Forward

Welcome to the refurbished Computer Science Preliminary Exams Study Guide.

The guide is organized in four sections. Each section is being sold separately. The first three sections are for the three core exams (hardware, software, and theory). Each one contains the eight most recent exams, along with solutions for five of them. Some solutions are written by faculty, most by high-scoring students. Our thanks go to those anonymous students contributing solutions.

Each section contains a syllabus which indicates the subject areas and reading material covered by the exam. The fourth section consists only of general descriptions and syllibi for each of the research area orals.

If you have questions about one of the exams or general questions about prelims, you can consult the *EECS Graduate Information* booklet, Kathryn Crabtree, or the faculty member in charge of prelims (currently Paul Hilfinger). Specific questions about the exams, such as *how the heck do you do question 4 of Spring 86 Software?* get answered by your fellow students in the Prelim Review sessions which the CSGSA will organize each semester.

Special thanks to David Gedye, who created the modern form of this guide, and to Joe Konstan.

Steve Lucco(CSGSA Prelim Liason Officer)

Kathryn Crabtree(CS Grad. Assistant and Prelims Coordinator)

Fall 1989

14 May 1987

GUIDELINES FOR THE CORE THEORY EXAM

The past few editions of the examination have been quite similar. Students can expect that future editions will not be very different from the established model. Typically, there are six questions, as indicated below. A minimum passing grade is usually gained by achieving essentially full credit on three questions, plus partial credit on another.

1,2 Algorithm Design (2 questions)

These questions call for creative design of algorithms for problems that typically involve graphs or other combinatorial structures. In order to deal with these questions, the student should be familiar with well-known principles of design, e.g., divide-and-conquer, dynamic programming, depth-first search. Knowledge of basic data structures, e.g., edge-list representation of graphs, heaps, 2-3 trees, union-find data structures, is also assumed as background. It is likely that an estimate of the (worst-case) running time and/or space requirements of the algorithm will be part of the question. For this purpose it is necessary to know such basic facts as the time required for insertion and deletion of keys from a priority queue. The student may also be required to formulate and solve simple recurrence relations in order to obtain a time bound.

3. Lower Bounding

A typical question of this type is "Show that such-and-such problem is at least as difficult as sorting." The student should understand and be able to apply the decision-tree model of computation, and adversary and information-theoretic bounding arguments.

4. Languages, Particularly Context Free Languages

Typical questions are of the form "Is the intersection of a context free language with a regular language a regular language?", "Is there a decision procedure for determining whether or not a context-free grammar is ambiguous?", "Prove that the intersection of two recursively enumerable sets is a recursive set", "Show that the language L shown below can be accepted by an automaton of type X (or generated by a grammar of type Y) but cannot be accepted by any device of type Z". The student should have a good working knowledge of basic definitions and properties of languages, grammars and machines, the Chomsky hierarchy, the pumping lemma, the Church-Turing thesis, proofs of undecidability, etc.

5. Machines, Particularly Finite State Machines

A typical question might be "Construct a finite state machine with a minimum number of states for recognizing the language represented by the following regular expression." The student should know about Mealy vs Moore machines, machines vs automata, determinism vs nondeterminism, acceptance vs recognition, be able to carry out state reduction, construct a regular expression from an automaton and vice versa. Knowledge about the properties of regular sets is assumed. Though the student is not necessarily expected to have any background knowledge concerning the topic, it would be fair to ask the student to devise a simple procedure for state identification or homing of a finite state machine.

6. NP-Completeness

It is traditional to ask for an NP-completeness proof. Typically this is done by suggesting a known NP-complete problem as candidate for the problem transformation.

SYLLABUS OUTLINE FOR THE THEORY CORE PRELIM EXAM

Recommended Courses: CS 170 and CS 172

1. Algorithms and Complexity
   [Baase, sections 1.1, 1.3, 1.4, 1.5]
   - average vs. worst cases analysis
   - upper and lower bounds
   - O, o, Ω notation

2. General techniques for Algorithm Design
   - divide and conquer
     [Aho, Hopcroft and Ullman, sections 2.6, 2.7]
   - dynamic programming
     [Aho, Hopcroft and Ullman, section 2.5]
   - correctness proofs using inductive assertions
     [Baase, pp. 17-20]
   - formulating and solving recurrences
   - methods for proving lower bounds
     - information bound
       [Baase, pp. 60-63]
     - adversary argument
       [Baase, section 1.5]

3. Algorithms to Manipulate Data Structures
   - binary search trees
     [Aho, Hopcroft and Ullman, section 4.4]
   - 2-3 trees
     [Aho, Hopcroft and Ullman, section 4.9]
   - AVL trees
     [Horowitz and Sahni, pp. 442-456]
   - heaps
     [Aho, Hopcroft and Ullman, section 3.4]
   - union-find data structure (omitting analysis)
     [Aho, Hopcroft and Ullman, sections 4.6, 4.7]

4. Sorting and Searching
   [Reingold, Nievergelt and Deo, sections 6.5, 7.1, 7.3]
   - binary search
   - heapsort
   - quicksort
   - bucketsort
   - hashing
   - linear time selection

5. Graph Algorithms
   [Baase, chapter 3]
   [Reingold, Nievergelt and Deo, sections 8.1, 8.2]
   - edge list and adjacency matrix representation of graphs
   - depth-first search and applications of it
     - biconnected components, strong components
   - breadth-first search
   - minimum spanning tree
   - shortest paths

6. Languages
   - basic properties of strings and languages
     [Lewis and Papadimitriou, sections 1.8, 1.9]
   - regular languages, grammars and expressions
     [Lewis and Papadimitriou, sections 1.9, 3.2]

- context free languages and grammars
  [Lewis and Papadimitriou, sections 3.1, 3.2]
- unrestricted grammars
  [Lewis and Papadimitriou, section 5.2]
- recursive (i.e., Turing decidable) and recursively enumerable (i.e., Turing acceptable) languages
  [Lewis and Papadimitriou, sections 4.2, 6.1]
- Church's thesis
  [Lewis and Papadimitriou, section 5.1]
- unsolvability
  [Lewis and Papadimitriou, section 6.1]
- e.g. halting problem

7. Machines
   - finite automata
     [Lewis and Papadimitriou, sections 2.1, 2.2]
   - pumping lemma
     [Lewis and Papadimitriou, section 2.6]
   - pushdown automata
     [Lewis and Papadimitriou, section 3.3]
   - Turing machines
     [Lewis and Papadimitriou, sections 4.1, 4.2, 4.5, 4.6]
   - determinism vs. nondeterminism
   - Church's thesis
     [Lewis and Papadimitriou, section 5.1]

8. NP-completeness
   [Garey and Johnson, pp. 1-62]
   - P, NP, NP-complete problems
   - Cook's Theorem
     - general understanding of proof
   - polynominal reductions and proof techniques

Sections 3, 4, and 5 list several important algorithms. In each case you should be able to:
a) State the algorithm clearly in a notation of your choice (pidgin PASCAL is often convenient);
b) Give an informal proof of correctness;
c) Determine the worst-case execution time and storage requirements and, if an elementary proof is possible, the average time and storage requirements;
d) Compare the algorithm with others available for the same task;
e) Apply the methods of analysis to other related problems.

In Sections 6 and 7 the emphasis is on understanding the basic definitions and properties. Proofs will be expected only when they are short and simple.

Aho, Hopcroft and Ullman: The Design and Analysis of Computer Algorithms
Baase: Computer Algorithms
Garey and Johnson: Computers and Intractability
Horowitz and Sahni: Fundamentals of Data Structures
Lewis and Papadimitriou: Elements of the Theory of Computation
Reingold, Nievergelt and Deo: Combinatorial Algorithms

Theory Core Exam: Fall 1985

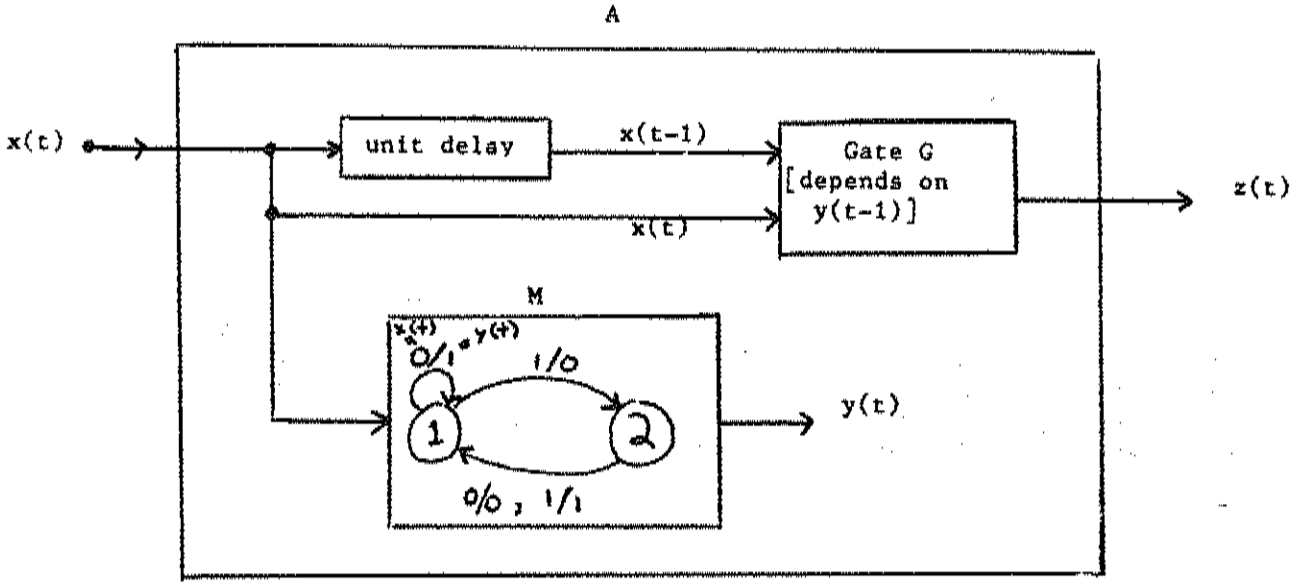
I.D. Number ____________________

Fall 1985

CS Undergraduate Theory Preliminary Exam

This examination contains five questions. Please answer each question using a separate piece of paper on which you have written your I.D. number. You have three hours in which to work. Partial credit will be given for all problems based on your reasoning. All problems carry the same weight.

GOOD LUCK



Problem 1

The system A shown above has binary input and output. At any time t, gate G (inside A) is either OR or AND, depending on the output of the finite-state machine M at time t - 1. Specifically,

Gate G at time t is OR if y(t - 1) = 0,

Gate G at time t is AND if y(t - 1) = 1.

(a) Define the states s(t) of a finite-state machine that models the system A.

(b) Draw the transition diagram of your model.

(c) Minimize the diagram produced in (b) (if not already a minimal finite-state machine).

Problem 2

Let x and y be two strings of characters from some alphabet. Consider the operations of deleting a character from x and inserting a character in x. We want to determine the minimum number of such operations needed to transform x into y. Describe an algorithm which finds this number and estimate its running time within O (big Oh; Order). (Note: The algorithm of interest is not the one which transforms x into y, but the one which computes the minimum number of operations required for this transformation!) The speedier your algorithm, the more credit you will receive.

Problem 3

Which of the following problems is decidable and which is not? Give your reasoning.

PROBLEM A PRINTING PROBLEM

INSTANCE: a one-tape Turing machine T with a start state q_0, a finitely inscribed tape t marked with a starting position for T, and an integer k = the number of 1's on tape t. t is infinite in both directions. $\sum = \{0,1\}$. ("0" is the blank symbol. t contains O's on all but k tape squares.)

QUESTION: Will T[t] (T started in state q_0 at the starting position of t) ever print a "1" (i.e., print a "1" in the place of a "0")?

PROBLEM B LOOPING PROBLEM

INSTANCE: Same as for Problem A.

QUESTION: Will T[t] loop, i.e. will T[t] enter the same Turing machine configuration twice (at two different times)? Recall that a Turing machine configuration is a tuple consisting of the Turing machine's state q_i, its tape configuration t', and the tape square being scanned c_j.

Problem 4

Give an example of a class of regular languages $L_1, L_2, \ldots, L_k, \ldots$ with the property that: (1) The smallest deterministic finite state automaton for L_k requires at least 2^k states, whereas (2) a deterministic pushdown automaton for L_k exists which has O(k) states.

Problem 5

Parts (a) and (b1) are worth 1/10 credit; (b2) is worth 8/10.

(a) State the VERTEX COVER problem (recall that it has to do with the existance of a set of nodes that covers all the edges).

(b) Read the following problem, then answer (b1) and (b2).

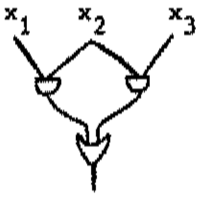
PROBLEM: $\wedge\vee$ CIRCUIT PROBLEM

INSTANCE: A cycle-free circuit with m inputs (m = a positive integer) and 1 output, which is constructed from 2-input AND gates

and OR gates

; a positive integer k, k $\leq$ m.

QUESTION: Does there exist a subset of $\leq$ k input lines such that when these input lines are 1 (TRUE), the output is 1?

(b1) 

Answer the above QUESTION for the following INSTANCES:

with k = 1: ____________________

with k = 2: ____________________

(b2) Prove that the $\wedge\vee$ Circuit Problem is NP-complete.

Theory Core Exam: Spring 1986

Spring 1986

CS Undergraduate Theory Preliminary Examination

Do not turn the page before you hear the starting gun.

In the meantime... Please print your I.D. number on the cover of your blue book.

This examination contains four questions, which are to be answered in your blue book.

The examination is *closed book:* you may not use any textbooks, notebooks, or other written material that you have brought into the examination room with you. Calculators, though unnecessary, are permitted.

You have three hours in which to work.

Partial credit will be given for all problems based on your reasoning.

All problems carry equal weight.

GOOD LUCK

— 1/2 hour —

1. A straight-line program for computing x^n is a finite sequence

$$x \rightarrow x^{i_1} \rightarrow x^{i_2} \rightarrow \ldots \rightarrow x^n$$

constructed as follows: The first element is x. Each succeeding element is either the square of some previously computed element or the product of two previously computed elements. The number of multiplications to evaluate x^n is the number of terms in the shortest such program-sequence minus 1.

