Written Qualifying Exam
Theory of Computation

Fall, 1995

Friday, September 29, 1995

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. Part of your grade will be based upon your ability to communicate concisely as well as clearly. Assume standard results, except where asked to prove them.

Problem 1 (10 points)   (new booklet please)

Let $f(i,j)$ be defined for integers i and j in $\{1,2,3,\ldots,n\}$. Suppose f is stored in an n by n array. Further suppose f has the following special property.

Let i_0 be any integer in $\{1,\ldots,n\}$, and suppose that j_0 happens to minimize $f(i_0,\ *\ )$ over all j: $f(i_0,j_0)=\min\{f(i_0,1),f(i_0,2),\ldots,f(i_0,n)\}$. Then for $i>i_0$, a minimizing j (which minimizes $f(i,\ *\ )$) is among $\{j_0,j_0+1,j_0+2,\ldots,n\}$, and for $i<i_0$, a minimizing j is among $\{1,2,\ldots,j_0\}$.

a. (**5 points**.) Devise an algorithm that finds the minimum of $f(i,j)$, for $i,j\in\{1,2,\ldots,n\}$, and that runs in much less than quadratic ($O(n^2)$) time.

b. (**5 points**.) Estimate the running time of your algorithm. Note: chances are that the running time will not be the solution of a recurrence equation, but rather the result of analyzing the structure of the recursive calls and the parameter ranges distributed among these calls.

Problem 2 (10 points)   (new booklet please)

The following is standard for boolean expressions.

A *literal* is either a variable x or its negation $\bar{x}$.

A *clause* is a sum (disjunction, OR) of zero or more literals (where the empty clause has the truth value *false*).

A boolean expression is in *conjunctive normal form* (CNF, "product of sums") if it is a product (conjunction, AND) of zero or more clauses; the empty clause has the truth value *true*.

We will define a CNF boolean expression to be of type CNF(k) if every clause contains at most k negated variables. For example,

$$(\bar{x}_1+x_2+x_3)(x_3+\bar{x}_4+\bar{x}_5)(\bar{x}_1+x_2)$$

is CNF(2) but not CNF(1). Notice that CNF(0) expressions (with no empty clauses) can be trivially satisfied by setting all variables to the value true, since there are no negations in any clause. The satisfiability question for CNF(1), it turns out, can be answered in polynomial time, which is the subject of part b.

a. (**4 points**.)

Show that the satisfiability problem for CNF(2) is NP-complete.

b. (**6 points**.)

Present, at a high level, an efficient algorithm for solving the satisfiability problem for CNF(1). Hint: consider how to solve parts of the problem for clauses of one variable, and how difficult the problem is for larger expressions.

GO TO THE NEXT PAGE

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

For parts a, b, and c, let $G = (V, E)$ be a directed graph where each edge (i, j) has a signed real edge cost given by the array entry $Cost[i, j]$. You may assume that $Cost[i, j] = \infty$, if G has no edge from i to j. Please feel free to make any other assumptions you wish about the representation, but be sure to state them clearly. Suppose that G has no cycles where the sum of the edge costs is negative.

a. (**2 points**.) Present an algorithm, such as that due to Floyd and/or Warshall, which solves the all-pairs-shortest-paths problem for G. **Include path recovery code**, and very **briefly** explain how the path data is used to construct a shortest path.

b. (**7 points**.) We now extend the meaning of path to permit cycles. The algorithm for part a, it turns out, can be modified to find, for each pair of vertices i and j, the subgraph of G that is the union of all shortest paths from i to j, where a path is defined in this extended sense. Mercifully, we will not ask you for the simple algorithm, and why it might be correct. You also get for free the following useful fact: Let $H(i, j)$ be the subgraph comprising all shortest extended paths from i to j. Then every cycle in $H(i, j)$ has weight zero.

Now suppose that each edge (i, j) in G has some (non-negative) number $k[i, j]$ of cookies. When you traverse the edge, you get to eat all the cookies on the edge. Sorry, once an edge's cookies are eaten, they are gone, so there is no way to get an infinite supply by circling around a cycle forever.

