arranged in any nested bracket (named parentheses) order, so that, for example, $i, \ldots, a_5, b_5, \ldots, a_2, b_2, \ldots, c_2, d_2, \ldots, c_5, d_5, \ldots, a_1, b_1, \ldots, c_1, d_1, \ldots, j$ would be **legal**, but $i, a_1, b_1, a_2, b_2, c_1, d_1, c_2, d_2, j$ would not be legal because the edge bracketing is not nested. Present an efficient postprocessing algorithm to solve the All-Pairs-Shortest-Paths problem for this restricted **legal** paths problem. Full credit will be awarded to solutions that run in $O(n^3 + k^2n^2)$ time, although faster solutions are possible.

Hints: For part iv) please note that the implicit language of **legal** paths is context free but not regular; hence simple graph adaptations will not work. Use ideas from part (iii) plus the correctness and efficiency schemas of the algorithm given for part (i).

Go to the next page

Problem 4 **[10 points]**

i) 4 pts Consider the alphabet $\Sigma = \{N, D, Q, W\}$. Intuitively, N says "deposit a nickel", D says "deposit a dime", Q says "deposit a quarter", and W says "withdraw fifteen cents". A word $x \in \Sigma^*$ is said to be *valid* if it represents a sequence of instructions that maintains a nonnegative balance. Initially the balance is 0. Hence no valid word can begin with W. For instance, $NDDQW$ is valid. Let $L_0 \subseteq \Sigma^*$ be the language of valid words. Find the exact place of L_0 in the Chomsky Hierarchy (i.e., regular language, CFL, CSL, or recursively enumerable).

ii) 6 pts Suppose that the bank balance cannot exceed fifteen cents. That is, consider the subset $L_1 \subseteq L_0$ representing all sequences whose balance is always between 0 and 15 cents. Give a regular expression that denotes L_1. HINT: You get 70% credit if you only draw the dfa.

Problem 5 **[10 points]**

Classify the following as recursive or r.e. but not recursive or co-r.e. but not recursive or neither r.e. nor co-r.e.

i) 2 pts $L_1 = \{x \mid x$ is a string encoding the adjacency matrix of an undirected graph with a Hamiltonian circuit$\}$.

ii) 3 pts $L_2 = \{M\#x\#q \mid q$ is a reachable state of Turing Machine M on input $x\}$.

iii) 5 pts $L_3 = \{x \mid \exists y \cdot M_x(y)\downarrow$ and $y \in \mathrm{TOT}\}$.
Hint: Show that $\mathrm{TOT} \leq L_3$. ($\mathrm{TOT} = \{w \mid \forall z \cdot M_w(z)\downarrow\}$.)

Problem 6 *Answer* **either** *this problem or problem 7 but not both.* **[10 points]**

State whether the following are true or false, and give brief explanations, which can cite standard theorems as necessary.

i) 3 pts (T/F) NPSpace=NSpace

ii) 3 pts (T/F) Context Sensitive Languages are closed under complementation (i.e. L is a CSL if and only if $\bar{L}$ is a CSL.)

iii) 4 pts (T/F) P $\neq$ Dspace(n^2).

If you select this problem, please circle the 6 on your booklet.

Problem 7 *Answer* **either** *problem 6 or this problem but not both.* **[10 points]**

i) 1 pt Define **RP**, randomized polynomial time.

ii) 2 pts Let r be a free parameter. Show that if $L \in \mathbf{RP}$ then there is a randomized algorithm A and a polynomial $p(n)$, such that on input x of size n, A runs in time $O(r \cdot p(n))$ and:

(i) If $x \in L$ then A accepts x with probability at least $1 - 1/2^r$.

(ii) If $x \notin L$ then A rejects x.

iii) 7 pts Let $L \in$ RP. Show that $L^* \in$ RP.
Hint: $x = x_1 x_2 \cdots x_n \in L^*$, $n \geq 1$, if and only if $x_1 \cdots x_i \in L^*$ and $x_{i+1} \cdots x_n \in L$, for some i, $0 \leq i < n$.

If you select this problem, please circle the 7 on your booklet.

Solutions

Solution to Problem 1 The items s are stored in a 2-3 tree (or any other balanced tree), using the values s as the key. Some additional information is stored at the internal nodes in order to support operations (iii)-(v). At internal node v, the maximum and minimum values in the subtree rooted at v are stored. In addition, a marker value is held; it indicates whether the values in v's subtree are to be negated.

To perform an insertion, a path is traversed to a new item and the marks are *pushed* from the path to the path children: the effect of a negation is to complement the marker at each child and interchange their maximum and minimum values. We say the nodes on the path and the path itself have been *cleaned.* Now the insertion is performed in the standard way.

A deletion is similar; the path to the item to be deleted is cleaned. Now the item is deleted in the standard way, except that when two nodes are combined, or their children shared, the nodes are first cleaned.

The maximum and minimum values are maintained in the natural way as these operations are performed.