What is the minimum number of multiplications to evaluate x^{30}? Do not assume that the obvious solution is best!

Prove that a smaller number of multiplications is impossible.
HINT: Don't simply enumerate all possibilities. While some enumeration may be useful, a completely enumerative lower bound is unnecessary and boring.

— 1/2 hour —

2. Consider the following TM (Turing Machine) problem:

INPUT: A 1-tape TM on alphabet $\{0, 1\}$, its start state $= q_0$, and a finitely inscribed tape t (t contains 0s on all but a finite number of tape squares) with pointers to the leftmost 1, the rightmost 1, and the starting position.

OUTPUT: YES if the TM halts when started in state q_0 on the starting position of tape t;

NO if it loops, by which we *specifically* mean that it reenters some previously entered Turing machine configuration. Recall that a *Turing machine configuration* is a 3-tuple consisting of the Turing machine's state q_i, its tape configuration t', and the tape cell being scanned c_j.

Does there exist an algorithm for solving the above Turing Machine problem in each of the following cases:

1) The algorithm is *not* required to halt if the Turing machine neither halts nor loops.

2) The algorithm *is* required to halt if the TM neither halts nor loops, in which case it may output anything at all including YES or NO (i.e., it is permitted to tell a lie when the TM neither halts nor loops!).

Give solid arguments to support your answer.

HINT: Recall the diagonalization argument used to prove the Halting Problem undecidable.

— 1 hour —

3. Prove that one of the following two problems is NP-complete, and that the other is solvable in polynomial time.

PROBLEM 1: DIRECTED FEEDBACK ARC SET
INPUT: A directed graph G and a positive integer K.
QUESTION: Is there a set of K edges whose removal from G eliminates all directed cycles?

PROBLEM 2: UNDIRECTED FEEDBACK ARC SET
INPUT: An undirected graph G and a positive integer K.
QUESTION: Is there a set of K edges whose removal from G eliminates all cycles?

You may assume that the following problem is NP-complete:
PROBLEM: VERTEX COVER
INPUT: An undirected graph G and a positive integer K.
QUESTION: Is there a set S consisting of K vertices of G, such that every edge of G meets (i.e., is "covered" by) at least one vertex in S?

— 1 hour —

4. Let $(x_1, x_2, \cdots, x_n)$ be an array of real numbers in the memory of a random-access computer, and let N be a real number which is less than $\sum_{i=1}^{n} x_i$. You are to devise an algorithm to compute the unique real number y such that $\sum_{i=1}^{n} \min\{x_i, y\} = N$.

EXAMPLE: If $(x_1, x_2, \cdots, x_n) = (9, 3, 10, 4, 2)$ and $N = 21$, then $y = 6$. Why?

Give the most efficient algorithm you can find for solving this problem. Efficiency is measured by the number of steps required in the worst case as a function of n. Arithmetic operations, comparisons and accesses to array elements each count as one step. Explain briefly why your algorithm works and how your time bound was arrived at.

Theory Core Exam: Fall 1986

University of California
College of Engineering
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Fall 1986

CS THEORY PRELIMINARY EXAMINATION

The six questions count equally. Please write answers in blue books. Brevity and clarity in your answers are important.

GOOD LUCK!!

1. If two sequences $a_1,a_2,\ldots,a_m$ and $b_1,b_2,\ldots,b_n$ are interleaved, we say that the resulting sequence $c_1,c_2,\ldots,c_{m+n}$ is a *shuffle* of the first two. For example,

$$2,3,3,2,2,5,4,4,5,3,2,3,2,4,5$$

is a shuffle of 2,3,2,5,4,3,2,4 and 3,2,4,5,2,3,5 since it can be obtained by interleaving those two sequences in this way:

```
2,3         2,5,4        3,          2,4
      3,2          4,5         2,3
```

You are to give a dynamic programming algorithm for determining whether or not a given sequence is a shuffle of two other given sequences. Your algorithm is to run in time $O(mn)$, where m,n and $m + n$ are the lengths of the three sequences and $m \leq n$.

2. In a directed graph G with vertex set V and edge set E, vertex u is called a *source* if every vertex is reachable from u by a directed path (by convention, u is automatically reachable from itself).

Give an algorithm, running in time $O(|V| + |E|)$, to find a source in G when one exists, and otherwise to determine that G contains no source. A high-level description of the algorithm will suffice, but your argument for the upper bound on execution time should be convincing.

3. In the *element distinctness problem* one is given a list of n numbers. The task is to determine whether the n given numbers are all distinct (i.e., that no two of them are equal). The primitive step is to compare two of the numbers, say x and y. Such a comparison has three possible outcomes: x less than y, x equal to y and x greater than y. Derive the best lower bound you can on the worst-case number of comparisons required by every algorithm that solves the element distinctness problem.

4. Prove that the following problem is NP-complete by showing that it lies in NP and providing a transformation from SATISFIABILITY.

SET SPLITTING

INSTANCE: Collection C of subsets of a finite set S.

QUESTION: Is there a partition of S into two disjoint subsets S_1 and S_2 such that no subset in C is entirely contained in-either S_1 or in S_2?

Example: $S = \{1,2,3,4\}$ and C contains subsets $\{1,2,3\}$, $\{2,3\}$, $\{1,4\}$ and $\{3,4\}$. Here the answer is "yes" since we can choose $S_1 = \{1,3\}$ and $S_2 = \{2,4\}$.

Suggestion: For each instance of SATISFIABILITY, with variables $x_1, x_2, \ldots, x_n$, let the elements of S be the $2n$ literals $x_1, x_2, \ldots, x_n$, $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$ plus a special symbol F intended to represent the constant value "false".

5. Let Σ be an alphabet consisting of the p symbols $a_1, a_2, \ldots, a_p$. Let $L \subseteq \Sigma^*$ be the set of all nonempty words x over Σ such that the last symbol of x does not occur elsewhere in x. Thus

$$L = \bigcup_{i=1}^{p} (\Sigma - \{a_i\})^* a_i .$$

a) Give a nondeterministic finite automaton with $p + 2$ states that recognizes the language L;

b) Prove that every deterministic finite automaton that recognizes L has at least $2^{p+1} - 1$ states.

Hint: If two input strings y and z lead the deterministic automaton to the same state, then y and z must share certain properties. What are these properties?

6. Prove: If L is a context-free language and R is a regular set then $L \cap R$ is a context-free language.

FALL 1986
CORE THEORY EXAM

~~Problem #1~~

10/10

1) Given strings $S_1[0..m]$ and $S_2[0..n]$, $S_3[0..m+n]$ to determine whether S_3 is a shuffle of S_1 and S_2:

Consider the table $A[0..m][0..n]$, where $A[i][j]$ is ~~true iff~~ defined to be true if the string of the 1st i characters of S_1 and the string of the 1st j characters of S_2 shuffle to the first $i+j$ characters of S_0, false otherwise.

The answer to whether S_1 and S_2 shuffle to S_0 is $A[m][n]$.

(0) $A[0][0]$ is true — shuffling empties is empty

(1) $A[0][j]$ for $0 < j \leq n$ is true iff $A[0][j-1]$ is true and $S_2[j] = S_0[j]$

(2) $A[i][0]$ is true (for $0 < i \leq m$) iff $A[i-1][0]$ is true and $S_1[i] = S_0[i]$,

(3) $A[i][j]$ for $0 < i \leq m$ and $0 < j \leq n$ is true iff one of ~~$A[i-1][j]$ is true iff~~

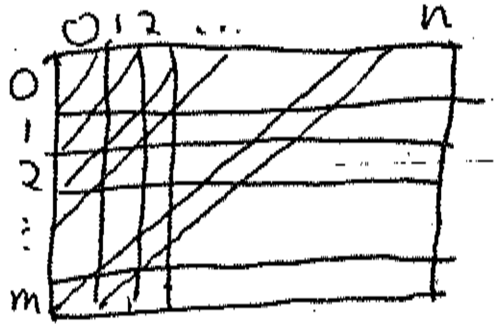
(a) $A[i-1][j]$ is true and $S_1[i] = S_0[i+j]$, $i \geq 1$

or (b) $A[i][j-1]$ is true and $S_2[j] = S_0[i+j]$, $j \geq 1$

Each of the above statements just says that if two strings shuffle to form a third, the last character of the 3rd must be the last char of 1 of the other 2 strings, and taking out that character the strings also shuffle.

1 cont'd

The program is just to fill in the entries of table A 1 diagonal at a time for the table shown. A[i][j] needs to look at 2 entries in A, and compare 2 pairs of chars, so running time to find A[m][n] is $\Theta(mn)$.



~~at~~

~~READ FIRST:~~

~~I mistakenly did everything from 1 to m or n. The table should start at 0, the ~~same~~ defs for A[1][i] etc. should be reworded for A[0][i] etc., changing the subscript of S_0 to be checked (i.e. check against S_0[i] for the A[0][i] case].~~

A[0][0] is defined to be true when the program starts.

10/10

#3 Keeping track of the result of each comparison gives you a partial order on the set of n numbers, at each step slightly more refined than the last. If, at any step, two of the elements are incomparable (according to the current partial order), the algorithm may not terminate because they might be equal. ~~But~~ To complete the partial order so that every 2 elt's are comparable is equivalent to sorting the numbers. ~~and clearly sorting the numbers~~ Thus, at least $\lceil n \log_2 n \rceil$ comparisons are required for the case when all n elements are distinct.

10/10

#4 Show in NP: say the instance is m subsets $C_1, \dots, C_m$ of set S, $|S| = n$. If the algorithm guesses S_1 and S_2 (which it can do in linear time), it can then check for each C_i that $C_i \not\subseteq S_1$, $C_i \not\subseteq S_2$ in no worse than $|C_i| \cdot 2n$ ~~times~~ comparisons, checking each elt of C_i against each elt of S_j, so Set Splitting is in NP. (at most $2n$ steps for each elt. read in the input, $< 2N^2$ where N is the total size of the inputs).

Take an instance of Satisfiability, and put it in the form

$$C_1' \wedge C_2' \wedge \dots \wedge C_t' \quad \text{where each } C_i'$$

is an $\vee$ of $x_1, x_2, \dots, x_n, \overline{x}_1, \overline{x}_2, \dots, \overline{x}_n$ ie. Conjunctive normal form (I think, I forget the terminology). This takes poly. time.

The instance of Set Splitting will be as in the suggestion:

$$S = \{x_1, \dots, x_n, \overline{x}_1, \dots, \overline{x}_n, F\}$$

We will need ~~Clauses~~ subsets $X_i = \{x_i, \overline{x}_i\}$

and $C_i =$ [illegible]

$=$ [illegible]

$= \{F\} \cup \{x_j \mid x_j \in C_i'\} \cup \{\overline{x}_j \mid \overline{x}_j \in C_i'\}$

This is clearly polynomial (in fact, linear!) transformation.

If there exist S_1, S_2 partitioning S

s.t. no C_i or X_i is a subset of S_1 or S_2 then:

~~considers~~ we can consider the S_i containing F as the false assignments to the original satisfiability problem, and the S_i without F the true assignments.

Since $X_i \not\subset S_1$, $X_i \not\subset S_2$ we have a valid assignment. Without loss of generality, say $F \in S_1$.

~~The original equation is satisfiabl if and only if its complement is not,~~

~~$\iff \overline{C_1'} \vee \overline{C_2'} \vee \cdots \vee \overline{C_t'}$ is false~~

~~$\iff \overline{C_i'}$ is false for all i~~

~~$C_i' = X_{a_1} \vee X_{a_2} \vee \cdots \vee X_{a_n} \vee \overline{X}_{b_1} \vee \cdots \vee \overline{X}_{b_t}$~~

~~$\overline{C_i'} = \overline{X}_{a_1} \wedge \overline{X}_{a_2} \wedge \cdots \wedge \overline{X}_{a_n} \wedge X_{b_1} \wedge \cdots \wedge X_{b_t}$~~

~~$\overline{C_i'}$ is true iff all elements of C_i are false.~~

~~The construction of the set splitting instance has a partition if and only if no C_i is a subset of S_1.* Thus Set Splitting is NP-complete.~~

~~* and $X_i \not\subset S_1$ $X_i \not\subset S_2$~~

~~#4 Maybe that was not clear.~~
~~If original problem was satisfiable,~~
~~then let~~

#4 cont'd

If the equation is satisfiable, ~~its complement is not & vice versa.~~ for assignment A
~~iff $\overline{C_1} \vee \overline{C_2} \vee \dots \vee \overline{C_t}$ is false~~
let $S_2 = \{X_i \mid x_i \text{ true in } A\} \cup \{\overline{X_i} \mid x_i \text{ false in } A\}$
let $S_1 = (S - S_2) \cup \{F\}$

Clearly no $X_i \subset S_1$ and no $X_i \subset S_2$ since A is an assignment. If any $C_i \subset S_1$ (and $C_i \not\subset S_2$ since $F \in C_i$), then C_i would be false, ~~iff~~ which contradicts assumption eqn is satisfiable, so no $C_i \subseteq S_1$ or S_2, so Set Splitting says

~~If Set-Splitting says NO, i.e. any partition $S_1 \cup S_2$ of S must have C_i or X_i in one of S_1 or S_2 for every such partition.~~

If the equation is not satisfiable, then for every assignment A some C_i is false. But any partition of $S = S_1 \cup S_2$, $X_i \not\subset S_1$, $X_i \not\subset S_2$ is an assignment under the rule that whichever X_i or $\overline{X_i}$ are in the same set with F are false, otherwise are true, and so for every partition $C_i \subset S_1$ or $C_i \subset S_2$ for some C_i, so Set splitting says N

Thus the equation is satisfiable if and only if the new instance can be set split.