Given the subgraph $H(i, j)$ that comprises the union of all shortest extended paths from i to j in G, explain how to find the subgraph (of $H(i, j)$) that comprises the best shortest super-extended path from i to j. A super-extended path may have cycles and it may even traverse some edges several times, if necessary. A shortest path is just that: the sum of its edge costs is the same as that for the simple paths computed in part b. A shortest super-extended path is best if it has contains at least as many cookies as any of the other shortest super-extended paths from i to j.

Note: You may wish to use your free fact in explaining the correctness of the algorithm.

c. (**1 point**.) Explain how to construct a graph G with $\Theta(n)$ edges and vertices where the best shortest super-extended path from 1 to n is a trip of $\Theta(n^2)$ edges.

Go to the next page

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

a. (**3 points**.) Show that $L_1 = \{w\#w \mid w \in \{a,b\}^*\}$ is not a CFL.

b. (**4 points**.) Show that $L_2 = \{uav\#wbx \mid |ux| = |vw| \text{ and } u,v,w,x \in \{a,b\}^*\}$ is a CFL. (Describe the CFL or p.d.a.; do not give the productions or a transition function.)

c. (**3 points**.) Conclude from part (b) that $L_3 = \{y\#z \mid y \neq z \text{ and } y,z \in \{a,b\}^*\}$ is a CFL.

Problem 5 (10 points) **(new booklet please)**

a. (**2 points**.) State Rice's Theorem.

b. (**4 points**.) Consider the set $L = \{x \mid \phi_x(0)\downarrow\}$. It follows from Rice's Theorem that L is not recursive. A different outline proof of this result follows; your task is to complete the outline.

Let $\phi_y = \psi$ be the following program:

```
$\psi(x)$  =  if $x = 0$ then $\psi(y+1)$
              else if $x = z+1$ $(z \geq 0)$ then
                  if $z \in L$ then do *Action 1*
                               else do *Action 2*
```

Specify *Action 1* and *Action 2* so as to produce a contradiction in the definition of function ψ.

c. (**4 points**.) Which of the following assertions are true? (True/False answers suffice.) For the true assertions, which of them can be shown using Rice's Theorem? Give one sentence justifications for each answer, positive or negative. (**Two questions per (true) part**).

(i) $\{x \mid \phi_x(x)\downarrow\}$ is not r.e.
(ii) $\{x \mid \phi_x(y)\uparrow \text{ for all } y\}$ is not recursive.
(iii) $\{x \mid \phi_x(2x)\uparrow\}$ is not recursive.
(iv) $\{x \mid \phi_x(x)\downarrow \text{ in } \leq x \text{ steps}\}$ is recursive.
(v) $\{x \mid \phi_x(y)\downarrow \text{ for all } y\}$ is not r.e.
(vi) $\{x \mid \phi_x(10)\downarrow\}$ is not recursive.

Problem 6 (10 points) **(new booklet please)**

a. (**2 points**.) Let $L \subseteq \{0,1\}^*$. Let $Pad(L, 2^{|x|^c}) = \{x\#0^y \mid 1+y+|x| = 2^{|x|^c} \text{ and } x \in L\}$. If $L \in \mathbf{NTIME}(2^{n^c})$, for some integer $c \geq 1$, show that $Pad(L, 2^{|x|^c}) \in \mathbf{NTIME}(am)$, for some constant $a \geq 1$, where $m = 2^{|x|^c}$.

b. (**5 points**.) Conclude that if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{NEXPTIME} = \mathbf{EXPTIME}$.
Hint: Show that $\mathbf{NTIME}(2^{n^c}) \subseteq \cup_{d\geq 1}\mathbf{DTIME}(2^{dn^c})$.

c. (**3 points**.) What is the difficulty in showing that $\mathbf{NTIME}(2^{n^c}) \subseteq \mathbf{DTIME}(2^{dn^c})$, for some fixed $d \geq 1$? (Continue to assume that $\mathbf{P} \neq \mathbf{NP}$.)

