

Due February 20, 2:45pm

**Instructions:** Write your name, the username for your instructional account, your TA's name, and your discussion section time (where you want the homework to be returned) prominently on the first page of your homework. Also list your study partners for this homework, or "none" if you had no partners. *If you do not write your username, you risk receiving no credit for the homework.* Your homework will be returned at the discussion section of the TA you named; if you do not write your TA's name and section time, the graded homework *will not be returned to you.*

When asked for an algorithm, please provide: (a) the pseudocode of your algorithm, (b) a brief explanation of the main idea/intuition, (c) a proof of correctness, (d) the asymptotic running time of the algorithm, using  $O(\cdot)$  notation, (e) justification for your running time calculation.

**1. (25 pts.) Work hard, graduate early**

Suppose a CS curriculum consists of  $n$  courses, all of them mandatory. The prerequisite graph is a directed acyclic graph  $G$  with a vertex for each course, and an edge from course  $v$  to course  $w$  if and only if  $v$  is a prerequisite for  $w$ . Find a linear-time algorithm that computes the minimum number of semesters necessary to complete the curriculum, assuming that you can take and pass any number of courses in one semester. (In other words, we're assuming you are an uberstudent who can take an unlimited number of courses in the same semester, and pass them all!)

**2. (30 pts.) Peace for Grudgeville**

In Grudgeville, people just can't seem to forgive. Every so often two people get into an argument, and then they hate each other forever after. Hating is a symmetric relation: If Alice hates Bob, then Bob hates Alice, too. People who hate each other can't stand to be in each other's presence. This has put a crimp on the social scene in Grudgeville, and Sheriff Brown is tired of stopping the fights that periodically break out.

Then Sheriff Brown has a flash of insight. There's an island offshore. If he could figure out a way to divide the  $n$  townspeople into two groups, so no one hates anyone else in their group, then he could ship one group out to the island, leave the other behind on the mainland, and everyone would be happy. Can you help keep the peace?

Find an efficient algorithm to check whether such a division is possible. You're given a list of  $m$  pairs of people who hate each other. Your algorithm should check whether such a division into two groups is possible, and if so, it should output a roster stating for each person where they will live (on the mainland or on the island).

### 3. (10 pts.) Feeling cross?

Let  $G = (V, E)$  be an undirected graph. Suppose we perform a depth-first search starting from some vertex. Is it possible for this to produce any cross-edges? In other words, is it possible for there to be an edge  $\{u, v\} \in E$  such that neither  $u$  is an ancestor of  $v$  in the resulting depth-first search tree nor  $v$  is an ancestor of  $u$ ? Explain briefly why or why not.

### 4. (35 pts.) Disrupt the terrorists

Let  $G = (V, E)$  denote the “social network” of a group of terrorists. In other words,  $G$  is an undirected graph where each vertex  $v \in V$  corresponds to a terrorist, and we introduce the edge  $\{u, v\}$  if terrorists  $u$  and  $v$  have had contact with each other. The police would like to determine who they should target to capture, to disrupt the coordination of the terrorist group as much as possible. More precisely, the goal is to find a vertex  $v \in V$  whose removal from the graph splits the graph into as many different connected components as possible. This problem will walk you through the design of a linear-time algorithm to solve this problem. In other words, the running time will be  $O(|V| + |E|)$ .

In the following, let  $f(v)$  denote the number of connected components in the graph obtained after deleting vertex  $v$  from  $G$ . Also, assume that initial graph  $G$  is connected (before any vertex is deleted) and is represented in adjacency list format. If you get stuck on one part below, continue on to the subsequent parts of the question, since many parts do not strictly rely upon previous parts.

- (a) Perform a depth-first search starting from some vertex  $r \in V$ . How could you calculate  $f(r)$  from the resulting depth-first search tree in an efficient way?
- (b) Suppose  $v \in V$  is a node in the resulting DFS tree, but  $v$  is not the root of the DFS tree (i.e.,  $v \neq r$ ). Suppose further that no descendant of  $v$  has any non-tree edge to any ancestor of  $v$ . How could you calculate  $f(v)$  from the DFS tree in an efficient way?
- (c) Definition: For each node  $v$  in the DFS tree, let  $d(v)$  denote the depth of  $v$  in the DFS tree. In particular, the root  $r$  has depth 0;  $r$ 's children have depth 1;  $r$ 's grandchildren have depth 2; and so on.

Describe how to compute  $d(v)$  for each vertex  $v \in V$ , in linear time. (For instance, if the vertices are numbered  $0..n-1$ , your goal would be to build and initialize an array  $d[0..n-1]$  so that  $d[v]$  holds the depth of  $v$ .)

- (d) Definition: If  $w$  is a node in the DFS tree, let  $up(w)$  denote the depth of the shallowest node  $y$  such that  $\{x, y\} \in E$  is a graph edge and either  $x$  is a descendant of  $w$  or  $x = w$ . We'll define  $up(w) = \infty$  if there is no edge  $\{x, y\}$  that satisfies these conditions.

Now suppose  $v$  is an arbitrary non-root node in the DFS tree, with children  $w_1, \dots, w_k$ . Describe how to compute  $f(v)$  as a function of  $k$ ,  $up(w_1), \dots, up(w_k)$ , and  $d(v)$ .

Hint: Think about what happened in part (b); think about what changes when we can have non-tree edges that go up from one of  $v$ 's descendants to one of  $v$ 's ancestors; and think about how you can detect it from the information provided.

- (e) Describe how to compute  $up(v)$  for each vertex  $v \in V$ , in linear time.
- (f) Describe how to compute  $f(v)$  for each vertex  $v \in V$ , in linear time.