Good

#6 If L is context-free, there is some non-deterministic PDA that accepts L, and since R is regular, there is a DFSA that accepts R. Let P(L) be a PDA that never reads more than 1 character from the input string at a time, (such NPDA's are of course as powerful as those that read strings on a state move)

~~$P(L) = \Sigma K$~~

To create a PDA that accepts $L \wedge R$, ~~an~~ give it states $K_L \times K_R$ where K_L are the states of P(L), and K_R are the states of the FSA. Change the transition function so that any time it would move from state q_0 to q_1 without reading input,(*) it will move from any $q_0 \times r$ to $q_1 \times r$. If it would move from q_0 to q_1 reading a, change it to

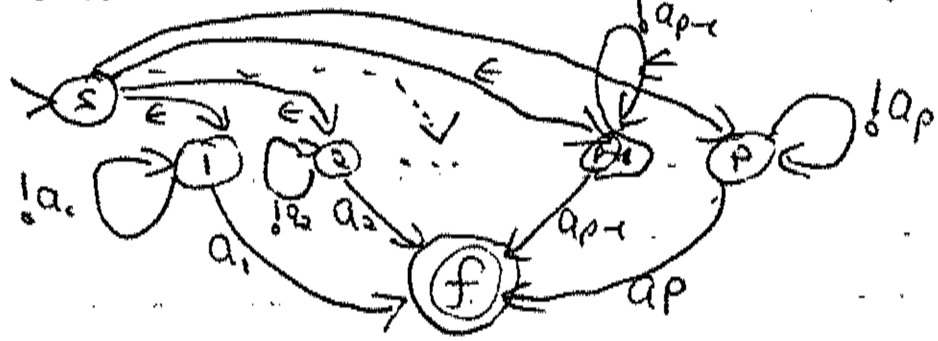
$q_0 \times r \rightarrow q_1 \times s$ where the transition in the FSA when in state r on input a is to s. ~~Keep~~ Let $l \times r$ be a final state in the new PDA if and only if l was final in P(l) and r was final in the DFSA for R.

YES.

(*) Leave any stack push/popping as it was.

#5

a) the Non-det automaton has a start state S, states 1 to P, and final state f. The transitions: from S there is an ε-move to each of the states $1, \ldots, p$ corresponding to guessing the last character. Each state $i \in \{1, \ldots, p\}$ has a transition on a_i to f, and on a_j ~~for~~ s.t. $j \neq i$ back to state i. There are no transitions from f.

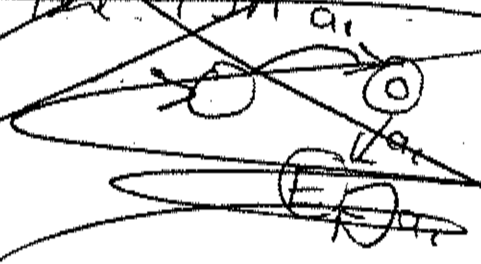


b) at any point when reading the string, it is clearly necessary to know for each i, $1 \leq i \leq p$, whether a_i has been read in the previous input. If you do not keep track of this info, ~~at any time, no DFA can [illegible]~~ i.e. for some i at step n you do not know whether you have read an a_i, you can't correctly process a further input of $a_1 a_2 .. a_{i-1}$ no a_i $a_{i+1} \cdots a_p a_i$ – you ~~is~~ can't determine whether to accept or reject. Thus, at least ~~a~~ state is needed for every subset of $\{1, \ldots, p\}$. Actually, for each subset except the $\overset{?}{\text{proper}}$ subset you need 2 states, 1 final and 1 not final since the first time you see a new symbol (except for the last new symbol) you must be in ~~[illegible]~~ a final state, but the second time you see the symbol you must be in a non-final state.

$$\text{Thus \# states} \geq \underbrace{2^p}_{\substack{\text{\# of subsets}\\ \text{of } \{1,\dots,p\}}} + \underbrace{2^p - 1}_{\substack{\text{\# of proper}\\ \text{subsets of } \{1,\dots,p\}}} = 2^{p+1} - 1$$

GOOD.

~~Note that for p = 1, the FSA~~



#2 The Algorithm: 1st, copy the graph if you want to save it.

NOT NECESSARY

Pick any vertex, V. Do a depth first search from V and remember if V was encountered (i.e. there exists a ~~circuit~~ cycle from V to itself). If there is any such cycle, ~~replace every ver~~ delete every

edge and

vertex in the cycle, and change any edge pointing to that vertex to point to V (you can put them on a stack when you return from the recursive descent, and delete them after the D.F. search is done). Now delete all of the rest of the vertices and

edges

encountered in the DFS (you could have put all vertices encountered on ~~a~~ another stack, now delete from the graph any on the stack that were not deleted as part of a cycle)

Now, V is a possible source. Check the vertex list - if empty, output V. If not, pick any edge pointing to V. Let the other vertex of the edge be W. Now delete V and any edges pointing to it. Repeat the algorithm for W. If there is no edge pointing to V (and thus no W) but there are still vertices in the graph, print out that there is no source.

$O(|V|+|E|)$

To set up the vertex list, vertex array pointing to edges, edge array, etc takes $O(|V|+|E|)$.

The algorithm looks at each edge 1 time in a DFS, and then deletes it. $O(|E|)$ The only other times the algorithm looks at edges is when it changes edges pointing to a vertex found in a cycle in the DFS. $O(|E$ Because DFS will find all cycles passing thru v, each edge will only be changed to point to the root of the DFS 1 time, because the root will never be in another cycle. Also, for each possible source 1 edge will be looked at, $O(|V|)$.

Each vertice may eventually be deleted - $O(|V|)$.

Time is $O(|E|) + O(|E|) + O(|V|) + O(|V|) + O(|E|+|V|) = O(|E|+|V|)$

~~Note: you might have to look a vertices twice~~

Theory Core Exam: Spring 1987

University of California
College of Engineering
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Spring 1987

CS THEORY PRELIMINARY EXAMINATIONS

The six questions count equally. Please write answers in blue books. Brevity and clarity in your answers are important.

GOOD LUCK!!

$(n \times m) \times (m \times p)$

nm^2p

$(t_1 \circ t_2 \ldots t_k) \circ (t_{k+1} \ldots t_n)$

or

$(t_k \circ t_{k+1}) \circ \ldots$

$(A_1 \cdot A_2) A_3 (\ldots A_n)$

$2 \times 3 \quad 3 \times 4 \quad 4 \times 5$

$(2 \times 3 \times 3 \times 4) \times (4 \times 5)$

2×4

$= 2 \cdot 3^2 \cdot 4 + 2 \cdot 4^2 \cdot 5$ or

$= 2 \cdot 3^2 \cdot 5 + 3 \cdot 4^2 \cdot 5$

$72 + 160$

$90 + 240$

1. The value of an arithmetic expression depends upon the order in which operations are performed. For example, depending upon how one parenthesizes the expression

$$5 - 3 * 4 + 6$$

one can obtain any one of the following results:

$$\begin{aligned} 5 - (3 * (4 + 6)) &= -25 \\ 5 - ((3 * 4) + 6) &= -13 \\ (5 - 3) * (4 + 6) &= 20 \\ (5 - (3 * 4)) + 6 &= -\ 1 \\ ((5 - 3) * 4) + 6 &= 14 \end{aligned}$$

Given an unparenthesized expression of the form

$$x_1\ o_1\ x_2\ o_2\ x_3\ \dots\ x_{n-1}\ o_{n-1}\ x_n,$$

where $x_1, x_2, \dots x_n$ are operands with known real values and $o_1, o_2, \dots, o_{n-1}$ are specified operations, we want to parenthesize the expression so as to maximize its value.

(a) Devise an algorithm to solve this problem in the special case that the operands are all positive in value and the only operations are + and *. Show how to apply your algorithm to the expression

$$5 * 8 + 4 * 6. \begin{array}{l} \nearrow 64 \\ \searrow 360 \end{array}$$

(Sketch the algorithm -- don't code.) The running time of your algorithm should be bounded by $O(n^3)$. Assume constant time for operations on reals.

(b) Explain how you would modify your algorithm to deal with the case in which operands can be positive or negative, and the only operations are + and −.

(c) (Optional). Briefly suggest how you would generalize your algorithm to deal with multiplications and divisions. (These operations are a bit nastier that + and − because of sign changes. Don't try to cover all cases. Also don't worry about division by zero; pretend that it never occurs.)

2. Given a tree on n vertices with (positive and negative) edge lengths, we wish to find a pair of vertices such that the path between them has maximum length. Describe (in English -- don't code) an $O(n)$ algorithm for determining such a pair of vertices.

3. Let $\{x_1, x_2,...,x_n\}$ be a set of n distinct keys, where $n = m*2^r$. We wish to partition the set into 2^r disjoint subsets $S_1, S_2,...,S_{2^r}$ of equal size m, such that: whenever $i < j$ then every key in S_i is less than every key in S_j.

   (a) Describe a comparison-based algorithm which does this. (No coding needed -- just explain your strategy clearly.) The more efficient your algorithm (in terms of O), the more credit you'll receive.

   (b) What is the time complexity of your algorithm?

   [Note that keys within each subset need not be sorted. It is possible to solve the problem in less than $O(n\log n)$ time, if $r < \log n$.]

4. Show that SEQUENCING is NP-complete by providing an appropriate transformation from CLIQUE:

CLIQUE

Instance: A graph $G = (V,E)$ and a positive integer k.

Question: Does G contain a clique of size k or more, i.e., a subset V' of V with $|V'| \geq k$ such that every two vertices in V' are joined by an edge in E?

SEQUENCING

Instance: A set of n jobs, each requiring one unit of time for execution, with

(i) a *deadline* d_j for each job $j = 1,2,...,n$,

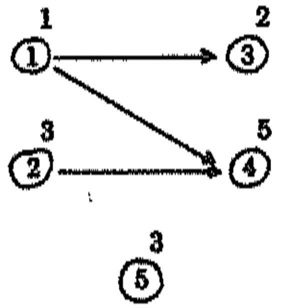
(ii) precedence constraints on the jobs, specified by an acyclic digraph with a node for each job,

(iii) a positive integer K.

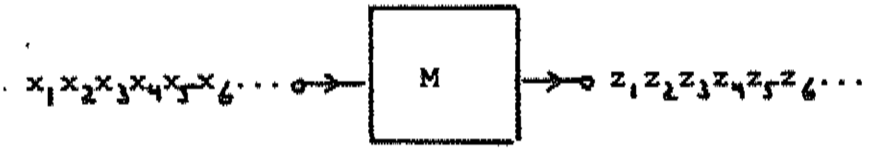
Question: Is there a sequence, consistent with the precedence constraints, for processing the jobs on a single machine so that a least K jobs are completed on or before their deadlines?

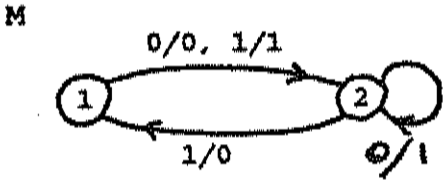
As an example, suppose there are five jobs with precedence constraints specified by the digraph below, with the deadline for each job written by its node in the digraph. Jobs 1 and 2 must be executed before job 4, but only job 1 must be executed before job 3. The sequence 1, 3, 5, 2, 4 results in only job 2 failing to meet its deadline. (Note that the first job in the sequence begins at time $t = 0$.)

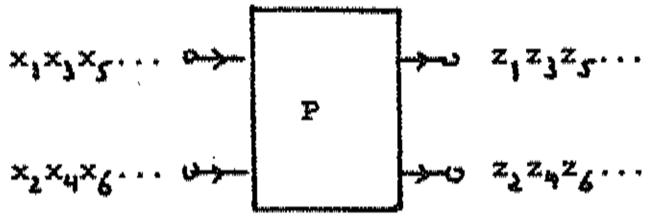
Hint: For an instance of CLIQUE, create an instance of SEQUENCING with a job for each vertex and a job for each edge of G.



5.







M is a Mealy-type finite-state machine whose transition diagram is shown on top right. (A Mealy machine has an output associated with every input.) *P* is a box with two input terminals (each accepting symbols from {0,1}) and two output terminals (each generating symbols from {0,1}). *P* is related to *M* as follows: Let $x_1\ x_2\ x_3\ x_4\ x_5\ x_6$... be an input sequence to *M* which yields the output sequence $z_1\ z_2\ z_3\ z_4\ z_5\ z_6$...; then the sequences $x_1\ x_3\ x_5$... and $x_2\ x_4\ x_6$..., when applied to *P* in parallel, cause the output terminals of *P* to generate $z_1\ z_3\ z_5$... and $z_2\ z_4\ z_6$... (For example, if the input sequence 100111, when applied to *M*, yields the output sequence 111010, then the pair of input sequences 101 and 011, when applied to *P*, yield the pair of output sequences 111 and 100.)

Draw the transition diagram of a Mealy machine which represents *P*.

6. Let *G* be a context-free grammar. We say that a nonterminal *A* is *self-embedding* if and only if there exists a string *uAv*, where *u* and *v* are any strings of terminals and nonterminals, such that

$$A \overset{+}{==>} uAv.$$

(Important Note: u or v, or both, may be empty strings.)

(a) Describe an algorithm to test whether a specific nonterminal of a given context-free grammar is self-embedding.

(b) Show that if G has no self-embedding nonterminal, then $L(G)$ is a regular language.

Theory Core Exam: Fall 1987

University of California, Berkeley
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Fall 1987

CS THEORY PRELIMINARY EXAMINATIONS

The six questions count equally. Please write answers in blue books. Brevity and clarity in your answers are important.

GOOD LUCK !!

1. You are given n items $x_1, \ldots, x_n$. Suppose that most of them are the same. More precisely, define a ***majority item*** as one that occurs more than $n/2$ times. Suppose there is a majority item. The only operation you may perform on items is to compare two of them.
Do only one of the following problems: (you receive credit as indicated)

   a) (half credit) Suppose the result of each comparison is one of "$<$", "$>$", and "$=$". Show how to find the majority item in $O(n)$ comparisons.

   b) (3/4 credit) Suppose the result of each comparison is one of "$=$" and "$\neq$". Show how to find the majority item in $O(n \log n)$ comparisons.

   c) (full credit) Suppose the result of each comparison is one of "$=$" and "$\neq$". Show how to find the majority item in $O(n)$ comparisons.

2. Give an algorithm to determine the length of the longest directed path in a directed acyclic graph with n vertices and m edges.
Credit for this problem depends on the efficiency of your solution. What is the asymptotic running time of your algorithm?

