Algorithms and Theoretical Computer Science
Ph.D. Qualifying Examination
Fall 2008

Name:
Alias:

- Please write your name and an alias of your choice in the boxes above, and submit this page separately from your answers. Please write your alias ***but not your name*** on your answers; this will allow us to grade your exam anonymously.
- The exam consists of eight written questions, four in the morning and four in the afternoon. You will have three hours to answer each group of four questions.
- Please start your answer to each numbered question on a new sheet of paper, so that we can distribute your answers to the faculty who posed each question.
- Write clearly and concisely. You may appeal to standard textbook results (algorithms, data structures, theorems, etc.) unless the problem explicitly asks for details or a proof.
- If you cannot solve a problem completely, you can get partial credit by expressing your main idea/approach clearly and concisely, or by stating necessary lemmas you believe to be true but cannot prove during the exam. All else being equal, it is better to solve some of the problems completely than to get partial credit on every problem.

If you go to Heaven without being naturally qualified for it, you will not enjoy yourself there.

— George Bernard Shaw

	Algorithms				Complexity				
#	1	2	3	4	1	2	3	4	$\sum$
Score									
Max	25	25	25	25	25	25	25	25	200
Grader									

Algorithms and Data Structures

1. A *minor* of a matrix M is a matrix obtained by deleting a subset of the rows and a subset of the columns of M. We are interested in the number of binary matrices (that is, matrices with each entry equal to 0 or 1) that exclude certain minors.

   (a) What is the total number of $n \times n$ binary matrices?

   (b) An *interval matrix* is a binary matrix where the 1's appearing in each row appear together in a contiguous block. Bound the number of $n \times n$ binary matrices that are interval matrices.

   (c) Bound the number of $n \times n$ binary matrices that do not have $\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$ or $\left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right]$ as a minor. Prove your bound. *[Hint: Transform such a matrix into a nicer-looking matrix, and bound the number of nicer matrices (but don't forget to count the transformations).]*

   (d) Prove that if a $n \times n$ binary matrix does not have $\left[\begin{smallmatrix}1&1\\1&1\end{smallmatrix}\right]$ as a minor, then the number of 1s in the matrix is at most $O(n^{3/2})$.

   (e) Bound the number of $n \times n$ binary matrices that do not have $\left[\begin{smallmatrix}1&1\\1&1\end{smallmatrix}\right]$ as a minor.

2. Let $G = (V, E)$ be a directed graph representing a flow network, and let $c\colon E \to \mathbb{Z}$ be a function that assigns an integer *capacity* to each edge. For an arbitrary subset X of the vertices, let $\delta^+(X)$ and $\delta^-(X)$ denote the set of edges leaving X and entering X, respectively. Let s and t be two distinct nodes, called the *source* and *sink* of the flow network. An *(s,t)-cut* is a subset $S \subset V$ such that $s \in S$ and $t \notin S$. The *capacity* of the (s,t)-cut S is the sum of the capacities of the edges leaving S:

$$\text{capacity}(S) := c(\delta^+(S)) = \sum_{e \in \delta^+(S)} c(e).$$

   We are interested in the *minimum-capacity* (s,t)-cuts, which we call *min-cuts*. Call a min-cut S *minimal* if no proper subset of S is also a min-cut.

   (a) Prove that there is a *unique* minimal min-cut. *[Hint: Prove that if S and S' are minimum cuts, then so is $S \cap S'$.]*

   (b) Describe a polynomial-time algorithm to compute the minimal min-cut.

3. Let $D[1\,..\,n]$ be an array of digits, each an integer between 0 and 9. A ***digital subsequence*** of D is an sequence of positive integers composed in the usual way from disjoint substrings of D. For example, $3, 4, 5, 6, 8, 9, 32, 38, 46, 64, 83, 279$ is an increasing digital subsequence of the first several digits of π:

$$\underbrace{3},1,\underbrace{4},1,\underbrace{5},9,2,\underbrace{6},5,3,5,\underbrace{8},\underbrace{9},7,9,\underbrace{3,2},\underbrace{3,8},\underbrace{4,6},2,\underbrace{6,4},3,3,\underbrace{8,3},\underbrace{2,7,9}$$

   The *length* of a digital subsequence is the number of integers it contains, *not* the number of digits; the preceding example has length 12.

   Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of D. *[Hint: Be careful about your computational assumptions. How long does it take to compare two k-digit numbers?]*

4. The well-known knapsack problem is the following. Given a set of n items, where each item i has a size s_i and a profit p_i, find a maximum-profit subset of the items that can fit in a knapsack of given capacity B.

   The *multiple* knapsack problem generalizes this to the setting where we have m knapsacks. If we are given an exact algorithm for the single knapsack problem, there is a natural greedy strategy for the multiple knapsack problem: Pack the first knapsack using the exact algorithm, remove the packed items, and recursively fill the remaining $m - 1$ knapsacks.

