Written Qualifying Exam
Theory of Computation

Spring, 1996

Friday, May 24, 1996

This is nominally a *three hour* examination, however you will be allowed up to four hours. All questions carry the same weight. Answer all six questions.

• Please check to see that your name and address are correct as printed on your blue-card.
• Please *print* your exam number but not your name on each exam booklet. Write your name and number on the envelope, however. Answer each question in a *separate* booklet, and *number* each booklet according to the question.

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

Problem 1 [10 points](*new booklet please*)

The Augmented Towers of Hanoi problem is the following. You are given four vertical posts, A, B, C and D. There are a stack of n rings on post A, and as in the standard problem, the each ring sits on top of a larger ring, so that the largest is at the bottom. As in the standard problem, you can only move one ring at a time, and it must be the highest in its stack. Furthermore, a ring can never be placed on top of a smaller ring. Posts A, B, and C can hold n rings, but post D can only hold one ring.

a) (4 points.) Present, at a high level, as efficient an algorithm as possible to move the stack from A to B.
Hint: from a recursive viewpoint, the use of D would seem most reasonable for either the smallest ring or perhaps the second largest. Indeed, one of these two possibilities does give the best possible algorithm.

b) (1 point.) Write the recurrence equation for the running time of your algorithm.

c) (5 points.) Give an informal but complete argument to show that your solution is the best possible.
Hint: recall how the proof works for the standard problem. At the time the bottom ring is moved to B, the $n-1$ other rings have been moved elsewhere, and sit on one post. Thus a $T(n-1)$ must have been performed, and a 1 move for the bottom ring as well. To complete the problem, another $T(n-1)$ must be performed. Here T represents both the most efficient algorithm and its running time.

Problem 2 **[10 points]**(*new booklet please*)

Let us define a *path unique* graph $G = (V, E)$ to be a directed graph with the following property: for each pair of vertices a and b, there is at most one cycle-free path from a to b.

The property allows cycles to be present in the graph, but requires a path from a to b to be unique, unless it has cycles. Needless to say, the definition also allows cycles to have cycles.

The problem is as follows.
Let $G = (V, E)$ be a path unique graph. Give an $O(|V| + |E|)$ time algorithm to compute, for each of the vertices v in V, the number of vertices reachable from v.

For simplicity, we may say that v is reachable from itself, so that if G were a tree, then the leaves would receive the count 1, and the root would receive the count $|V|$.

Hint: consider the sets that can reach the same vertex groups.

Problem 3 **[10 points]**(*new booklet please*)

For the following problem, you may use any standard algorithm or algorithmic method without presenting it in any detail. Simple algorithmic descriptions will suffice for each part.

Let $W = (w_1, w_2, \ldots, w_n)$, $X = (x_1, x_2, \ldots, x_n)$, $Y = (y_1, y_2, \ldots, y_n)$, $Z = (z_1, z_2, \ldots, z_n)$ be four sorted sets of of positive numbers, so that $0 < x_1 \leq x_2, \ldots$, etc.

a) Give a $\Theta(n)$ time algorithm to answer the query: Given the value q, does $q = w_i + x_j$ for some pair (i, j)?

b) Give a $\Theta(n^2 \log n)$ time algorithm to answer the query: Given the value q, does $q = w_i + x_j + y_k$ for some tuple (i, j, k)?

c) Give a $\Theta(n^2 \log n)$ time algorithm to answer the query: Given the value q, does $q = w_i + x_j + y_k + z_l$ for some tuple (i, j, k, l)?
Hint: Try reducing this problem to one of type a).

Go to the next page

Problem 4 **[10 points]**(*new booklet please*)

Let L be a language over the alphabet Σ with the following *pumping* property. There is an integer n, such that for every $x \in \Sigma^*$, $|x| \geq n$, x can be written as $x = uvw$, with $|v| \geq 1$ and

$$\forall y \in \Sigma^* : \; uvwy \in L \Longleftrightarrow uwy \in L$$

a. (5 points.) Show that L is regular. Hint: Show that L has a finite spanning set.

b. (5 points.) Conversely, show that if L is regular, then it has the pumping property.