3. Let A be an $n \times n$ matrix whose entries are real numbers. Assume that along any column and along any row of A the entries appear in (increasing) sorted order.

   a) Design an efficent algorithm that decides whether a real number x appears in A. How many entries of A does your algorithm "look at" in the worst case?

   b) Prove a lower bound for the number of elements of A that any such algorithm has to consider in the worst case. (the higher the bound the higher the credit)

4. a) Given a context-free grammar G and a word x, is it recursively decidable (i.e. Turing decidable) whether there exists a word y such that $xy \in L(G)$?

   b) Given a context-free grammar G and a word x, is it recursively decidable (i.e. Turing decidable) whether there exists a word y such that $xy \notin L(G)$?

   Hint: The following problems are undecidable for general context free grammars G and G':

$$
\begin{array}{l}
L(G) \cap L(G') = \emptyset \\
L(G) = \Sigma^* \\
L(G) = L(G') \\
\overline{L(G)} \text{ is CFL} \\
L(G) \cap L(G') \text{ is CFL}
\end{array}
$$

5. Outline a decision procedure for the following problem:

   Input: Regular expression denoting the language L

   Question: Is there a string in L whose reversal is not in L?

   You may use any of the usual textbook algorithms that operate on representations of regular languages as steps in your procedure.

6. Prove that the following problem is NP-complete:

   DOMINATING SET

   Instance: Graph $G = (V,E)$, positive integer $K \leq |V|$.

   Question: Is there a dominating set of size K or less for G, i.e., a subset $V' \subset V$ with $|V'| \leq K$ such that for all $u \in V - V'$ there is a $v \in V'$ for which $\{u,v\} \in E$?

   Hint: Use a reduction that involves SAT.

University of California, Berkeley
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Fall 1987

CS THEORY PRELIMINARY EXAMINATIONS

The six questions count equally. Please write answers in blue books. Brevity and clarity in your answers are important.

GOOD LUCK !!

1. You are given n items $x_1, \ldots, x_n$. Suppose that most of them are the same. More precisely, define a *majority item* as one that occurs more than $n/2$ times. Suppose there is a majority item. The only operation you may perform on items is to compare two of them.
Do only one of the following problems: (you receive credit as indicated)

   a) (half credit) Suppose the result of each comparison is one of "$<$", "$>$", and "$=$". Show how to find the majority item in $O(n)$ comparisons.

   b) (3/4 credit) Suppose the result of each comparison is one of "$=$" and "$\neq$". Show how to find the majority item in $O(n \log n)$ comparisons.

   c) (full credit) Suppose the result of each comparison is one of "$=$" and "$\neq$". Show how to find the majority item in $O(n)$ comparisons.

2. Give an algorithm to determine the length of the longest directed path in a directed acyclic graph with n vertices and m edges.
Credit for this problem depends on the efficiency of your solution. What is the asymptotic running time of your algorithm?

3. Let A be an $n \times n$ matrix whose entries are real numbers. Assume that along any column and along any row of A the entries appear in (increasing) sorted order.

   a) Design an efficent algorithm that decides whether a real number x appears in A. How many entries of A does your algorithm "look at" in the worst case?

   b) Prove a lower bound for the number of elements of A that any such algorithm has to consider in the worst case. (the higher the bound the higher the credit)

4. a) Given a context-free grammar G and a word x, is it recursively decidable (i.e. Turing decidable) whether there exists a word y such that $xy \in L(G)$?

   b) Given a context-free grammar G and a word x, is it recursively decidable (i.e. Turing decidable) whether there exists a word y such that $xy \notin L(G)$?

   Hint: The following problems are undecidable for general context free grammars G and G':

$$L(G) \cap L(G') = \emptyset$$
$$L(G) = \Sigma^*$$
$$L(G) = L(G')$$
$$\overline{L(G)} \text{ is CFL}$$
$$L(G) \cap L(G') \text{ is CFL}$$

5. Outline a decision procedure for the following problem:

   Input: Regular expression denoting the language L

   Question: Is there a string in L whose reversal is not in L?

   You may use any of the usual textbook algorithms that operate on representations of regular languages as steps in your procedure.

6. Prove that the following problem is NP-complete:

   DOMINATING SET

   Instance: Graph $G = (V,E)$, positive integer $K \leq |V|$.

   Question: Is there a dominating set of size K or less for G, i.e., a subset $V' \subset V$ with $|V'| \leq K$ such that for all $u \in V - V'$ there is a $v \in V'$ for which $\{u,v\} \in E$?

   Hint: Use a reduction that involves SAT.

1 (c) Our algorithm is as follows :

(*) We first compare (x_1, x_2) (x_3, x_4) - - - - (x_{n-1}, x_n)

(ie $n/2$ comparisons)

10

If $x_i = x_j$ we send x_i through to the other list (ie new list)

If $x_i \neq x_j$ we reject both and ~~[illegible]~~ dont let them go into new list

We then go back to * with our new list and peform the same operations — this is continued till the list is of constant size say $k = 5$ or $k = 6$ — we then count the # of ~~operations~~ times each element in this short list is represented. The majority item in this list is the majority item for the original list

Proof : We show that ~~[illegible]~~ if x is a majority item in the original list then x remains a majority item in the new list

Let x be the majority item (we know ∃ one)

~~Then comparisons~~ Let # of occurrences of x be $\frac{n}{2} + k$ then non-x's are $\frac{n}{2} - k$ $(k \geq 1)$

Consider the comparison of pairs x_i, x_j there are 3 kinds of comparisons

Type 1 : between ~~xs x~~ and 2 occurrences of x :

~~Each~~ This result in one occ. being eliminated

Type 2 : bet. x and non-x both are eliminated.

Type 3 : bet. non x and non x – either both are eliminated or they are equal and ∴ only 1 is eliminated.
(In the worst case only 1 is eliminated ∴ we will assume that is the case)

Let there be 'p' comparisons of type 1

(Assume n even for Time being)

∴ p x's are sent into next list

but that leaves $\frac{n}{2}+k-2p$ x's left to

be compared in type 2 comparisons

∴ $\frac{n}{2}+k-2p$ non x's ~~and x's~~ are

eliminated This leaves ~~$\frac{n}{2}+k$~~

$\left[\frac{n}{2}-k\right]-\left[\frac{n}{2}+k-2p\right]$ non x's

ie $2p-2k$ non x's

and ½ of these are sent to the other list

∴ $p-k$ are sent to the other list

~~(Clearly there is a restriction p [illegible] should~~

~~ie [illegible] $\frac{n}{2}$ [illegible] p [illegible] $\leq \frac{n}{2}-k$~~

(Note: $p \geq K$ for $p-k$ to be +ve o/w ~~because~~ ~~ten~~ the comparisons are not consistent)

Thus ~~case~~ $k \geq 1 \Rightarrow$ more elements of 'x' type are sent into list than other elements

For n odd we note that we could have e.g.

$\underline{x\ x}\ \underline{x\ x}\ \underline{x\ a}\ \underline{b\ b}$ ⓑ

$x \quad x \quad - \quad b$

~~compare last element to each in the new list~~

or

$\underline{x\ x}\ \underline{x\ x}\ \underline{b\ b}\ \underline{b\ b}\ x$

$x \quad x \quad b \quad b$

∴ compare element to each one in the new list and we ~~go~~ add ~~to~~ it to the list if it is equal to at least half the elements in the list
~~It~~ It can be easily verified that this works in all cases.

Thus ~~when~~ inductively when we perform $O(\log n)$ iterations of list shortening we get the desired result.

Time: - First step: $n/2$ comparisons $\left(+ n/2 \text{ if } n \text{ odd} \right.$
2^{nd} step $n/4$ $\left. + n/4 \text{ if } \frac{n}{2} \text{ odd} \right.$

$$\therefore \quad 2\left[\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots\right]$$

If $n = 2^k$ this is $2\left[2^{k-1} + 2^{k-2} + \cdots + 2^0\right]$
$= 2\left[2^k - 1\right] = 2[n-1]$ $\therefore$ ✓ $O(n)$ Time

2 Find longest directed path in a DAG with n vertices, m edges.

7

~~number each ve[illegible]~~
~~can be done in O(max(|V|, |E|)).~~
~~Call this depth D, [v].~~

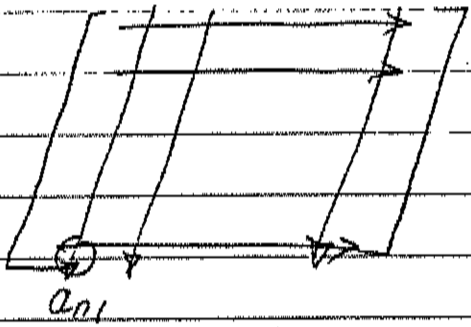
Use a dynamic programming method:
Let C_{ij}^{k} be the longest path from V_i to V_j without passing through vertices numbered larger than k.
would be [illegible] though?

```
So  $C_{ij}^{0} = \begin{cases} 0 & \text{if no edge from } v_i \text{ to } v_j \\ 1 & \text{if there is an edge from } v_i \text{ to } v_j \end{cases}$
  for k from 1 to n   for i,j from 1 to n  do
      $C_{ij}^{k} = \max(C_{ij}^{k-1}, C_{ik}^{k} + C_{kj}^{k})$   $C_{ik}^{k-1}$  $C_{kj}^{k-1}$
  end do
  Sort the $C_{ij}^{n}$'s to find the largest.
```

This will work, since any path through vertices numbered ≤k must either not go through k, in which case the max. length is C_{ij}^{k-1} or must go through k exactly once, in which case the longest path is $C_{ik}^{k} + C_{kj}^{k}$ (length to k and then from k).

There are n^3 C_{ij}^{k}'s and calculating each takes constant time. Sorting the n^2 C_{ij}^{n}'s takes $O(n^2 \log n^2)$ time. Thus, the algorithm is $O(n^3)$

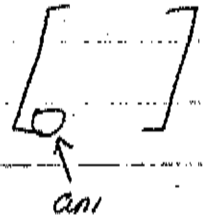
(3) Given A :



10

(a) The algorithm is as follows:

It starts out comparing x to a_{n1} ie

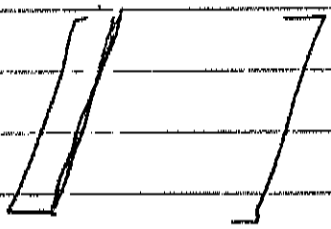


If $x = a_{n1}$ then it stops else if

(i) $x > a_{n1}$ then $x >$ all elements in the 1st column.

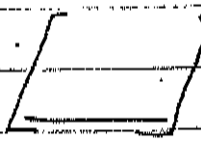
(ii) $x < a_{n1}$ then $x <$ all elements in n^{th} row.

∴ we have in case (i)



and the problem is now essentially the same as before but with n elements less.

Similarly in case (ii) we have :



we ~~I~~ ∴ repeat the same process on the ~~gr~~ smaller matrix looking at the lower left hand element.

Time bound : ~~In the worst case~~ In each iteration at least one row or column is eliminated. There are n rows and n columns — however in the worst case we will proceed along the diagonal ∴ the last element will be a_{1n} which will be both a row and column by itself ∴ $2n-1$ comparison steps required in ~~each case~~ worst case. (in other words $2n-1$ eliminations of rows/columns "remove" entire matrix) ✓

(b) Lower bound : Consider any algorithm which solves the problem. Consider input x then we can create a matrix as :

$$\begin{bmatrix} & & & & x-1 \\ & & & \iddots & x+1 \\ & & \iddots & & \\ & & x-1 & & \\ & x-1 & x+1 & & \\ x-1 & x+1 & & & \end{bmatrix}$$

Now we claim that the algorithm will have to compare x with each element shown (ie on & below diagonal). Clearly any comparison with elements above or below the elements shown will not help in eliminating the shown element positions as possibilities je As assume that some alg. does not compare x with some ele ~~[illegible]~~ shown. Then we make that element = x (instead of x-1 or x+1) and the algorithm will behave just as before and wont find out x is in A ∴ lower bound on # of comparisions is $2n-1$ (# of elements shown)

✓

note (b)

4(b) This is undecidable

∴ suppose M decides it ie
M on input (G, x) says
yes — if $\exists y$ s.t. $G_s \overset{*}{\not\Rightarrow} xy$ (G_s is start symbol of G)
no — if $\forall y \in \Sigma^*$ $G_s \overset{*}{\Rightarrow} xy$

Create M' which decides $L(G) = \Sigma^*$ as follows

M' gives (G, ϵ) to M ← empty string

If M says yes then $\exists y$ s.t. $G_s \overset{*}{\not\Rightarrow} y$ ∴ M' says <u>no</u>

If M says no then $\forall y \in \Sigma^*$ $G_s \overset{*}{\Rightarrow} y$ ∴ M' says <u>yes</u>

4(a). ~~[illegible]~~ <u>This is decidable</u> : Convert the grammar into Chomsky Normal Form (ie all productions $A \to BC$ or $A \to a$ type) ← There are algorithms to do this. Now consider any string of the form $xy = x_1 x_2 \ldots x_n y_1 y_2 \ldots y_m$

If ~~S~~ (start symbol $S \overset{*}{\Rightarrow} xy$ then $\exists$ a derivation in which $S \overset{*}{\Rightarrow} A_1 A_2 \ldots A_n B_1 B_2 \ldots B_m$ (A_i, B_j may be same) ~~[illegible]~~ $\overset{*}{\Rightarrow} xy$ s.t. $A_i \to x_i$ $B_j \to y_j$

So 1st we check if $\forall x_i$ $\exists A_i \to x_i$ and then ~~[illegible]~~ for all such A_i we have to check (over) if ~~B~~ $S \Rightarrow A_1 A_2 \dots A_n \alpha$

This can easily be done by moving backwards from the A_i. (or alternately by ~~s~~ starting at S) ~~[illegible]~~ if $S \overset{*}{\Rightarrow} A_1 A_2 \dots A_n \alpha$ and there exists an α such that α is composed of terminals and useful non-terminals (useful non-terminal derives a string of non-terminals) then clearly $\exists y$ s.t. $xy \in L(G)$ If not ~~then~~ ie. $S \overset{*}{\not\Rightarrow} A_1 A_2 \dots A_n \alpha$ (α-useful) then clearly $\forall y \in \Sigma^*$ $xy \notin L(G)$ and the algorithm answers <u>No</u>.

