In preparing your solutions to the exam, you are **allowed to use any sources** including your textbook, other students and professors, previous homeworks and solutions, or any sources on the Internet. The only restriction is that I will not help you with exam questions (although you are welcome to ask me general questions that do not specifically relate to a problem on the exam). I am also happy to answer clarifying questions about exam problems.

The exam will take place in MC Reynolds 317 from 2–5pm on Monday, May 9. You are **not** allowed to bring any notes, textbooks, calculators, or any other resources with you to the exam. Bring only something to write with; I will provide a fresh copy of the exam, paper for writing your solutions, and scratch paper.

As usual, *efficient* means polynomial in the size of the input—where smaller polynomials are better! When asked to describe and analyze an algorithm, you should give a careful description of the algorithm (using pseudocode, if appropriate), analyze its asymptotic running time, and prove its correctness.

Question 1 (20 points)**.** Describe and analyze a polynomial-time algorithm to test whether a *directed* graph has any directed 3-cycles, that is, cyclic directed paths of the form $u \to v \to w \to u$. For this question, **any** correct polynomial time algorithm will receive full credit; you do not have to give the fastest possible algorithm.

Question 2 (20 points)**.** Describe and analyze a polynomial-time algorithm for detecting small cycles in a graph. Specifically, given an undirected graph G and a positive integer $k \geq 3$, your algorithm should determine whether the graph contains any cycles of length $\leq k$. Your algorithm should run asymptotically faster than $O(n^4)$, where n is the number of vertices. In particular, the runtime of your algorithm should not depend on k.

Question 3 (20 points)**.** Suppose that instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of *fits* (*fi*bonacci dig*its*), where each fit is either 0 or 1, and the i^{th} least significant fit indicates whether the sum includes the i^{th} Fibonacci number F_i. For example, the fit-string 101110_F represents the number

$$F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14.$$

Describe and analyze an algorithm to increment a fit-string in $O(1)$ amortized time. As with the example above, assume that $F_1 = F_2 = 1$, that the array of bits is 1-indexed, and that the i^{th} index of the array stores the i^{th} fit. Also observe that fit strings do not necessarily have unique representations, but it does not matter what representation your algorithm uses. If it helps, you may assume that your algorithm is always called iteratively starting from zero, so it does not have to handle arbitrary representations as inputs, only zero and those fit-strings which are produced as the output of your algorithm.

Question 4 (20 points)**.** A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`.

Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome. For example, the longest palindrome subsequence of

`MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM`

is `MHYMRORMYHM`, so given that string as input, your algorithm should output the number 11.

Question 5 (20 points)**.** A *dominating set* for an undirected graph $G = (V, E)$ is a set of vertices $D \subseteq V$ such that for every vertex $v \in V$, either

1. $v \in D$, or
2. v is adjacent to at least one member of D.

For example, the marked black vertices form a dominating set in the graph at the top of Figure 1; every other vertex is adjacent to at least one of the marked vertices. On the other hand, the marked vertices in the bottom graph do not form a dominating set, since the lower-right vertex is not adjacent to any marked vertex.

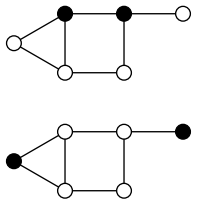

Figure 1: Examples of dominating and non-dominating sets

The DOMINATING SET problem asks, given a graph G and a positive integer k, does G have a dominating set of size $\leq k$? Prove that DOMINATING SET is NP-complete.

Acknowledgment Questions 3 and 4 are adapted from Jeff Erickson (`http://jeffe.cs.illinois.edu/teaching/algorithms/`).