Problem 5 **[10 points]**(*new booklet please*)
Let f be a computable function. Suppose L_1 is recursive. Inductively define $L_{i+1} = \{x \mid f(x) \in L_i\}$.

a. (5 points.) Show that $L = \cup_{i \geq 1} L_i$ is r.e.

b. (5 points.) Show that L need not be recursive. Hint: Let x be a string representing a configuration of some Turing Machine M. What should be in the recursive set L_1? Define f and M so that L is not recursive.

Problem 6 **[10 points]**(*new booklet please*)
Suppose that there is a polynomial time computable function f which on input F, an n clause 3-CNF formula, returns $f(F) = k$, where $1 \leq k \leq n$ is an integer. Further, suppose that f is *satisfiability consistent*, i.e. $f(F) = f(G)$ implies both F and G are satisfiable or neither F nor G is satisfiable.

Show that if f exists, 3-SAT $\in P$.
Hint. Consider the various simpler formulae G that arise in determining the satisfiability of F (formulae G are obtained by substituting in values True and False for some of the variables in F). Use the values $f(G)$ to limit the number of formulae whose satisfiability needs to be determined.

Solutions

Solution to Problem 1

a)

> **procedure** ATH(n,A,B,C,D);
>   **if** $n = 2$ **then**
>   Move top ring on A to D;
>   Move top ring on A to B;
>   Move top ring on D to B
>   **elseif** $n = 1$ **then** Move top ring on A to B
>   **else**
>   ATH$(n-2, A, C, B, D)$;
>   ATH$(2, A, B, C, D)$;
>   ATH$(n-2, C, B, A, D)$
>   **endif**
> **end**_ATH.

b) $T(1) = 1$; $T(2) = 3$;

$T(n) = 2T(n-2) + 3$, $n > 2$.

c) Again consider what happens when ring n is moved. All $n-1$ rings must be elsewhere. Hence at least $n-2$ are on some post that is not A, and another ring has also been moved to some post other that A. When ring n lands on B, there is a stack of at least $n-2$ rings that must be moved to B, and another ring also.

So if T is the number of moves for the most efficient solution, we see that $T(n) \geq 2T(n-2) + 1 + 1 + 1$, where the $2T(n-2)$ counts the moves for two stacks of $n-2$ rings, a $1+1$ counts the moves for the "other" ring (even if it is on the stack with the other $n-2$ rings), and the other 1 is for moving ring n.

The initial conditions $T(1) = 1$, $T(2) = 3$ are not interesting.

Solution to Problem 2

The answer to the hint is that vertices in the same strong component have the same reachability sets.

The simplest solution is to preprocess the graph to identify the strong components, construct the reduced graph where each strong component is represented by a single vertex, and give each such vertex a count that is equal to the number of vertices in the strong component. This reduced structure is a DAG with unique paths, with means that the graph is structured (if, say we ignore edge directions) like a collection of trees. We may now use a standard DFS with postorder processing to compute the (partial) sums of the weights.

A simpler program would result from reversing these two steps: first use a postorder DFS to compute (sometimes incorrect) reachablity counts, where no back edges are used.

It is easy to see that the first vertex (reached by the DFS) in each strong component would get the correct counts. Now Tarjan's SC algorithm can be used to spread these counts to the other members of each strong component. This way, we need not construct an auxiliary graph.

Solution to Problem 3

a) Use a linear scan of the indices (conditionally decrementing one cursor and incrementing the other) to answer the query. In (unnecessary) pseudocode, we might write the following.