-2 DETAILS MISSING

5. 1 Given a regular expression generating L, a nfa can easily be constructed, algorithmically, which accepts L.

2. This nfa can be converted algorithmically to a dfa, which accepts L

3. Given dfa: start state s
final states {F}
states {S}
transitions $\{(s_1, s_2, k)\}$ where $k \in \Sigma$

Construct nfa: start state $\circledS$
final state {s}
states {S}
transitions $\{(s_2, s_1, k)$ where (s_1, s_2, k) is in the original dfa$\} \cup \{(\circledS, f, e) \mid f \in F\}$,

That is, the new nfa has all the arrows reversed, a start state going to the original final states, and a final state that is the original start state.

This nfa accepts L^R where L = language accepted by the dfa. This is clear, since the nfa follows the same transitions in the original dfa, but in the reverse order.

4. Now convert the nfa to a dfa, accepting L^R.

5. Construct a dfa to calulate the complement of the dfa in 4, i.e. accepting $\overline{L^R}$

6 Construct a dfa to calculate the dfa in 5 intsected with the dfa in 1.

This dfa accepts $L \cap \overline{L^R}$

7. Test if the dfa in 6 accepts any string.

This is a decision procedure to test if there is a string in L and in $\overline{L^R}$, Thus, this is a decision procedure to test if there is a string in L whose reversal is not in L

6 Prove the dominating set problem (DOM) is NP-complete.

Proof:

1. Claim: DOM is in NP.
Proof:
Given: $G=(V,E)$, k
Select nondeterministically $k_1 \leq k$ call this V_1.
Select nondeterministically k_1 distinct vertices,
For each vertex in $V-V_1$, call it v
   Check each vertex in V_1 to see if there is any edge (v,v') with $v' \in V_1$
If each vertex in $V-V_1$ has a vertex $v' \in V_1$ with edge (v,v') then DOM is satisfied.

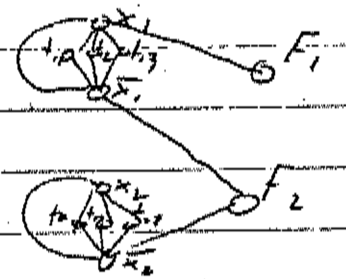
This can be done in polynomial time, so DOM is in NP.

2. Claim: DOM is NP-complete.
Pf:
It is known that SAT is NP-complete, so it is only necessary to show a reduction from SAT to DOM.

Given: A CNF boolean equation E to be satisfied. (any boolean equation can be put in CNF form in polynomial time)

Suppose the equation has K variables, $\{x_1, \ldots, x_k\}$ and n factors $F_1, \ldots, F_n$. So $E = F_1 \cdots F_n$ where F_i is the sum of terms x_i or $\bar{x}_i$

Construct the graph G with:
k vertices labelled $x_1, \ldots, x_k$
k vertices labelled $\bar{x}_1, \ldots, \bar{x}_k$ and k edges $(x_i, \bar{x}_i)$
k^2+k vertices labelled t_{ij}, $i \leq k, j \leq k+1$
n vertices labelled $E_1, \ldots, E_n$
$2k^2$ edges (x_i, t_{ij}), $(\bar{x}_i, t_{ij})$, $i \neq j \leq k+1$
$<kn$ edges (x_i, F_j) or $(\bar{x}_i, F_j)$ where x_i or $\bar{x}_i$ appears in F_j
This transformation can clearly be done in poly. time.
(So the graph will look like:
e.g.



for $E = (x_1)(\bar{x}_1 + \bar{x}_2)$

)

Let K for the dominating set problem $= k$.
Thus, a SAT problem can be xformed to a DOM problem in poly time
Now suppose there is a solution to DOM on the above graph.
Lemma 1. There must be at least one vertex from each set $\{x_1, \bar{x}_1\}$, $\{x_2, \bar{x}_2\}, \ldots$ in the dominating set.
Pf. If there isn't, there are K+1 vertices $t_{i1}, \ldots, t_{ik+1}$ with no neighbor in V'. These can't all be in V', so there is at least one not dominated. Contradiction.

Lemma 2: There must be exactly one vertex from each of $\{x_1, \bar{x}_1\}, \{x_2, \bar{x}_2\} \dots$ in the dominating set V'.

Proof: Pidgeonhole principle. There are k sets $\{x_i, \bar{x}_i\}$, k vertices, at least one vertex in each set.

∴ Since there are at most k vertices in the dominating set, the dominating set is exactly one vertex from each set $\{x_i, \bar{x}_i\}, i = 1, \dots, k$.

Thus, each vertex $F_1, \dots, F_k$ must have a neighboring vertex in V' since $F_1, \dots, F_k \notin V'$. But there will only be a neighbor if x_i or $\bar{x}_i$ is in V' and x_i or $\bar{x}_i$ respectively is a term in F_j. Thus, each term F_i will be true in the corresponding SAT problem if the values in V' are assigned true. So each F_i is true, so the satisfiability problem is satisfied.

∴ A solution to DOM ⇒ solution to SAT.

Given a solution to SAT, let V' = variables or complements assigned true. There will be k of these since the SAT problem has k variables. It is clear that all vertices labelled $x_i, \bar{x}_i, t_{iQ}$ will be in V or neighboring a vertex in V'. Also every vertex F_i must have a neighboring vertex in V' since the SAT problem

is satisfied. Thus, this is a solution to DOM
∴ Solution to DOM $\Leftrightarrow$ solution to SAT.
∴ Since SAT is NP-complete and the transformation
SAT $\Rightarrow$ DOM is polynomial, DOM is NP-complete.

Theory Core Exam: Spring 1988

Spring 1988

CS Undergraduate Theory Preliminary Examination

* Do not turn the page before you hear the starting gun.

* In the meantime . . . Please print your I.D. number here:______________________

* This examination contains five questions. The symbol 💡 appearing by Problems 2B, 4C, and 5 indicates above-average difficulty.

* You are to put all your work, including scratchwork, on the pages of this examination.

* The examination is closed book: you may not use any textbooks, notebooks, bluebooks, or other written material that you have brought into the examination room with you. Calculators, though unnecessary, are permitted.

* You have three hours in which to work.

* Partial credit will be given for all problems based on your reasoning.

[Each of the parts below carries equal weight
Do 6 of those 7 parts.
If you answer all 7, your grade will be based on the best 6.]

Problem	Page	Grade
1	3	
2A	5	
💡 2B		
3	7	
4AB	9	
💡 4C		
💡 5	11	

TOTAL: ________

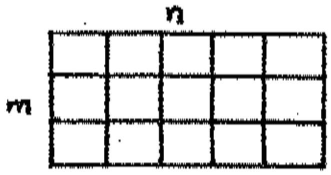
MAX = 60

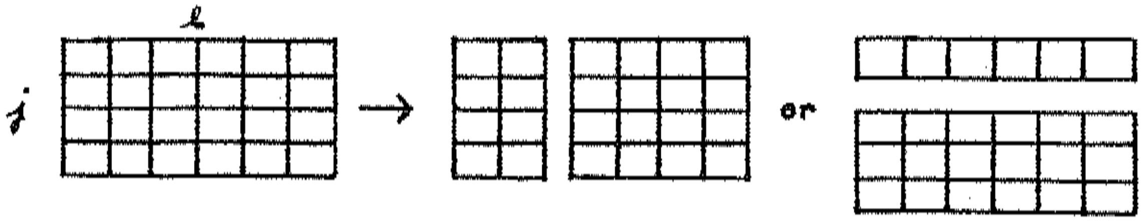
GOOD LUCK

PROBLEM 1

CHOCOLATE BAR PROBLEM

You are given an m x n Hershey Bar which you are to crack into mn 1 x 1 pieces. An elementary move (step) takes a j x ℓ piece and cracks it along a vertical or horizontal edge:





How many steps does it take to completely reduce the m x n bar to 1 x 1 pieces?

PROBLEM 2

A. Draw the TRANSITION DIAGRAM of the MINIMAL, DETERMINISTIC finite-state automata A1, A2 and A3 which accept the following sets (respectively):

(i) $R1 = \{0,1\}^*\{1\}$

(ii) $R2 = \{00,01,10,11\}^*$

(iii) $R3 = R1 - R2$ (i.e., all strings that are in R1 but not in R2)
(Remember to minimize!)

B. Let $R \subseteq \{0,1\}^*$ be a regular set. Define $\sqrt{R} = \{x \in \{0,1\}^* \mid xx \in R\}$.
Is $\sqrt{R}$ regular?

PROBLEM 3

Use the NP-completeness of HAMILTON CYCLE to prove that if $P \neq NP$ then HAMILTON PATH $\notin P$.

HAMILTON PATH

INPUT: A graph G.

QUESTION: Does G have a Hamilton path?

A Hamilton path is a path that starts at some node u, ends at a different node v, and goes through all other nodes once and only once.

PROBLEM 4

Insert the language classes given in A into the table in B in order so that each language class is contained in the class immediately below it. Then fill in each entry of table B with $\boxed{\text{YES}}$, $\boxed{\text{NO}}$ or $\boxed{?}$ (if you don't know).

A. Language Classes:
1. P (polynomial time bounded)
2. Regular
3. Context free
4. Recursively enumerable
5. NP (nondeterministic polynomial time bounded)
6. Recursive
7. PSPACE (polynomial space bounded)

B. Is the given class of languages closed under the given operation?

LANGUAGE CLASS	INTERSECTION $\cap$	UNION $\cup$	COMPLEMENTATION (Σ^*-L) $\neg$

C. Prove your answers for $\cap$, $\cup$, $\neg$ in the case of context free languages.

PROBLEM 5

You are given a weighted graph G = (V,E,W) and a minimum spanning tree T of G, both given by adjacency lists. Suppose the weight of 1 edge of G is changed. How would you update the spanning tree?

Notice that there are 4 types of update. Each type of update should be as efficient as you can make it. In particular, O(|V|) is better than O(|E|).

Without loss of generality, you may assume that all edge weights are always different.

Addendum to Problem 5.

More formally you may assume that the single change in weight is specified as [u, v, edge-type, old-weight, new-weight] where (u,v) is the edge whose weight is changed and edge-type specifies whether (u,v) is a tree-edge or not (i.e. (u,v) $\in$ T?).

We may classify the change as being one of 4 kinds: according to whether (u,v) $\in$ T or not and whether the weight of (u,v) increases or decreases. In each of the four cases your algorithm should be as efficient as possible; you can specify the new minimum spanning tree T' by specifying how it differs from T (i.e. output T-T' and T'-T).

Spring 1988

CS Undergraduate Theory Preliminary Examination

* Do not turn the page before you hear the starting gun.

* In the meantime . . . Please print your I.D. number here: 14 .

* This examination contains five questions. The symbol 💡 appearing by Problems 2B, 4C, and 5 indicates above-average difficulty.

* You are to put all your work, including scratchwork, on the pages of this examination.

* The examination is closed book: you may not use any textbooks, notebooks, bluebooks, or other written material that you have brought into the examination room with you. Calculators, though unnecessary, are permitted.

* You have three hours in which to work.

* Partial credit will be given for all problems based on your reasoning.

[Each of the parts below carries equal weight
Do 6 of those 7 parts.
If you answer all 7, your grade will be based on the best 6.]

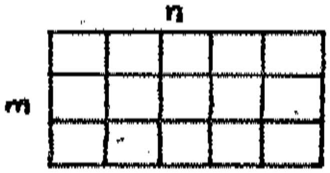
Problem	Page	Grade
1	3	10
2A	5	10
💡 2B		10
3	7	10
4AB	9	(9)
💡 4C		10
💡 5	11	10
TOTAL:		60

MAX = 60

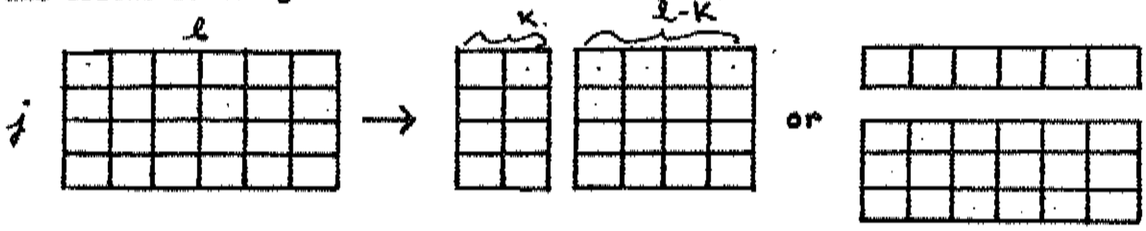
GOOD LUCK

PROBLEM 1

CHOCOLATE BAR PROBLEM



You are given an m x n Hershey Bar which you are to crack into mn 1 x 1 pieces. An elementary move (step) takes a j x l piece and cracks it along a vertical or horizontal edge:



How many steps does it take to completely reduce the m x n bar to 1 x 1 pieces?

Either break a bar j x l into position K vertically or into position K horizontally.

$$\cancel{C(j,l)} = \min_{K\le} \left\{ \begin{array}{l} \cancel{\{C(j,K) + C(j,l-K)\}} \\ \{C(K,l) \cancel{+ C(j-K,l)} \} \end{array} \right\},$$ - vertical break.

