This document contains a sample problem and solution. Use it as a template for how to write and format a solution in LaTeX.

1 Celebrity Identification Problem

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. 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.

Grader's note: there is no need to include a copy of the problem statement when you submit your problem set. We include it here so that this document is self contained.

2 Mathematical Formulation

We model the relationships among the guests using a digraph $G = (V, E)$ as follows: 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 digraph to be a vertex with indegree $n-1$ and outdegree 0. A *celebrity* corresponds to a sink of the digraph. We note that a digraph can have at most one sink. We also note that the edges E are given to us implicitly—we have an oracle HasEdge(u, v) that tells us whether there is an edge from u to v and we seek to identify a sink (if one exists) using as few calls to HasEdge(u, v) as possible.

Grader's note: If the problem has already been stated explicitly in mathematical terms, skip this section.

3 Brute-Force Solution

We compute the digraph G explicitly by calling HasEdge(u, v) for each potential edge. At this point, we can check whether a vertex v is a sink by computing its indegree and its outdegree. The digraph has at most $n(n-1)$ edges, so the algorithms makes $\Theta(n^2)$ calls to HasEdge() to build the digraph. It yields little, if any, new insight into the problem. Below, we show how to do it with at most $3(n-1)$ calls to HasEdge().

Grader's note: Very little partial credit for a correct, but grossly, inefficient solution. Moreover, while the solution is concise, it is missing both a formal statement and proof of correctness and a formal statement and analysis of its efficiency.

4 An Efficient Solution

Our algorithm for finding a sink consists of two phases: in the *elimination phase*, we eliminate all but one vertex s as a possible sink; in the *verification phase* we check whether s is a sink.

The elimination phase maintains a list of possible sinks. Initially it contains all n vertices. In each iteration, we remove one vertex. We exploit the following key observation: *if there is an edge from u to v, then u is not the sink; if there is not an edge from u to v, then v is not the sink.* Thus, by calling HasEdge(u, v) we can eliminate either u or v from the list of possible sinks. We use this idea repeatedly to eliminate all vertices but one, say s.

Algorithm 1 Eliminating non-sinks.

```
procedure ELIMINATE(V, E)
    L ← MAKELIST
    for v ∈ V do
        ADD(L, v)
    while L contains at least two elements do
        u ← REMOVE(L)
        v ← REMOVE(L)
        if HASEDGE(u, v) then
            INSERT(L, v)
        else
            INSERT(L, u)
    s ← REMOVE(L)
    return s
```

We now verify by brute force whether s is a sink: for every other vertex v, we call both HasEdge(s, v) and HasEdge(v, s). If the former always returns FALSE and the latter always returns TRUE, then we declare s to be the sink. Otherwise, we conclude there is no sink.

Algorithm 2 Verifying a potential sink.

```
procedure VERIFY(V, s)
    for v ∈ V \ {s} do
        if HASEDGE(s, v) or ¬ HASEDGE(v, s) then
            return false
    return true
```

5 Correctness

Proposition 1. *The eliminate-and-verify algorithm either outputs the sink (if one exists) or reports that there is none.*

Proof. During the elimination phase, we maintain the invariant that if there exists a sink, then the sink is in the list. We omit the trivial proof by induction on the number of iterations. Thus, when the elimination phase ends, either s is a sink or there is no sink. This is determined by brute force during verification. □

6 Analysis

Proposition 2. *The eliminate-and-verify algorithm makes between n and $3(n-1)$ calls to* HasEdge().

Proof. The elimination phase requires exactly $n-1$ calls to HasEdge() since each call decreases the size of the list by one. In the verification phase, we call HasEdge() between 1 and $2(n-1)$ times, depending on whether s is a sink. □

7 A Slight Improvement

We note that it is possible to save an additional $\lfloor \log_2 n \rfloor$ calls in the verification phase by not repeating any call we already asked during the elimination phase. By maintaining the list of vertices in a *FIFO queue*, the sink is involved in at least $\lfloor \log_2 n \rfloor$ questions during the elimination phase.

Also, it is not hard to see that *any* algorithm must make at least $2(n-1)$ calls if there exists a sink because we must verify that the sink points to no vertices and that every other vertex points to the sink.

Grader's note: Some bonus points for a better-than-expected solution.

Grader's note: Typically there won't be a simple observation like this that can be tacked onto the end, and a bonus-worthy solution will be a modification to the core of the algorithm or analysis.

8 Appendix

A typical solution to a problem will contain a mathematical formulation and any notation you are introducing (if needed), a description of your algorithm, a proof of correctness, and an analysis of efficiency. Feel free to use sections and subsections to help clarify your answer, but don't go *too* crazy with them or we will not be able to follow your argument.

It is worth noting that this sample solution contains two independent solutions, one of which has a full analysis as well as an improvement. In most cases, you should submit only one solution and analysis. If you submit two solutions, we will use the worst one when assigning your score.