COS 423 **Theory of Algorithms** **Spring 2005**

Writing Solutions to Problem Sets

Learning how to write clear and rigorous solutions is an important component of this course. Vague and sloppy solutions often turn out to have inaccuracies that render them incorrect. You will lose a significant number of points if your solution is imprecise or lacks sufficient explanation, even if the solution turns out to be correct. Here are a few guidelines.

- Your solutions should be neatly written and well-organized. It need not be typed, unless you have really awful handwriting.
- You'll notice that the problem sets contain some "word problems" that consist primarily of an English description of a problem, with little or no mathematical notation. This is intentional. For such problems, you should first extract the essence of the underlying problem and formalize it mathematically, then solve the problem.
- When a question asks you to "design an algorithm" or "solve" a problem, you are expected to provide an efficient algorithm, along with a proof of correctness, and an analysis of its running time.
- Typically, the clearest way to describe your solution is to first explain the key ideas in English, possibly with the use of some notation (that you clearly define), and possibly some high-level pseudocode. A solution consisting solely of pseudocode and no accompanying explanation will receive little if any credit.
- We will award partial credit for partial solutions. However, to be considered for partial credit, you must clearly indicate where your proof or algorithm falls apart, and what would be needed to fix it. It is better to acknowledge that you are stuck than to pretend that a bogus solution is correct.
- Once you have discovered a solution to the problem, try to simplify it, and make your solution as elegant and clean as possible. This will not only help the grader understand your solution, but it will give you an opportunity to clarify your thoughts, and gain insight into the problem. At this point, you may even be in a position to improve your algorithm and analysis. Along these lines, we will award bonus points for especially elegant or efficient solutions.

Sample problem. [1] Rita, a columnist for the Daily Princetonian, is covering a party. Rita's job is to identify a celebrity, if one exists. A *celebrity* is person that is known by every other person, but doesn't know any of them. Rita asks questions to the guests of the following form: "Excuse me pal. Do you know the person over there?" Assume that all of the guests at the party are polite (even the celebrity) and answer any question with the correct answer. Explain how Rita can identify the celebrity using as few questions as possible. Before looking at the solution, think about the problem on your own.

[1] Reference: *Introduction to Algorithms: A Creative Approach* by Udi Manber.

Mathematical formulation. Let $G = (V, E)$ be a directed graph. There is a vertex for each of the n guests, and an edge from u to v if guest u knows guest v. We define a *sink* of a directed graph to be a vertex with indegree $n-1$ and outdegree 0. A celebrity corresponds to a sink of the graph. We note that a graph can have at most one sink.

Brute force solution (not much partial credit). The graph has at most $n(n-1)$ edges, and we can compute it by asking a question for each potential edge. At this point, we can check whether a vertex is a sink by computing its indegree and its outdegree. This brute-force solution asks $n(n-1)$ questions and does $\Theta(n^2)$ bookkeeping work building the graph. It yields little, if any, new insight into the problem. Below, we show how to do it with at most $3(n-1)$ questions and $\Theta(n)$ bookkeeping work.

An elegant solution. Our algorithm consists of two phases: in the *elimination phase*, we eliminate all but one guest from being the celebrity; in the *verification phase* we check whether this one remaining guest is indeed a celebrity.

The elimination phase maintains a list of possible celebrities. Initially it contains all n guests. In each iteration, we delete one guest from the list. We exploit the following key observation: *if Alice knows Bob, then Alice is not a celebrity; if Alice doesn't know Bob, then Bob is not a celebrity.* Thus, by asking Alice if she knows Bob, we can eliminate either Alice or Bob from the list of possible celebrities. We use this idea repeatedly to eliminate all guests but one, say Zeus.

We now verify by brute force whether Zeus is a celebrity: for every other guest u, we ask Zeus whether he knows u, and we ask u whether they know Zeus. If Zeus always answers no, and the other guests always answer yes, then we declare Zeus as the celebrity. Otherwise, we conclude there is no celebrity at the party.

Correctness. During the elimination phase, we maintain the invariant that if there exists a celebrity, then the celebrity is on the list. We can prove this by induction on the number of iterations. Thus, when the elimination phase ends, either Zeus is a celebrity or there is no celebrity.

Analysis. The elimination phase requires exactly $n-1$ questions, since each question reduces the size of the list by 1. In the verification phase, we ask Zeus $n-1$ questions, and we ask the other $n-1$ guests one question. This phase requires at most $2(n-1)$ questions, possibly fewer if Zeus is not a celebrity.

To efficiently implement the elimination phase, we maintain a queue that contains the remaining celebrities. Initially, we insert all n guests on the queue. At each iteration we remove the top two elements off the queue, say v and w, and ask v whether she knows w. Depending on the outcome, we either insert v or w at the end of the queue. Each queue operation takes $\Theta(1)$ time, so the whole process takes $\Theta(n)$ time.

An even better solution (bonus points). We note that it is possible to save an additional $\lfloor \log_2 n \rfloor$ questions in the verification phase by not repeating any questions we already asked during the elimination phase. By maintaining the elements in a queue, the celebrity is involved in (i.e., either asked or asked about) at least $\lfloor \log_2 n \rfloor$ questions during the elimination phase. This explains why we chose a queue instead of a stack.

Also, it is not hard to see that any algorithm must ask at least $2(n-1)$ questions if there exists a celebrity, since we must verify that the celebrity doesn't know anyone, and that everyone knows the celebrity.