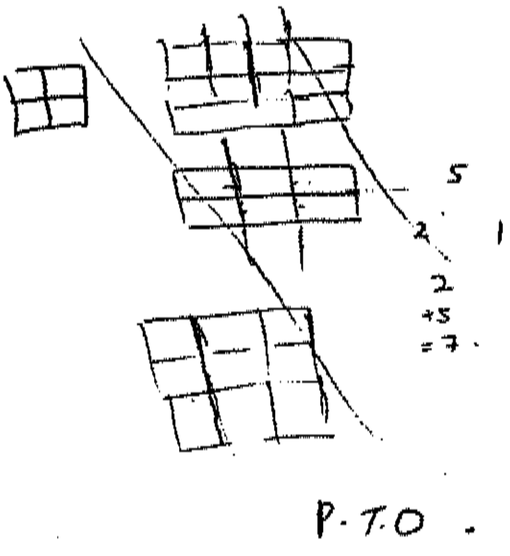
Hence.

$$c(j,l) = \min . \left[\begin{array}{l} \min\limits_{1\le K\le l-1} \{ c(j,K) + c(j,l-K) \}, \\ \min\limits_{1\le K\le j-1} \{ c(K,l) + c(j-K,l) \} \end{array} \right] + 1$$ ← vertical break ← horizontal break

/if $j>1$ & $l>1$ /

else $C(1,l) = l-1$ & $C(j,1) = j-1$. because of the symmetry of the situation, ~~C(K,l) = C(~~[illegible] It does not matter if we break at K or at $j-K$

$C[1,1] = 0$.
$C[1,2] = C[2,1] = 1$.
$C[3,1] = C[1,3] = 2$.
$C[2,2] = 3$.
$C[3,2] = 5$
$C[3,3] = 8$



P.T.O.

C[j j]. BLANK PAGE

j	1	2	3	4
K 1	0	1	2	3
2	1.	3.	5	
3	2			
4	3			

Answer $C(m,n) = mn - 1$.

proof by induction, on the area of the Hershey bar. holds for $j = l = 1$. ~~(area = 1)~~. i.e. assume holds for all ~~values~~ bars of lesser area ~~[illegible]~~ with product $jl' < jl$.

$$C(j,l) = \min\{\min(jk - 1 + j(l-k) + 1,\ \min(kl - 1 + jl - kl - 1)] + 1.$$

$$= (jl - 2) + 1$$

$$= jl - 1.$$

∴ ~~Answer (mn - 1)~~

Proof :- by induction on area of bar. for bar of area 1, we need zero breaks, $= 1 \times 1 - 1 = 0$. for bar of area jl, we ~~[illegible]~~ have (assuming result for bars of lesser area)

$$C[j,l] = \min\left[\min_k [jk - 1 + jl - jk - 1],\ \min_k [kl - 1 + jl - kl - 1]\right\} + 1.$$

holds by inductive hypothesis on

$$= \min(jl - 2,\ jl - 2) + 1 = jl - 2 + 1 = jl - 1.$$

Ans. $C(m,n) = mn - 1$.

(From the solution, it is apparent that it does not matter where we break & in what direction first).

PROBLEM 2

A. Draw the TRANSITION DIAGRAM of the MINIMAL, DETERMINISTIC finite-state automata A1, A2 and A3 which accept the following sets (respectively):

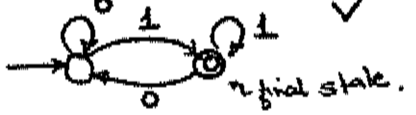
(i) $R1 = \{0,1\}^*\{1\}$,

(ii) $R2 = \{00,01,10,11\}^*$

(iii) $R3 = R1 - R2$ (i.e., all strings that are in R1 but not in R2) (Remember to minimize!)

B. Let $R \subseteq \{0,1\}^*$ be a regular set. Define $\sqrt{R} = \{x \in \{0,1\}^* \mid xx \in R\}$. Is $\sqrt{R}$ regular?

(i)



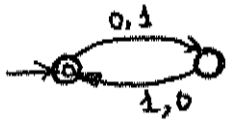
✓

All strings ending in 1.

◎ - final states

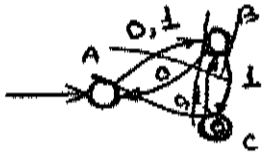
→○ - start states.

(ii) This is all even strings. (Assuming $\Sigma = \{0,1\}$).

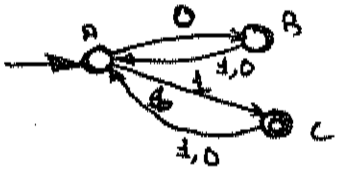


✓

(iii) All ~~even~~ odd strings ending with a 1.



- ~~[illegible]~~



✓

A = even strings

B = odd strings with 0.

C = odd strings with 1.

A&B are distinct due to their transition on 1, A goes to final state, B goes to non final.

B. Yes, the set $\sqrt{R}$ is regular.

giving a Non deterministic F.S.A for it.

Given the machine M for R, guess, non deterministically the state it will be in when you will see ~~[illegible]~~ end of the string.

ϵ = empty string.

Simulate x on the m/c & accept if final state matches.

$$\sqrt{R} = \{x \mid xx \in R\}.$$

More formally,

~~let Q = the set of states of M accepting R~~

Let M machine accepting R be.

$$(\underset{\text{states}}{Q}, \underset{=\{0,1\}}{\Sigma}, \overset{\text{transition}}{\delta}, \underset{\text{final state,}}{F}, \overset{\text{start state.}}{q_0})$$

M' will consist of. $(Q', \Sigma, \delta', F', \underset{\text{start state}}{S})$.

Where $Q' = Q \times Q \times Q \cup \{S\}$ triplets of states of Q,

first ~~state~~ one → simulates M on x.

Second state make a guess as to which state it will be after finding x.

Third state starts simulating on x the second phase, starting from the guess.

and δ' is now.

$$\delta'([q_1, q_2, q_3], \overset{a \in (0,1)}{a}) = [\delta(q_1, a), q_2, \delta(q_3, a)]$$

$$= ~~\delta[\delta(q_1, a)]~~$$

also add a new start state S with transition on empty string.

$$\delta'(S, \underset{\text{em}}{e}) = [q_0, q, q] \text{ for all } q \in Q.$$

/* non-deterministically choose the state at end of string */

finally our accepting states are as ~~[illegible]~~.

$(\in F')$ $[q_1, q_2, q_3]$ is accepting iff $\underset{(\text{guess works})}{q_1 = q_2}$ & $\underset{(xx \in R)}{q_3 \in F}$.

It is obvious that if xx is in the laguage M, there is at least one choice that leads to the final state.
So if M* accepts a string xx, M' accepts x.

Assume M' accepts, x, then say it had guessed the state q'. M after seeing the string x had landed into q' since $\delta(q_0, x) = q'$ δ- extended to strings.
also $\delta(q', x) = q_3$ say & $q_3 \in F$ by our construction.
$\therefore \delta(q_0, xx) = q_3 \in F$ & hence M accepts the string xx.

N.B. Machine M' is described on facing page. ✓

PROBLEM 3 (abb. H.C.).

Use the NP-completeness of HAMILTON CYCLE to prove that if P ≠ NP then HAMILTON PATH ∉ P.

HAMILTON PATH (abbreviate to H.P.)

INPUT: A graph G.

QUESTION: Does G have a Hamilton path?

> A Hamilton path is a path that starts at some node u, ends at a different node v, and goes through all other nodes once and only once.

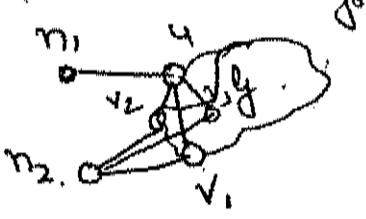
To show that P ≠ NP ⇒ H.P ∉ P, it suffices to show that H.P. is N.P. Complete.

First of all. H.P. is in N.P. given the two nodes u and v, we guess the |V|. edges in sequence and check that all the edges do really form a Hamitogian Path. This is just matching endpoints & ~~[illegible]~~ start point of next edge. And is obviously polynomial

To show. H.P. is N.P. Hard.

given an instance of H.C. reduce it to H.P. as below. (given in H.C.)

take ~~two~~ ~~adjacent~~ any vertices u in the graph $\mathcal{G}$. Add two new vertices n_1 & n_2 to the ~~two adjacent vertices (u & v)~~ graph. n_1 is connected to u and n_2 to all vertices adjacent to u. Call these vertices $v_1, v_2 \ldots v_K$. where K = degree of u.



= $\mathcal{G}'$

I show that a H.P. exists in $\mathcal{G}'$ iff a H.C. exists in $\mathcal{G}$.

pf: Suppose $\mathcal{G}$ has a H.C. Without loss of generality, assume it starts at u. It reenters u through one of its neighbouring vertices v_i. Construct a H.P

$\langle n_1, u \rangle, \langle$H.C. from u to $v_i, \rangle \langle v_i, n_2 \rangle$.

Conversely assume a H.P. exists from n_1 to n_2.

It has to be $<n_1, u>$ (a path covering all vertices of G). $<v_i\ n_2>$ for some v_i adjacent to u.

Connect this v_i to u & get a H.C.

This construction is clearly polynomial.

Hence H.P. is N.P. Complete. If H.P. $\in$ P then all elements in NP can be reduced to H.P in polynomial time and hence solved in polynomial time ~~s~~implying P = NP. But this goes contrary to assumption $P \neq N.P.$

hence H.P. $\notin$ P.

PROBLEM 4

Insert the language classes given in A into the table in B in order so that each language class is contained in the class immediately below it. Then fill in each entry of table B with $\boxed{\text{YES}}$, $\boxed{\text{NO}}$ or $\boxed{\text{?}}$ (if you don't know).

A. Language Classes:
1. P (polynomial time bounded)
2. Regular
3. Context free
4. Recursively enumerable
5. NP (nondeterministic polynomial time bounded)
6. Recursive
7. PSPACE (polynomial space bounded)

B. Is the given class of languages closed under the given operation?

LANGUAGE CLASS	INTERSECTION $\cap$	UNION $\cup$	COMPLEMENTATION (Σ^*-L) $\neg$
Regular.	Yes	Yes	Yes.
Context Free.	No	Yes	No.
~~Recursive~~ P	Yes.	Yes	Yes.
NP	Yes	Yes	?
PSPACE	Yes	Yes	?
Recursive.	Yes	Yes	Yes.
Recursively enumerable.	~~Yes~~ Yes	Yes	No.

C. Prove your answers for $\cap$, $\cup$, $\neg$ in the case of context free languages.

(i) C.F.L is closed under Union. ✓
Assume L1 & L2 are C.F.Ls.
let G_1 & G_2 be grammars, generating L_1 & L_2, with start symbols S_1 & S_2
give a new grammar ~~take~~ with a new start symbol. S.
& add productions. $S \rightarrow S_1$, $S \rightarrow S_2$ to already existing grammars G_1 & G_2.

Assuming all ~~term~~ nonterminals of G_1 & G_2 are distinct (Otherwise rename), we get the result. The new grammar generates all words in L_1 & all words in L_2 & exactly these words.

(ii) C.F.L. is not closed under intersection,

proof

✓ consider $L_1 = \{a^i . b^j c^k \mid j = k\}$ is CFL.

$L_2 = \{a^i b^j c^k \mid i = j\}$ is CFL.

but $L_1 \cap L_2 = \{a^i b^j c^k \mid j = k = i\}$ is not a C.F.L. (see below).

(iii) C.F.L not closed under complementation,

if it was closed under complementation then,

✓ $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ would also be a CFL.

but not closed under intersection.

Proof that $a^i b^j c^j$ is CFG. consider grammar

$S \rightarrow AB.$
$B \rightarrow bBc \mid \epsilon.$
$A \rightarrow aA \mid \epsilon.$

$a^i b^i c^j$ is CFL, consider grammar.

$S \rightarrow AC$
$A \rightarrow aAb \mid \epsilon.$
$c \rightarrow cC \mid \epsilon.$

✓

and $a^i b^i c^i$ is not a CFL by pumping lemma, by the pumping lemma, (uvwxy theorem) let n be the constant of the pumping lemma, choose the string $a^n b^n c^n$. we have ~~[illegible]~~. $|vwx| \leq n$. So v & x can overlap atmost two of the three groups of a, b and c. whichever it may be, if we pump it (v & x) more than once, the number of the elements in the third group ceases to be equal.

u,v is edge that changes;
T is old. m.s.t.

10

PROBLEM 5

You are given a weighted graph G = (V,E,W) and a minimum spanning tree T of G, both given by adjacency lists. Suppose the weight of 1 edge of G is changed. How would you update the spanning tree?

Notice that there are 4 types of update. Each type of update should be as efficient as you can make it. In particular, O(|V|) is better than O(|E|).

Without loss of generality, you may assume that all edge weights are always different.

Assuming edges are neither created nor deleted.

Two cases preserve the m.s.t. property.

- edge is in T & weight of edge decreases.
- edge is not in T & weight of edge increases.

Otherwise.

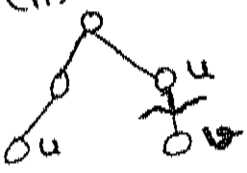
(i) edge is not in T and weight of edge ~~increases~~ decreases. Add this edge to ~~[illegible]~~ T. This creates a cycle. Scan this cycle and break the edge with the greatest weight. Thus you get new tree T'.

(ii) edge is in T and its weight ~~[illegible]~~ increases. Break this edge, this breaks up T into two sets T_1 and T_2. find the minimum cost edge from a ~~element~~ node in T_1 to a ~~element~~ node in T_2. add this edge to form the new tree T'.

Case (i)

Complexity: The detection of the cycle will require ~~[illegible]~~ a d.f.s. with only tree edges, (of which |V|-1 are (u,e)) so the complexity is O(|V|) also the finding of ~~minimum~~ maximum weight edge can be done in O(|V|) time.

Case (ii)



Say (u,v) ~~[illegible]~~ ~~[illegible]~~ is the relevant edge. Without loss of generality assume that u is father of v. When we break the edge, T_2 will contain the descendents of v. A scan of the tree to find the min. along the back edges will be O(e) in

The correctness of the ~~proof~~ algo. is based on the following lemma.

→ if T_1 is a set of nodes (partial tree) & T_2 is another set of nodes (partial tree), then the minimum spanning tree includes the edge with the least cost between T_1 & T_2. The proof is similar to the steps of algo in case (i) edge not in T & the weight decreases.

The complexity of case (ii) Tree edge - wt increases be reduced by this way

→ arrange the adjacency list representation of the edges of graph G as a min-heap.

then we have → distribute the graph into T1 & T2 $O(||V||)$.

→ choose the one with fewer ~~edges~~ nodes.

→ traverse this node, selecting the edge at top of the min heap

/* This may be an edge into T2 itself (say T2 was chosen), but we have to skip that. This will bring the ~~ex~~ worst case time to $O(|E|)$ but expected time to $O(|V|)$ */

→ select the minimum $O(||V||)$

→ output this edge replacing edge (u-v) (the one that changed).