Negate is performed as follows. By means of searches for the values *low* and *high*, the $O(\log n)$ subtrees spanning the range $[low, high]$ are identified. These two paths and the roots of these subtrees are cleaned. Then, at the root of each of these subtrees, the marker is set to negative and the maximum and minimum values are interchanged. Finally, on the two paths to the values *low* and *high*, the maximum and minimum values are updated.

Max is performed as follows. Again, by means of searches for the values *low* and *high*, the $O(\log n)$ subtrees spanning the range $[low, high]$ are identified. These two paths and the roots of these subtrees are cleaned. Then, the maximum of the maxima at the roots of these $O(\log n)$ subtrees is reported.

Clearly, each of these operations runs in $O(\log n)$ time.

Solution to Problem 2 i) is implicitly answered in ii) and or iii).
ii) Let $TH(n, A, B, C)$ be the slowest, non-repeating algorithm that moves n rings from A to B. Then the recursive solution is:
procedure $TH(n, A, B, C)$;{ moves n rings from A to B }
    if $n > 1$ then $TH(n-1, A, B, C)$ endif;
    move top ring on A to C;
    if $n > 1$ then $TH(n-1, B, A, C)$ endif;
    move top ring on C to B;
    if $n > 1$ then $TH(n-1, C, B, A)$ endif
end-TH.
iii) The solution does not repeat positions because each of the three recursive calls moves $n-1$ rings with the n-th ring on a different post. Formally, the fact would follow from induction; the base case is clear, and the above argument is exactly what is needed to go from the induction hypothesis for $n-1$ to n.
iv) The recurrence is easy:

$$T(n) = \begin{cases} 2, & \text{if } n = 1; \\ 3T(n-1) + 2, & \text{if } n > 1. \end{cases}$$

There are many ways to see that $T(n)$ is the maximum number of moves. One way is to observe that $T(n) + 1 = 3^n$. Since each ring can be on one of three posts, at any given time, there are 3^n different ways n rings can be assigned to the three posts. As the ordering of a fixed set of rings is determined by the ring diameters, 3^n is the total number of different configurations. Thus there can be at most $3^n - 1$ moves before a position is repeated.

We could also let $S(n)$ be the maximum number of ring moves that an algorithm could use without repeating a configuration. By applying our careful reasoning that gave us the algorithm for TH, we would get

$$S(n) \geq \begin{cases} 2, & \text{if } n = 1; \\ 3S(n-1) + 2, & \text{if } n > 1. \end{cases}$$

Solution to Problem 3 i) Use the Floyd-Warshall algorithm.
initialize $cost[*,*]$ with $Cost(*,*)$
for $k \leftarrow 1$ to n do
    for $i \leftarrow 1$ to n do
        for $j \leftarrow 1$ to n do
            $cost[i,j] \leftarrow \min(cost[i,j], cost[i,k] + cost[k,j])$
endall.
ii)forall i, j do $Ccost[i,j] \leftarrow \min(cost[i,j], cost[i,a] + Xcost[a,b] + cost[b,c] + Xcost[c,d] + cost[d,j])$ endfor.
We must avoid using more than one edge-pair, which forces us to save the data in a new array to preserve the values $cost[i,a]$, $cost[b,c]$, etc. iii) Same as (ii), but take the min over $cost[*,*]$ and all k pairs of a–b, c–d edges.
iv) Answer:
repeat k times
    for $l \leftarrow 1$ to k do
        for $i \leftarrow 1$ to n do
            for $j \leftarrow 1$ to n do
            for $l \leftarrow 1$ to k do
                $cost[i,j] \leftarrow \min(cost[i,j], cost[i,a_l] + Xcost[a_l,b_l] + cost[b_l,c_l] + Xcost[c_l,d_l] + cost[d_l,j])$
endall.
This is FW-like (or Bellman-Ford-like) in the updating. The solution to part (iii) is sure to install the deepest nested bracket where it is needed as a deepest bracket. Because the costs are positive, it cannot be that a path will require two instances of this edge pair, since $\ldots, a, b \ldots a, b \ldots$ can be short-cut, as can $\ldots, a, \ldots, a \ldots, b \ldots, b \ldots$. So the first pass of the k loop will install all (deepest) edge pairs in all path pieces that require just one edge pair. It should be evident that each iteration will make progress unless the solution is already complete. How many iterations are needed? Consider a solution path. Let the nesting depth of a specific instance of an edge pair be the number of pairs that contain the instance. Let the nesting height of a specific pair p be depth of the deepest pair within the subpath restricted to be between the two edges of p. It is not difficult

to see that the height is at most $k-1$, and the h'th iteration of the k-loop correctly computes all path costs for paths where the heights are at most $h-1$.

As in the F-W solution, the ordering of the outer loops matters. A solution path may have to revisit edges to get the shortest path.