Solutions

Solution to Problem 1

a. This is a divide-and-conquer problem. The idea is to take a middle i value, i_0, and scan the j's to find a j_0 where $f(i_0,*)$ is minimized. Then the process is repeated for the two subproblems comprising (a) the subintervals $i \le i_0$ and $j \le j_0$, and (b) the subintervals $i > i_0$ and $j \ge j_0$. The least f value of these two recursive subproblems is returned. While this gives full credit, a more detailed (and unnecessary) description is given below. We even consider the case where the middle value of i_1 and i_2 is i_2.

Define $J(i, j_1, j_2)$ to be some value j where $j_1 \le j \le j_2$ and $f(i,j) = \min_{j\in[j_1..j_2]} f(i,j)$. Clearly $J(i, j_1, j_2)$ can be computed in $O(j_2 - j_1 + 1)$ time by a simple sweep. Then we may define

```
**function** minf($i_1, i_2, j_1, j_2$)
    **if** $|i_1 - i_2| \le 1$ **then return**$\{\min(J(i_1, j_1, j_2), J(i_2, j_1, j_2)\}$
        **else**
            $ii \leftarrow \lfloor \frac{i_1+i_2}{2} \rfloor$;
            $jj \leftarrow J(ii, j_1, j_2)$;
            **return**$\{\min(minf(i_1, ii, j_1, jj), minf(ii+1, i_2, jj, j_2))\}$
    **endif**
**end**.
```

b. The set of recursive calls issued by minf comprise (essentially) a complete binary tree. Each call does an amount of work that is proportional to the range scanned by the call to J. Unfortunately, these ranges are not known, at any node of the tree (other than the root, where $j_1 = 1$, and $j_2 = n$.) However we know that the sum of the ranges, at any level of the tree is about n. To be more precise (which is unnecessary), we know that only the endpoints of each consecutive range can be the same, which means that the number of values considered by J, at a level of the tree with k nodes is $n + k - 1$. Well, this value is bounded by $2n$. There are at most $1 + \log n$ levels to this tree, since the length of the i intervals halve at each level. We conclude that the total work is $\Theta(n \log n)$.

Solution to Problem 2

a. We show that CNF(2) is in NPC.

First, it is in NP. This can be established by observing that a satisfying assignment to CNF(2) can be trivially verified in linear time (or by observing that CNF(2) is within CNF).

Second we give a polynomial time reduction of 3-SAT to CNF(2), thereby showing that 3-SAT$\le_P$ CNF(2). Since 3-SAT is in NPC, it follows that CNF(2) is as well. The reduction is easy. For 3-SAT clauses that do not have 3 negations, we use them as-is, since they are already in CNF(2). For each clause of the form $(\bar{x}_i + \bar{x}_j + \bar{x}_k)$, we introduce a new variable y_{ijk}, and include the new pair of clauses $(\bar{x}_i + \bar{x}_j + y_{ijk})$,$(\bar{y}_{ijk}, \bar{x}_3)$. It is straightforward to see that $(\bar{x}_i + \bar{x}_j + y_{ijk}) \cdot (\bar{y}_{ijk}, \bar{x}_3) \equiv (\bar{x}_i + \bar{x}_j + \bar{x}_k)$. The resulting expression is in CNF(2), and we are done.

b. The hint is helpful. It points out that a clause with one literal must have the variable set so that the literal evaluates to true, and a single CNF(1) clause with more than one literal can be satisfied by setting the named variables to true, since it must contain an unnegated variable. We apply these two ideas (in the order listed) in a **dynamic** setting.