Thus expected time is $O(||V||)$ although worst case is $O(||V||+||E||)$

↑

excellent!
using this ordering you can get worst case $O(|V|)$ time by a more careful analysis!

THEORY CORE EXAM
FALL 1988

Fall 1988

CS Undergraduate Theory Preliminary Examination

- Do NOT turn the page before you hear the starting gun.
- In the interim, please print your ID number here:____________________
- This is a closed book exam with SIX questions. Blank pages are included between some questions, so make sure you read all of them.
- Put all your answers, including your reasoning, on the pages of this examination. Partial credit can be given if you show your work.
- All questions carry equal points, but are not guaranteed to be of equal difficulty.
- You have three hours to answer all six questions.
- Good Luck !

1. Let G be a directed graph with n vertices and e edges. The *transitive closure* of G is a graph $H \supseteq G$ with the same vertices as G such that $u \rightarrow v$ is an edge of H if and only if there is a directed path in G from u to v.

   (a) If G is an *acyclic* directed graph, give an algorithm for computing its transitive closure that runs in time $O(ne)$.

   (b) Assuming an $O(ne)$ algorithm for acyclic digraphs, give an algorithm that computes the transitive closure of a general digraph in time $O(ne)$.

   You can assume the existence of efficient algorithms for the following problems:

   TOPOLOGICAL SORTING

   Given a directed, acyclic graph G with n vertices, a topological sort is a one-to-one, onto function $f : V(G) \rightarrow \{1, \ldots, n\}$, such that whenever $u \rightarrow v$ is an edge of G, we have $f(u) < f(v)$. An acyclic digraph can be topologically sorted in time $O(n + e)$.

   STRONG COMPONENTS

   Given a directed graph G, a *strongly connected component* is a maximal subgraph $F \subseteq G$, such that for every pair of vertices u and v in F, there is a path from u to v. The strongly connected components of a directed graph can be computed in time $O(n + e)$.

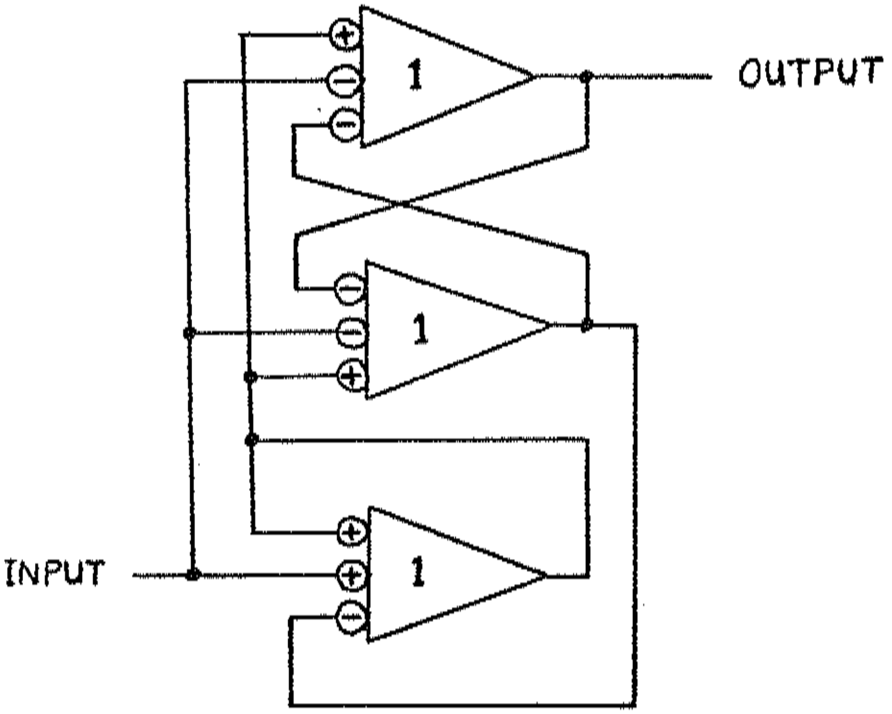
2. Given an unsorted list of real numbers $x_1, x_2, \ldots, x_n$, the CLOSEST PAIRS PROBLEM is to compute a function $c : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that $x_{c(i)}$ is the closest number in the list to x_i. In other words $c(i) = j$ where $j \neq i$ and $|x_i - x_j|$ is minimized.

   Now consider a computational model where comparisons are allowed between x_i's, and between differences of x_i's. Give a lower bound on the worst-case running time for any algorithm which solves the closest pairs problem, i.e. which computes $c(i)$, $i = 1, 2, \ldots, n$, in this model.

3. The figure below shows a "neural" network. Each neuron (triangle) can output either a 0 or a 1. The output is 1 iff a weighted sum of the input values is at least as great as a threshold, which is the number inside the triangle. The inputs are either *excitatory* (weight = 1) which are represented as circles containing plus signs, or *inhibitory* (weight = −1) which are shown as circles with minus signs. The input to the network is either 0 or 1 at all times, and initially, all neurons have zero output.

   (a) Construct an equivalent finite state machine which is in an accepting state whenever the output of the network is a 1. Assume a small delay through the neurons, but compute only the stable states of the network, i.e. when the input changes, follow the signals through the network until a steady state is reached. You will find that the network is symmetric enough that some transitions will be non-deterministic. (Checksum: your non-deterministic machine should have 3 states)

   (b) Give an equivalent *deterministic* finite state machine, and minimize the number of states.



4. Given a language L_0, let L_1 denote the language that contains all strings in L_0, plus all strings that can be obtained by substituting a different symbol in one position. For example, if L_0 is a language over the symbols a, b, c and

$$L_0 = \{a, abb\}$$

then

$$L_1 = L_0 \cup \{b, c, bbb, cbb, aab, acb, aba, abc\}$$

(a) If L_0 is regular, does it follow that L_1 is also regular ? Prove or disprove.

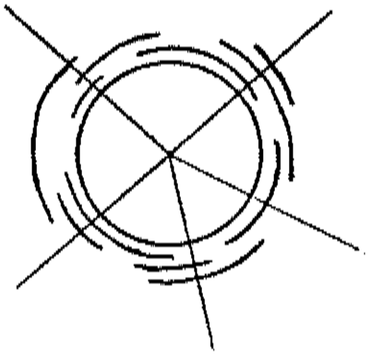
(b) If L_0 is context-free, does it follow that L_1 is context-free ? Prove or disprove.

5. Let $[a_i, b_i]$, $i = 1, 2, \ldots, n$ be n closed intervals on the real line. We say that a set S of points *covers* the intervals if

$$[a_i, b_i] \cap S \neq \emptyset, \quad \text{for } i = 1, 2, \ldots, n$$

(a) Describe an efficient algorithm for finding a covering set of minimum cardinality. Estimate the worst-case running time in big-oh notation.

(b) Now suppose each interval is an arc of a circle. The covering set we are looking for is a finite set of rays. as shown below. Describe an efficient algorithm for finding a minimum covering set and estimate its worst-case running time.



6. Next we generalize the covering problem to disconnected sets. In contrast to question 5, which concerned single intervals, we now consider sets A_i which are unions of two intervals in the real line, $A_i = [a_i, b_i] \cup [c_i, d_i]$, and we seek a set S of points which covers the A_i, so that

$$A_i \cap S \neq \emptyset \quad \text{for } i = 1, 2, \ldots, n$$

Show that deciding if there is a k-point covering set is NP-complete. You may want to use the fact that the VERTEX COVER problem is NP-complete:

VERTEX COVER

INSTANCE: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.

QUESTION: Is there a *vertex cover* of size k or less for G, that is, a subset $V' \subseteq V$ such that $|V'| \leq k$ and for each edge $\{u, v\} \in E$, at least one of u and v belongs to V'.

Fall 1988 theory
prelim - answers

1(a)
First, use topological sort to order the vertices of G. Let sort[i] be the ith vertex in the ordering, and let in[v] be a list of the in-neighbors of the vertex v, i.e. the vertices u such that u -> v is an edge of G. For each vertex v, we create a list reach[v] of the vertices that can reach v. Then to compute the transitive closure, we do

```
for i = 1 to n do
  v := sort[i]
  reach[v] := in[v]      ; vertices that can reach v include its in-neighbors
  for u in in[v] do
    reach[v] := reach[v] + reach[u] ; plus the vertices that can reach its
    endfor                          ; in-neighbors
  endfor
```

Then H is the graph whose vertices are the vertices of G, and whose edges are all edges of the form u -> v, where u is in reach[v].

We assume inductively that reach[u] contains all the vertices that can reach u for any u less than v in the ordering, and it follows that after the ith step, reach[v] will contain all vertices that can reach v.

The algorithm runs in time O(ne) since the inner for is executed exactly e times, once for each edge of G, and it involves a set union of two vertex sets containing at most n vertices, an O(n) operation.

(b)
For a general digraph G, first find the strongly connected components of G, and form its superstructure graph G'. The vertices of G' are the strong components of G, and there is an edge between two vertices v0 -> v1 of G' iff there is an edge between two vertices u0 -> u1 of G, such that u0 lies in the strong component v0, and u1 lies in v1. G' is clearly acyclic.

Now compute the transitive closure of G' using the algorithm from part (a), and let H' be the result. Then we compute

```
vertices(H) := vertices(G)

for v in vertices(H') do   ; join all pairs of vertices within a strong
  for u0 in v do           ; component with arcs in both directions.
    for u1 in v and u1 <> u0 do
      edges(H) := edges(H) + (u0 -> u1)
      endfor
    endfor
  endfor

for e in edges(H') do     ; Then we add an edge u0 -> u1 whenever u0 and u1
  v0 := tail(e)           ; lie in distinct strong components, and there is
  v1 := head(e)           ; an edge between these components in H'
  for u0 in v0 do
    for u1 in v1 do
      edges(H) := edges(H) + (u0 -> u1)
      endfor
    endfor
  endfor
```

1(b) (contd.)
Computing the strong components and the superstructure G' takes time O(n+e). Now since G' has at most as many edges and vertices as G, computing its transitive closure takes time O(ne). The last step is the computation of H from H'. But if we look at the two inner loops where edges of H are added, it is not difficult to see that a different edge u0 -> u1 is added each time through the loop. This follows because the strong components partition the vertices of G, and distinct strong components will contain disjoint sets of vertices. So the running time of this step is bounded by the number of edges in the transitive closure, H.

We claim that H has size O(ne).

ASIDE: clearly H has size O(n^2) but this may be larger than O(ne). The graph G may not be weakly connected, so it is possible to have e << n.

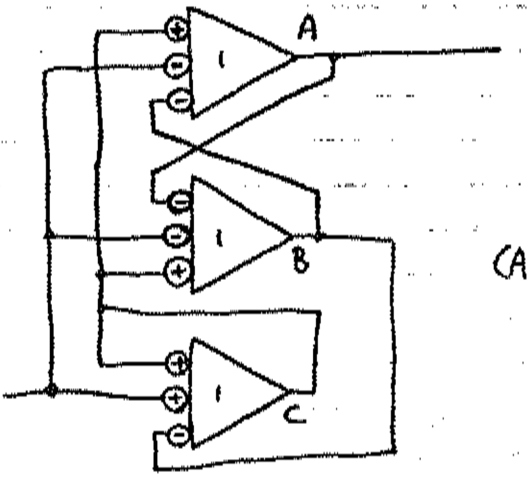
H has size O(ne) because if u -> v is an edge of H, there must be some edge w -> v of G, i.e. some edge must point into v or v would not be reachable. The number of reachable vertices in H is at most e, since at most e vertices of G lie at the end of edges. Finally, each reachable vertex of H can be reached by at most n-1 other vertices, so the total number of edges of H is at most e(n-1) which is O(ne).

NOTE: A slight strengthening of the above argument shows that the bound on the size of H is O(min(n^2,e^2)).

2. The simplest way to derive a lower bound is to show that there are many possible sequences c[1],c[2],...,c[n], and then give an information-theoretic lower bound on the number of tests necessary to select one of them. Suppose our real numbers xi are some permutation of the sequence 1,4,9,16,25,36,.... The nearest neighbor of each xi (except 1) is the predecessor of xi in the sequence.

There will be exactly one pair of indices i,j such that c[i] = j and c[j] = i, corresponding to the positions of the numbers 1 and 4. Since c[k] for the other vertices is the predecessor of xk in the ordering, the remaining c[k] uniquely determine the order of the xi's. Since there are n! possible orderings, there must be at least as many distinct sequences of c[k]'s, so a lower bound on computing the c[k]'s is log(n!) which is Omega(n log n).

Q3 (a)

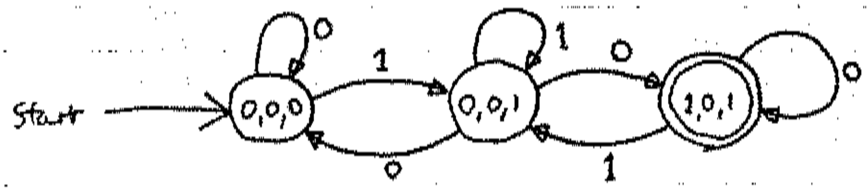


with the neurons labeled A, B, C the possible stable states are

(A,B,C) = (0,0,0) or (0,0,1) or (1,0,1)

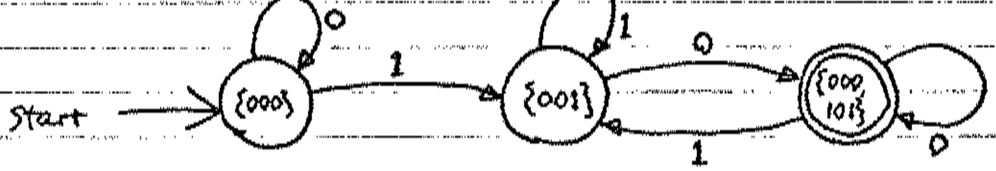
and (1,0,1) is an accepting state

the NDFA is:



3(b) the reachable sets of states for the DFA are {000}, {001} {000, 101}

and the machine is:



and it is clearly minimal

Answers to Fall 1988 Undergraduate Theory Prelim

E. L. Lawler

4.(a) L_1 is regular. There are various ways to prove this. One way is by induction over the form of regular expressions. Another is as follows: Since L_0 is regular, it has an FSA. Construct a nondeterministic FSA for L_1 as follows. Make Make two copies of the transition diagram for the FSA for L_0, priming all the states in the second copy. Then add transitions from the first copy of the diagram to the second copy as follows. For each transition (q,r) on a symbol a, provide a transition (q,r') on each of the symbols in $\Sigma - \{a\}$. Since L_1 is accepted by this nondeterministic FSA, L_1 is regular.