> $i \leftarrow 1$;
> $j \leftarrow n$;
> $found \leftarrow$ `FALSE`;
> **while** $i \le n$ `AND` $j \ge 1$ `AND NOT` $found$ **do**
>   **if** $q = w_i + x_j$ **then** $found \leftarrow$ `TRUE`
>   **elseif** $q < w_i + x_j$ **then** $i \leftarrow i + 1$
>   **else** $j \leftarrow j - 1$
>   **endif**
> **endwhile**;
> **return**($found$).

b) For each i, j pair, use binary search to solve:

Does $q - w_i - x_j = y_k$, for some k?

Here the time is $\Theta(n^2 \log n)$.

A more efficient solution can be found by reducing b) to a).

For each i, use a) to solve:

Does $q - w_i = x_j + y_k$, for some j and k?

Here the time is $\Theta(n^2)$.

c) Form the sets $A = \{w + x : w \in W, x \in X\}$, and $B = \{y + z : y \in Y, z \in Z\}$. Sort the sets and use part a. Here the time is $\Theta(n^2 \log n)$; the sorting is the slowest part.

Solution to Problem 4

a. For each string x, $|x| \ge n$, by an inductive application of the pumping property, there is a string z, $|z| < n$, such that

$$\forall y \in \Sigma^* \cdot xy \in L \Longleftrightarrow zy \in L$$

Thus x and z are in the same equivalence class with respect to L. It follows that the strings of length less than n suffice to specify all the equivalence classes with respect to L; but this is a finite number of strings and so L has a finite spanning set, i.e. L is regular.

b. Suppose L is regular. Then there is a dfa M of m states accepting L. Consider any string x, $|x| \geq m$. Consider the states that M passes through in reading x: $q_0, q_1, \cdots, q_m, \cdots$.. There must be a repetition among these states as M has m distinct states. Let q_i, q_j be the first repetition (i.e., j is chosen as small as possible). Let $x = x_1 x_2 \cdots x_p$, $p \geq m$. Define $u = x_1 x_2 \cdots x_{i-1}$, $v = x_i \cdots x_{j-1}$, and $w = x_j \cdots x_p$; then $x = uvw$, $|v| \geq 1$ and if $xy \in L$ if and only if $uwy \in L$ for M will be in the same state after reading either x or uw.

Solution to Problem 5

a. The following algorithm halts on exactly those $x \in L$.
while $f(x) \notin L_1$ **do** $x \leftarrow f(x)$.
For if the algorithm halts after k iterations, then $x \in L_k$.

b. Let L_1 be the set of accepting configurations of deterministic TM M. Let $f(C) = D$, where $C \overset{M}{\vdash} D$. Then L_k is the set of configurations of M from which M reaches an accepting configuration in $k-1$ steps. Thus initial configuration $C \in L$ if and only if M halts starting in configuration C. If L were recursive then we could test if TM M halts on initial configuration C, by testing if $C \in L$. But if M were the TM accepting K, this would show K was recursive, a contradiction.

Solution to Problem 6

Let $x_1, \cdots, x_r$ be the variables in 3-CNF formula F.

Consider determining if $F \in$ 3-SAT by recursively determining this for the formula with x_r set to True and False in turn (the base case are the variable free formulae).

This is an exponential time algorithm. To speed it up, we use dynamic programming. In particular, we keep in a table the values $f(G)$ for formulae G which have been determined to be unsatisfiable. Now, when the satisfiability of a new formula H is to be determined, we first check if $f(H)$ is already in the table, and if so there is no need to check further: H is also unsatisfiable.

It remains to bound the running time. Consider the subtree of the recursion tree that is actually explored.
Claim: each value k, $1 \leq k \leq n$ is associated with at most one internal node at each level of the tree.
For once a value k has occured at a node v of the explored recursion tree, thereafter when if it occurs at another explored node u, with u to the right of v (i.e., not a descendant of v), u will be a leaf of the portion of the explored recursion tree (for by then it will be known that the formula at node v is unsatisfiable).
Thus the explored recursion tree has at most nr internal nodes and at most $3nr$ nodes in all. As the evaluation of f takes polynomial time, as does the simplification of F for the recursive calls, it follows that the above algorithm runs in polynomial time.