We dump clauses of one variable into a ready queue, and put the longer clauses into a wait queue. Those interested in implementation will, for each variable, form a set of pointers to the clauses containing that variable (negated or otherwise). The procedure is:

```
while the ready queue is not empty and unsatisfiability has not been established do
    Pick a clause from the ready queue and assign the satisfying value to the variable $x$
    contained in it.
    Update all clauses $c$ still containing $x$ as follows:
    If the assignment to $x$ satisfies $c$ then throw $c$ away;
    else remove the literal in $x$ from $c$, and move $c$ to the ready queue if $c$ now contains just
    one literal.
    If now $c$ contains zero literals, it is not satisfied, and the system is found to be unsatisfi-
    able.
endwhile
```

If the system is not yet unsatisfiable, then set all unassigned variables to true, since the remaining clauses contain more than one literal.

It is easy to see that the system is satisfiable iff the procedure succeeds. The running time is linear in the size of the CNF(1) expression, provided we have a unit time look-up procedure for variable names, which will be the case if they are named 1,2,3,$\cdots$, and array access is used.

Solution to Problem 3

a)

```
procedure FW(var A[1..n,1..n], FirstIn[i,j]);
initialize all FirstIn[i,j] to j;
assume $A[i,j] = \infty$ if there is no edge from $i$ to $j$, and is the edge cost otherwise;
forall k in (1..n) do
        for all pairs i,j in [1..n] do
                if $A[i,j] > A[i,k] + A[k,j]$ then
                        $A[i,j] \leftarrow A[i,k] + A[k,j]$;
                        $FirstIn[i,j] \leftarrow FirstIn[i,k]$
                endif
        endfor
endfor.
```

FirstIn[i,j] gives the first vertex after i on some shortest path from i to j.

b) From the problem description, we see that any zero cost cycle that touches a shortest path should be taken so that its cookies can be found. Further, any cycle in $H[i,j]$ has zero cost.

The algorithm at a very high level is: find all the strong components in $H(i,j)$, and find the cookie count for each component. The best path is the shortest path with the largest cookie count for its bridge edges (outside of the strong components) plus the strong components touched by the path.

In greater detail, we might solve the problem by using the FW algorithm to compute all shortest path distances, and use, say Tarjan's algorithm (or Sharir's or somebody else's) Strong Components algorithm to identify all strong components. If vertex i has an edge into strong component X, and X has an edge exiting the SC to vertex j, then we may create an edge from i to j with cost computed by the FW algorithm, and cookie count equal to the count from the strong component plus the cookie count on the edge from i and the edge to j. We are using the fact that each edge is on some shortest path, and that when two shortest paths cross, any branching will give a shortest path. Then we may solve the problem on the new graph, which is cycle free. Here we want the path with the largest number of cookies (it must be as short as possible). There are many solutions. The Floyd-Warshall algorithm will work with inequalities reversed, since the graph is cycle free. Alternatively, a postorder topological processing can be used.
c) Let all edge costs be zero. Use $2n$ nodes. Let there be a directed chain from 1 to 2 to 3 to $\cdots$ to n. Now include the following $2n$ edges: $(n,n+1),(n,n+2),(n,n+3),\ldots,(n,2n)$ and $(n+1,1),(n+2,1),(n+3,1),\cdots,(2n,1)$. There are $3n-1$ edges, and n different basic cycles, which all go through the directed chain. The cycles each differ in one vertex. The total number of edges needed by a trip that traverses every edge must be $n(n+1)$.

Solution to Problem 4

a. We use the Pumping Lemma. If L_1 is a CFL, there is a parameter n, such that for strings $w\#w$ with $|w\#w| > n$, we can find strings u,v,x,y,z such that $w\#w = uvxyz$, $uv^ixy^iz \in L$ for all i, $0 < |vy|$ and $|vxy| \le n$. Now consider the string $w = 0^n1^n$. Then $uvxyz = 0^n1^n\#0^n1^n$. $uxz \in L_1$ by the Pumping Lemma, thus $uxz = 0^m1^p\#0^m1^p$, for some $m,p \le n$. But then $v = y = 0^{n-m}1^{n-p}$ and $|vxy| > n$, a contradiction.