(b) L_1 is context free. Again there are various ways to prove this. One way is by a construction involving PDAs, similar to that in part (a). Another is by modifying a grammar for L_0. We indicate this modification by example. Suppose L_0 generated by the following grammar, with a,b,c as terminals:

$$\begin{array}{l} S \dashrightarrow SA \mid ABc \\ A \dashrightarrow BA \mid ab \\ B \dashrightarrow bc \end{array}$$

Let unprimed nonterminals allow terminal subsitutions and primed unterminals allow no substitutions. Then a grammar for L_1 is as follows:

$$\begin{array}{l|l} S \dashrightarrow S'A & SA' \mid A'Bc \mid AB'c \mid A'B'a \mid A'B'b \\ S' \dashrightarrow S'A' & A'B'c \\ A \dashrightarrow B'A & BA' \mid ab \mid bb \mid cb \mid aa \mid ac \\ A' \dashrightarrow B'A' & ab \\ \multicolumn{2}{l}{B \dashrightarrow bc \mid ac \mid cc \mid ba \mid bb} \\ \multicolumn{2}{l}{B' \dashrightarrow bc} \end{array}$$

(By specialization to right/left linear grammars, this construction also works for part (a).)

5.(a) A covering set S must contain at least one point x such that $x \leq b_{\min} = \min\{b_i\}$. Any such point x covers a subset of the intervals $[a_i, b_i]$ with $a_i \leq b_{\min}$. And $b_{\min}$ itself covers *all* such intervals. Therefore there exists a minimum cardinality covering set that contains $b_{\min}$, and no other points to the left of it. It follows that the following procedure computes an optimal covering set S:

```
$I = \{1,2,...,n\}$;
$S = \emptyset$;

while ($I \neq \emptyset$) {
     $b_{\min} = \min\{b_i \mid i \in I\}$;
     $S = S \cup \{b_{\min}\}$;
     $I = I - \{i \mid a_i \leq b_{\min}\}$;
}
```

With a priority queue that supports the operations *MIN*-a_i, *MIN*-b_i, *DELETE MIN*-b_i, each carried vut in $O(\log n)$ time, the procedure can be implemented to run in $O(n \log n)$ time.

(b) If there is some point on the circle that is not contained in at least one of the n arcs, then the problem is like that in part (a). So suppose this is not so. Without loss of generality, we may assume that a minimum cardinality covering set S contains only right endpoints of arcs, i.e., "clockwise" right endpoints. There are n such endpoints. If one chooses a given right endpoint to be in S and eliminates all the arcs that are covered by it, the remaining problem is like that in part (a). This means that the problem reduces to at most n problems like that in part (a). Hence the problem can be solved in $O(n^2 \log n)$ time.

6. Given a set of k points, it is easy to check that they cover all regions Ai. Hence the problem is in NP.

The problem is NP-complete, by transformation from VERTEX COVER: Let $G = (V,E)$ be the given graph. Assign the n vertices to any n distinct points on the real line. For each edge $\{u,v\}$ create a region A_i that is the union of the two intervals $[u,u]$ and $[v,v]$. Quite clearly G has a vertex cover of size k or less if and only if there is a covering set S of the same size.

THEORY CORE EXAM
SPRING 1989

UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering
and Computer Science
Computer Science Division

Spring 1989

CS THEORY PRELIMINARY EXAMINATION

Do NOT turn this page before you hear the starting gun.

You have three hours to complete all questions.

This is a closed book examination.

There are SIX questions. They all carry equal number of points, but are not necessarily of equal difficulty.

Put all calculations and answers in blue books.

Write your ID number on the front cover of every book.

GOOD LUCK !!

PROBLEM 1

Let a = (a(1),a(2), ...,a(n)) and b = (b(1),b(2),...,b(n)) be arrays, each consisting of n distinct integers in increasing order. Assume that no integer occurs in both a and b. Give the fastest algorithm you can for finding the n'th-smallest element in the union of the two arrays.

[Hint: How would you decide whether a particular element a(i) is among the n smallest elements in the union of the two arrays?]

PROBLEM 2

A set of vertices S in graph G is said to be independent if no two vertices in S are adjacent; S is said to be a maximum independent set in G if it is independent in G, and no independent set in G contains more vertices than S does.

Give the fastest algorithm you can find for the following problem:

INPUT: A tree T, represented via adjacency lists (i.e. for each vertex v, a linked list of vertices adjacent to v is given).
OUTPUT: The number of vertices in the maximum independent set of T.

State the running time of your algorithm as a function of the number of vertices in T.

You may assume without proof the properties of any textbook algorithm that you wish to use as a subroutine.

PROBLEM 3

L1 is a SORTED list of 10 numbers; L2 is an UNSORTED list of 10 numbers. You may assume that the 20 numbers in L1 and L2 are distinct. Determine the worst-case information-theoretic lower bound on the number of comparisons required to find:

(a) The 5th smallest number among the 20 numbers in L1 and L2.

(b) The 5th and 6th smallest numbers among the 20 numbers in L1 and L2.

PROBLEM 4

(a) Prove that the following language over the alphabet {a,b,c} is not context free: the set of all strings containing equal numbers of a's, b's and c's.

(b) Give an informal description of a nondeterministic pushdown automaton that accepts the complement of the following language: {ww | w ∈ {a,b}*}.

PROBLEM 5

(a) M and M′ are Mealy machines. In each machine, x(t), z(t) and s(t) denote the input, output and internal state, respectively, at time t. Shown below are the (incomplete) transition tables for these machines.

M

x(t) / s(t)	s(t+1)		z(t)	
	0	1	0	1
1	1	2	0	1
2	2	1	1	0
3				

M′

x(t) / s(t)	s(t+1)		z(t)	
	0	1	0	1
1′	1′	2′	0	1
2′	2′	1′	1	0
3′				

It is known that M and M′ are EQUIVALENT; however, they are NOT ISOMORPHIC (i.e. one cannot be obtained from the other simply by renaming states).

Fill in the missing entries in the table. (The answer is not unique; any correct one will do.)

(b) The set of strings T_1 is represented by the regular expression

$$R_1 = 0^*1(0+1)^*$$

(where + denotes the OR operator). The set of strings T_2 is represented by the regular expression

$$R_2 = ((0+1)0^*1)^*$$

Write a regular expression R which represents the set of strings $T = T_1 - T_2$. [R should use only the concatenation (•), iteration (*) and OR (+) operations; don't use the - operator.]

PROBLEM 6

Prove that the following problem is NP-complete.

DISJOINT PATHS PROBLEM

INPUT: An undirected graph G and a sequence
(s(1),t(1)), (s(2),t(2)), ..., (s(k),t(k))
of pairs of vertices in G, where all 2k vertices are distinct.
QUESTION: Does G contain k paths such that:
(i) for i = 1,2,...,k, the i'th path joins s(i) with t(i), and
(ii) no two of the k paths have a vertex in common ?

[Hint: Give a polynomial-time transformation from the satisfiability problem to the disjoint paths problem. The transformation should be such that an instance of the satisfiability problem with k clauses transforms to an instance of the disjoint paths problem requiring k paths.]

Prob. 3 Solution

(a)

If 5th elt is in L1: 5 possibilities

" " " " " L2: $\underline{10}$ "

15 — 4 pt.

$ITLB = \lceil \lg 15 \rceil = \underline{\underline{4}}$

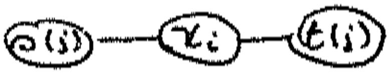
(b)

In L1	In L2	# possibilites
5th, 6th		5
	5th, 6th	9·10 = 90
5th	6th	5·10 = 50
6th	5th	6·10 = $\underline{60}$
		205

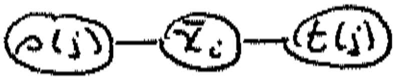
6 pts

$ITLB = \lceil \lg 205 \rceil = \underline{\underline{8}}$

for j = 1,2,...,k and for each variable x_i in C(j):



for j = 1,2,...,k and for each complemented variable $\bar{x}_i$ in C(j):



The path chosen to connect S(i) with T(i) determines a truth-value setting. If the upper path is taken, then x_i is false; if the lower path is taken, then x_i is true. It is possible to connect the remaining source-sink pairs if and only if this truth-value setting satisfies all the clauses. This establishes that SAT is polynomial-time transformable to DPP, and thus that DPP is NP-complete.

Solutions to Questions 1,2,4 and 6 on the Theory Prelim
R.M. Karp

1. Call a(i) <u>small</u> if it is among the n smallest elements, and otherwise <u>large</u>. Clearly a(i) is small if and only if it is less than b(n-i+1). Also, the small elements of array a precede the large ones. Therefore, by binary search, we can determine, in $\lceil \lg(n+1) \rceil$ comparisons, which elements of the a-array are small. If none are small then b(n) is the nth-smallest element. If a(j) is the last small element in the a-array, then the nth-smallest element is max(a(j), b(n-j)). The algorithm requires $1 + \lceil \lg(n+1) \rceil$ comparisons.

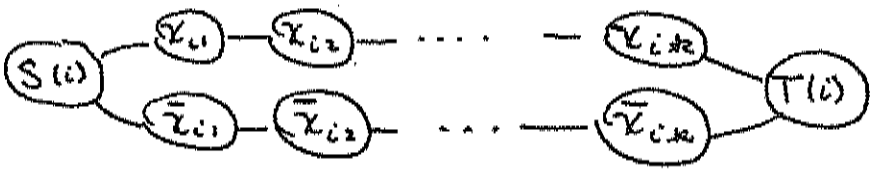
2. Let T be rooted at some vertex r. Clearly, there is a maximum independent set containing all the leaves of this rooted tree. Applying this observation inductively, we can build a maximum independent set S by moving from the leaves to the root, applying the following rule: vertex x is in S if and only if none of its children is in S. The set S can be constructed in time O(n) as a byproduct of a depth-first search of the rooted tree. The membership of each vertex in S is determined on the last visit to that vertex (when it gets popped from the depth-first search stack). Whenever a vertex enters S it marks its parent ineligible, and a vertex x enters S only if it hasn't been marked ineligible by the time it is popped from the stack.

4. Let L be the given language, and assume for contradiction that L is context-free. Let R be the regular language $a^* b^* c^*$. Then $L \cap R$ is context-free, since the intersection of a context-free language with a regular language is context-free. Note that $L \cap R = \{a^n b^n c^n : n \geqslant 0\}$. By the pumping lemma, every sufficiently long string in $L \cap R$ is of the form uvwxy, where v and x are not both empty and, for all i, $uv^i wx^i y$ lies in L R. Neither v nor x contains two distinct letters, since the occurrences of those letters would be interleaved in $uv^2 wx^2 y$. Hence some letter is missing from vx, and all three letters cannot occur with equal frequency in $uv^2 wx^2 y$. This contradiction establishes that L is not context-free.

6. The Disjoint Paths Problem (DPP) lies in NP, since there is a polynomial-time algorithm to check whether a given set of paths lies in G, is vertex-disjoint, and has the correct end-points.

To prove that DPP is NP-complete we give a reduction from Satisfiability (SAT). Let an instance of SAT have clauses C(1), C(2),...,C(k) and variables x_1, x_2,..., x_n. The corresponding DPP instance will have the n+k source-sink pairs (S(1),T(1)),..., (S(n),T(n)) and (s(1),t(1)),...,(s(k),t(k)) and additional vertices x_{ij} and $\bar{x}_{ij}$, for i = 1,2,...,n and j = 1,2,...,k. Its graph will be the union of the following subgraphs:

for i = 1,2,...,n



Prob. 5 Solution

(a) Since M and M' are equivalent but not isomorphic, they cannot be minimal.

From the tables it is evident that

$1 \sim 1'$, $2 \sim 2'$

$1 \not\sim 2$, $1' \not\sim 2'$

Hence, for M and M' to be reducible we must have

$3 \sim 1$, $3' \sim 2'$

or $3 \sim 2$, $3' \sim 1'$

4 pts

Possible answers:

M

s(t)	0	1	0	1
3	1	2	0	1

M'

s(t)	0	1	0	1
3'	2'	1'	1	0

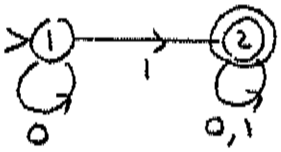
or

3	2	1	0	0

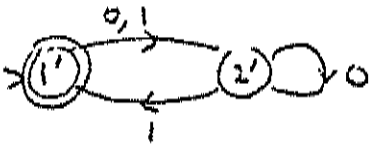
3'	1'	2'	0	1

(There are other variations.)

(b) $R_1 = 0^*1(0+1)^*$:

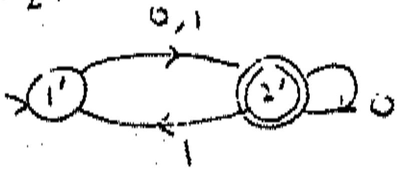


$R_2 = ((0+1)0^*1)^*$:

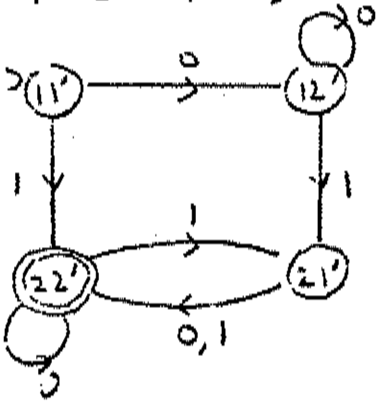


6 pts

$\overline{R_2}$:



$R = R_1 - R_2 = R_1 \cap \overline{R_2}$



By inspection:

$R = (1 + 00^*1(0+1))(0 + 1(0+1))^*$

(There are other forms.)