   (a) Prove that this greedy strategy does not give an exact solution to the uniform multiple knapsack problem, even if all the knapsacks have the same capacity.

   (b) Prove that this greedy strategy gives a constant-factor approximation for the multiple knapsack problem, when all the knapsacks have the same capacity.

   (c) Now suppose the knapsacks have different capacities. Suppose we order the knapsacks in order of *decreasing* capacity, and then apply the same greedy strategy, first optimally packing the largest knapsack, then the second largest knapsack, and so on. Prove that this strategy gives a constant-factor approximation for the multiple knapsack problem.

   (d) **Bonus:** In part (c), does the ordering of the knapsacks actually matter?

Formal Languages and Complexity Theory

1. The *size* of a boolean formula is the number of literals it contains. Show that a boolean function $f : \{0,1\}^n \to \{0,1\}$ is computable by a boolean formula of polynomial size (as a function of n) if and only if f is computable by an unbounded fan-in boolean circuit with logarithmic depth (that is, if and only if f is in non-uniform NC^1).

   [Hint: Let f be an arbitrary boolean formula, and let g be a subformula of f. Let $f \backslash g$ denote the formula obtained from f by replacing the subformula g with a single new variable. Describe how to rewrite f so that its depth is at most $\max\{\mathit{depth}(f \backslash g), \mathit{depth}(g)\} + 2$ *and size is no more than* $2 \cdot \mathit{size}(f \backslash g) + 2 \cdot \mathit{size}(g)$*.]*

2. Recall the following standard definitions.

   - A function f is ***computable*** if there is a Turing machine M with a read-only input tape, write-only output tape, and a finite number of work tapes, such that M, given input x, halts with $f(x)$ on its output tape.
   - A function $f : \Sigma^* \to \Sigma^*$ is a ***many-one reduction*** from $L_1 \subseteq \Sigma^*$ to $L_2 \subseteq \Sigma^*$ if f is computable, and for every $x \in \Sigma^*$, we have $x \in L_1$ if and only if $f(x) \in L_2$. We say L_1 is ***many-one reducible*** to L_2, and we write $\boldsymbol{L_1 \leq_m L_2}$, if there is a many-one reduction from L_1 to L_2.
   - Let $\mathcal{C}$ be a set of languages. A language L is $\mathcal{C}$***-hard*** if $L_1 \leq_m L$ for every language $L_1 \in \mathcal{C}$.
   - $\mathrm{RE} = \{L(M) \mid M \text{ is a Turing machine}\}$ is the set of all recursively enumerable languages.
   - $\text{co-RE} = \{L \mid \bar{L} \in \mathrm{RE}\}$ is the set of all complements of recursively enumerable languages.

   The ***universality problem*** for Turing machines is defined by the language $\textsc{Univ} = \{x \mid L(M_x) = \Sigma^*\}$, where M_x is the Turing Machine whose code is x.

   (a) Prove that $\textsc{Univ}$ is both RE-hard and co-RE-hard.

   (b) Prove that $\textsc{Univ} \notin \mathrm{RE} \cup \text{co-RE}$.

3. (a) Prove that if a unary language $L \subseteq \{1\}^*$ is NP-complete then $\mathrm{P} = \mathrm{NP}$.

   [Hint: Given a SAT instance φ, consider the (exponentially large) binary tree with 2^i nodes in the ith level, corresponding to the 2^i possible assignments to the first i variables of φ. Use the polynomial-time reduction from SAT to L to prune each level to a shorter list, such that if φ is satisfiable at least one partial assignment that can be extended to a satisfying assignment is retained in the list.]

   (b) A language $L \subseteq \{0,1\}^*$ is said to be ***sparse*** if the number of n-bit strings in L is polynomially bounded, that is, if there is some polynomial p such that $|L \cap \{0,1\}^n| \leq p(n)$ for all n. Show that if a sparse language L is NP-complete then $\mathrm{P} = \mathrm{NP}$. (This result is known as *Mahaney's Theorem.*)

   [Hint: Let Σ be the set of pairs (φ, x) where φ is a satisfiable boolean formula and x is a partial assignment such that some partial assignment $y \leq x$ can be extended to a satisfying assignment of φ. Use a reduction from Σ to L.]

4. Show that redirecting an edge in a minimal DFA yields a different language.

   More precisely: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton with states Q, alphabet Σ, transition function $\delta \colon Q \times \Sigma \to Q$, start state $q_0 \in Q$, and accepting states $F \subset Q$. Assume that M is minimized. Let $M' = (Q, \Sigma, \delta', q_0, F)$ be obtained from M by redirecting one transition; that is, $\delta(q, a) \neq \delta'(q, a)$ for exactly one state $q \in Q$ and one symbol $a \in A$. Prove that $L(M) \neq L(M')$.

   [Hint: Beware of nontrivial automorphisms!]