b. The p.d.a. to accept L_2 proceeds as follows. The idea is to verify that $|ux| = |vw|$. To do this, as it reads u it pushes one copy of the symbol \$ onto the stack for each symbol read. As it reads v it pops the stack once for each symbol of v until either the stack is empty or all of v is read. In the event the stack is emptied, it continues by pushing a second symbol # for each subsequent symbol of v read. On reading w, the symbols # are popped, one for each symbol of w read; then for each remaining character of w, a \$ is pushed. Finally, as x is read, for each character of x the stack is popped. The stack will be empty exactly when x is fully read if $|ux| = |vw|$. The p.d.a. non-deterministically recognizes the a and b (separating u and v, and w and x, respectively).

c. Suppose $y \neq z$. If $|y| = |z|$, we can write $y = ucv$ and $z = wdx$, where $|u| = |w|$, $|v| = |x|$, $c \neq d$, and $c,d \in \{a,b\}$. Thus a p.d.a. can nondeterministically decide to check either that $|y| \neq |z|$ or that the string is in L_2, or a similar language $\widetilde{L_2}$ with the roles of a and b reversed. For if $|y| = |z|$ and $y\#z \in L_2$ then $|ux| = |wv|$ implies that $|uv| = |ux|$ and hence $|v| = |x|$; similarly, $|u| = |w|$.

Solution to Problem 5

a. Let C be a class of programs (i.e. if $\phi_i = \phi_j$ then $i \in C$ iff $j \in C$). If C is recursive then either C or its complement is the empty set.

b. *Action 1* $= \uparrow$; *Action 2* $= \downarrow$.
Then if $\psi(y+1)\downarrow$, it must be because $y \notin L$, i.e. $\phi_y(0)\uparrow$, i.e. $\psi(0) = \psi(y+1)\uparrow$.
While if $\psi(y+1)\uparrow$, it must be because $y \in L$; i.e. $\phi_y(0)\downarrow$, i.e. $\psi(0) = \psi(y+1)\downarrow$.
This a contradiction.

c. (i) is false, the others are true.
Rice's Theorem shows that sets are not recursive; thus it cannot be used to show (iv) or (v). Clearly (ii) and (vi) are class properties and thus can be shown by Rice's Theorem. (iii), however, is not a class property, and so cannot be shown using Rice's Theorem (i.e. there may be i and j with $\phi_i = \phi_j$, and yet $\phi_i(2i)\uparrow$ while $\phi_j(2j)\downarrow$; indeed such i and j can be constructed, although this is outside the scope of this question).

Solution to Problem 6
a. Let M be a TM accepting L in nondeterministic time 2^{n^c}. TM M' accepts $Pad(L, 2^{|x|^c})$ as follows. First it checks that the input has the form $x\#0^{|x|^c-|x|-1}$; it is straightforward to do this in time $O(n)$. Second, it simulates M, ignoring all input to the right of x. This takes nondeterministic time n. Thus there is an $a \geq 1$ such that M' runs in nondeterministic time bounded by an.

b. It suffices to show how to simulate an arbitrary nondeterministic TM M running in time bounded by 2^{n^c}. M' performs the simulation as follows.
Step 1. Pad the input x to form $x\#0^{2^{|x|^c}-|x|-1}$. This takes time bounded by $e \cdot 2^{n^c}$, for some constant e, where $|x| = n$.
Step 2. Let M'' be the $Ntime(am)$ TM accepting $Pad(L(M), 2^{|x|^c})$, where $m = 2^{|x|^c}$. By assumption, there is a deterministic TM M''' running in time bounded by $(am)^f$ for some constant f, with $L(M'') = L(M''')$. Run M''' on the input calculated in Step 1. M''' takes time bounded by $(a \cdot 2^{|x|^c})^f = a^f \cdot 2^{f|x|^c}$.
The total time taken by M' on input x is bounded by $e2^{n^c} + a^f \cdot 2^{fn^c} \leq 2^{dn^c}$, for some constant d.

c. The obstacle to obtaining a fixed d is that there may be a distinct constant f for each M''. Unfortunately, each M results in a distinct M'' and potentially in a distinct f. Further it may be that these constants f are unbounded and thus the corresponding constants d must also be unbounded.