ALGORITHMS EXAMINATION
Department of Computer Science
New York University
December 18, 2006

This examination is a *three hour* exam. All questions carry the same weight.
Answer all of the following *six* questions. Please be aware that to pass this exam you need to provide good answers to several questions; it is not sufficient to obtain partial credit on each question.

• Please print your name and SID on the front of the envelope only (not on the exam booklets).
• Please answer each question in a *separate* booklet, and *number* each booklet according to the question.

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

Good luck!

Question 1

Let G be a directed graph. We say that G is *k-cyclic* if every (not necessarily simple) cycle in G contains at most k distinct nodes.

a. Give a linear-time algorithm to determine if a directed graph G is k-cyclic, where G and k are given as inputs. Justify the correctness and running time of your algorithm.

b. Give a linear-time algorithm for the following problem. The input is a directed graph $G = (V, E)$, with non-negative edge weights, along with nodes $s, t \in V$. Furthermore, assume that G is 10-cyclic. The algorithm is to determine the weight of the shortest path from s to t. Justify the correctness and running time of your algorithm.

Question 2

Give a data structure, along with corresponding procedures, implementing a priority queue supporting the following operations:

(a) *Insert x into Q:* insert a value x into the queue Q.

(b) *DeleteMin Q:* remove and return the smallest value in the queue Q.

(c) *AddValue δ to Q:* add the value δ to all values currently in Q.

(d) *Merge Q_1, Q_2:* merge two queues into one (destroying the originals).

All operations are to take time $O(\log n)$, where n is the number of items in Q in (a–c), and the total number of items in Q_1 and Q_2 in (d). Justify the correctness and running times of your procedures.

Go To the Next Page

Question 3

a. The fat chance satisfiability (FATCHAT) problem is the following.
Instance: A 3-CNF formula with m clauses where each clause contains three distinct literals (so that a clause such as $(\overline{x}_{14} \vee x_{33} \vee \overline{x}_{14})$ cannot occur).

Question: Is there an assignment to the Boolean variables such that at least $7m/8$ clauses are satisfied?

Show that the answer to FATCHAT is always yes.
Hint. Consider a random assignment.
For expositional simplicity, you can assume that no Boolean variable and its negation appear as literals in any one clause.

b. The very satisfiable problem (VERSAT) is the following.
Instance: A 3-SAT problem with m clauses.
Question: Is there an assignment to the Boolean variables so that at least $8m/9$ of the clauses are satisfied?

Prove that VERSAT is NP-Complete.

Go To the Next Page

Question 4

Let $A[1..n]$ and $B[1..n]$ be arrays of n characters each. A variant of the greatest common subsequence problem is as follows. Let $M(\alpha, \beta)$ return the numerical "goodness" of the match for the individual characters α and β. The best matching subsequence problem is to find, for the best k, the k-character A-subsequence $A[i_1], A[i_2,], \ldots, A[i_k]$ where $1 \le i_1 < i_2 < \cdots < i_k \le n$, and the corresponding B-subsequence $B[j_1], B[j_2,], \ldots, B[j_k]$ where $1 \le j_1 < j_2 < \cdots < j_k \le n$, where the "goodness" $M(A_{i_1}, B_{j_1}) + M(A_{i_2}, B_{j_2}) + \cdots + M(A_{i_k}, B_{j_k})$ is as large as possible. So when $Match(\alpha, \beta) = 1$ if $\alpha = \beta$ and 0 otherwise, then this problem is just the standard greatest common subsequence problem for the strings A and B.

a. Present a high-level recursive specification to find the numerical "goodness" of the best match for this variant of the greatest common subsequence problem. The algorithm should be efficient when properly implemented with table look-up to avoid the recomputation of already solved subproblems, but this look-up does NOT have to be included in your specification.

b. Now suppose that the value to be optimized is not $M(A_{i_1}, B_{j_1}) + M(A_{i_2}, B_{j_2}) + \cdots + M(A_{i_k}, B_{j_k})$ for the best k and i and j subsequences, but is instead $M(A_{i_1}, B_{j_1}) + M(A_{i_2}, B_{j_2}) + \cdots + M(A_{i_k}, B_{j_k}) - k^2$ for the best k.

Present a high-level recursive specification to solve this problem. The target efficiently should be $O(n^3)$ when the solution is properly implemented with table lookup.
Hint: introduce a new dimension to the recursive solution that is indexed by k.
Please note: for your convenience, the string sequences A and B are required to have the same length n.

Go To the Next Page

Question 5

Let p be a suitably large prime. Let Σ be the alphabet $\{0, 1, \cdots, p-1\}$. Finally, suppose that arithmetic mod p takes constant time.

Give a dynamic data structure D that maintains a "compact representation" of two dynamically growing strings A and B over the alphabet Σ, where both strings are initially empty. D needs to support the following update operations:

- *Append*(S, a). On input $a \in \Sigma$ and $S = A$ or B, appends a to string S (i.e., sets $S := Sa$). It also returns a bit indicating whether or not $A = B$ now.
- *Prepend*(S, a). On input $a \in \Sigma$ and $S = A$ or B, prepends a to string S (i.e., sets $S := aS$). It also returns a bit indicating whether or not $A = B$ now.

Your implementation of D should also satisfy the following constraints:

(a) D occupies constant space on the machine, irrespective of the number of update queries.

(b) D is *randomly initialized* at the beginning (i.e., when A and B are empty) in constant time.

(c) After initialization, all the updates are *deterministic* and take constant time each.

(d) The n-th update operation is answered incorrectly with probability at most n/p.

Make sure you explicitly prove that your D satisfies properties (a)-(d) above.

Go To the Next Page

Question 6

In this question you will show that NFAs can be exponentially more efficient than DFAs.

(a) Given a language L, define its *Myhill-Nerode* relation by: strings x and y are equivalent (denoted $x \equiv y$) if for *any* string z,

$$xz \in L \iff yz \in L$$

It can be shown that $\equiv$ is indeed an equivalence relation. Let the index of L, $index(L)$, be the number of equivalence classes (i.e., pairwise distinguishable strings) in $\equiv$, which, in general, could be finite or infinite.

Show that if L is accepted by a DFA with N states, then $index(L) \leq N$.
(*Hint:* what is a "compact description" of the equivalence class of x in terms of the DFA?)

(b) Let C_k ($k \geq 1$) be the language over $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ consisting of all the strings whose k^{th} symbol from the end is a: $C_k = \Sigma^* \mathsf{a} \Sigma^{k-1}$. Assuming part (a) as given (whether or not you solved it), prove that no DFA can recognize C_k with fewer than 2^k states.[1]

(c) Describe an NFA with $(k+1)$ states that recognizes C_k, both in terms of the state diagram and a formal description.

[1] You also get full credit for a direct proof, without using part (a).