Solution to Problem 4 i) If the language L of valid instructions were regular, there would be a dfa M that accepts L. Let M have n states. Consider the word x that consists of a sequence of deposits so that the balance exceed $5n$ cents. Then $xW^{[n]}$ is a valid sequence. Consider the words $xW^{[i]}$ for $i = 0, \ldots, n$, and let q_i denote the state of M after reading $xW^{[i]}$, starting from the initial state. There must be two values $j < k$ such that $q_j = q_k$. This means $xW^{[j]}W^{[m(k-j)]}W^{[n-k]}$ is valid for all $m \geq 0$. This is clearly a contradiction.
To see that the language is context free, we note that the balance can be "counted" by a PDA that pushes a 5-cent token for every five cents deposited, and pops three tokens for every withdrawl. A word is accepted if the PDA never attempts to pop an empty stack.
ii) Notice that a withdrawl can only occur when the balance reaches 15 cents, at which point the balance returns to zero. Thus the language will cycle through zero balances before terminating with a balance of 0, 5, 10, or 15 cents. Quarters can be ignored. It is not hard to see that the following defines these actions perfectly:

$$((N^3 + ND + DN)W)^*(0 + N + N^2 + N^3 + D + DN + ND).$$

Solution to Problem 5
i) There is an exponential time algorithm for testing whether a graph has a Hamiltonian path; consequently this is a decidable property and hence L_1 is recursive.

ii) L_2 is r.e.: simply dovetail the simulations of $M(x)$, checking whether they reach state q, and list those strings $M\#x\#q$ which are discovered to have this property. To see that L_2 is not recursive, it suffices to note that this is essentially the halting problem: to test if $M_y(x)$ halts, simply choose q to be the halting state of M_y and ask if $M\#x\#q \in L_2$.

iii) We construct computable function f such that $x \in \text{TOT}$ if and only if $f(x) \in L_3$.

$$M_{f(x)}(z) = \begin{cases} \downarrow & \text{if } z = x \\ \uparrow & \text{if } z \neq x \end{cases}$$

As TOT is neither r.e. nor co-r.e., the same is true of L_3.

Solution to Problem 6
i) True. By Savich's Theorem, NSpace($f(n)$ $\subseteq$ DSpace(($f(n)^2$), for all well behaved functions $f(n) \geq \log n$, including all polynomials. Consequently, NPSpace $\subseteq$ PSpace.

ii) True. By Immerman's Theorem, NSpace($f(n)$) = Co-NSpace($f(n)$) for all well behaved functions $f(n) \geq \log n$, including $f(n) = n$. By the CSLs are exactly the languages accepted by linear space nondetermistic Turing Machines, i.e. NSpace(n).

iii) True. Suppose for a contradiction that P = DSpace(n^2). By the Space Hierarchy Theorem, DSpace(n^2) $\subset$ DSpace(n^4) and DSpace(n^2) $\neq$ DSpace(n^4). But we will show that any $L \in$ DSpace(n^4) is also in P = DSpace(n^2), which yields the contradiction.

Let $\mathrm{Pad}(L) = \{x\$^i \mid x \in L \text{ and } i = |x|^2 - |x|\}$, where \$ is a character outside L's alphabet. Clearly, $\mathrm{Pad}(L) \in \mathrm{DSpace}(n^2)$ (simply check the input has the right format, and then simulate the $\mathrm{DSpace}(n^4)$ acceptor for L on the initial portion of the input string). Hence $\mathrm{Pad}(L) \in \mathrm{P}$. To accept L in polynomial time, given input x, we first determine $\mathrm{Pad}(x) = x\$^{|x|^2-|x|}$, and then simulate the polynomial time acceptor of $\mathrm{Pad}(L)$. If $\mathrm{Pad}(L)$ is accepted in time $O(n^k)$, $k \geq 1$, then L is accepted in time $O(n^{2k})$, which is also polynomial time.

Solution to Problem 7

i) **RP** is the set of languages L such that L has a randomized polynomial time algorithm $\mathcal{A}$ accepting L, i.e.

(a) if $x \in L$, then $\mathcal{A}$ accepts x with probability at least $1/2$.

(b) If $x \notin L$, then $\mathcal{A}$ rejects x.

ii) Let $L \in \mathbf{RP}$ and let $\mathcal{B}$ be the randomized polynomial time algorithm accepting L. Suppose $\mathcal{B}$ runs in time $p(n)$. Let $\mathcal{A}$ be algorithm $\mathcal{B}$ iterated r times, with A accepting only if $\mathcal{B}$ accepts on every iteration. Clearly $\mathcal{A}$ achieves the desired probability of success.

iii) Choose r to be $2\log n + 1$. We test if $x_i \cdots x_j \in L$ for all i, j, $1 \leq i \leq j \leq n$. There are n^2 strings being tested, and thus with probability at least one half all the reported results are correct; in any event all positive results are correct. Clearly each test runs in polynomial time, and thus the collection of tests requires polynomial time too. Now, by means of dynamic programming, we determine which strings $x_1 \cdots x_h \in L^*$: this takes time $O(n)$ per index h, or time $O(n^2)$ altogether. Thus the overall running time is polynomial. If the negative results of tests of membership in L were all correct, the same is true of negative tests of of membership in L^*; as this occurs with probability at least $1/2$, we conclude that if $x \in L^*$ then x is accepted with probability at least $1/2$. And of course, if $x \notin L^*$ then this is correctly reported (for any incorrect tests of membership in L only result in fewer strings being recognized as being in L^*).