

CS170 Fall 2013 Solutions to Homework 7

Zackery Field, section Di, 103, `cs170-fe`

October 25, 2013

1. (20 pts.) A greedy algorithm - so to speak

We can formalize this as a graph problem. Let the undirected graph $G = (V, E)$ denote LinkedIn's relationship graph, where each vertex represents a person x who has an account on LinkedIn. There is an edge $\{u, v\} \in E$ if u and v have listed a professional relationship with each other on LinkedIn (we will assume that relationships are symmetric). We are looking for a subset $S \subseteq V$ of vertices so that every vertex $s \in S$ has edges to at least 20 other vertices in S . And we want to make the set S as large as possible, subject to these constraints. Design an efficient algorithm to find the set of super-schmoozers (the largest set S that is consistent with these constraints), given the graph G .

We can begin by identifying any vertices that have degree < 20 . Let the set of these vertices be called \bar{S} . Since the definition of an edge $s \in S$ depends on its neighbors recursively, there has to be a way tracking the number degree of the edges that fall into S and then that information must be returned to all other nodes in the path.

This can be accomplished with a DFS with ties being broken on back edges. Simply mark each edge with pre- and post-visit numberings. As you run the search you will run into three cases:

- You will arrive at a vertex v that is a sink in the sense that it is connected to < 20 vertices $\in \bar{S}$. By definition, $v \in \bar{S}$.
- You will arrive at a vertex v that is connected to ≥ 20 vertices $\in S$. In this case you can not be sure whether or not $v \in S$ because it is not yet known whether the vertices connected to it are $\in S$.
- Arrival at a vertex v reveals a back edge whose vertex l . If the relationship $post(v) - pre(l) \geq 20$ holds, then the vertex v is in a cycle of length ≥ 20 containing vertices

Continued on Page 6

2. (20 pts.) A funky kind of coloring

Let $G = (V, E)$ be an undirected graph where every vertex has degree ≤ 51 . Let's find a way of coloring each vertex blue or gold, so that no vertex has more than 25 neighbors of its own color.

Consider the following algorithm, where we call a vertex 'bad' if it has more than 25 neighbors of its own color:

1. Color each vertex arbitrarily
2. Let $B := \{v \in V : v \text{ is bad}\}$
3. While $B \neq \emptyset$:
 4. Pick any bad vertex $v \in B$
 5. Reverse the color of v .
 6. Update B to reflect this change, so that it again holds the set of bad vertices.

Notice that if the algorithm terminates then, it is guaranteed to find a coloring with the desired property.

- (a) Prove that this algorithm terminates in a finite number of steps. I suggest that you define a *potential function* that associates a non-negative integer (the potential) to each possible way of coloring the graph, in such a way that each iteration of the while-loop is guaranteed to strictly reduce the potential.

Following the hint, allow $p(G) : G \mapsto \mathbb{N}_0$ to denote the potential of a graph G . Let this potential be explicitly defined as the $|B|$, which by definition always takes on a value $b \in \mathbb{N}_0$.

- (b) Prove that the algorithm terminates after at most $|E|$ iterations of the loop.
Hint: You should figure out the largest value the potential could take on.

Optional: Think about how to implement the algorithm so that its total running time is $O(|V| + |E|)$

Continued on Page 7

3. (20 pts.) Job Scheduling

You are given a set of n jobs, each runs in unit time. Job i has an integer-valued deadline time $d_i \geq 0$ and a real-valued penalty $p_i \geq 0$. Jobs may be scheduled to start at any non-negative integer time ($0, 1, 2, \text{etc}$), and only one job may run at a time. If job i completes at or before time d_i , then it incurs no penalty; otherwise, it is late and incurs penalty p_i . The goal is to schedule all jobs so as to minimize the total penalty incurred. For each of the following greedy algorithms, either prove that it is correct, or give a simple counterexample (with at most three jobs) to show that it fails.

- (a) Among the unscheduled jobs that can be scheduled on time, consider the one whose deadline is the earliest (breaking ties with the highest penalty), and schedule it at the earliest available time. Repeat.
- (b) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties with the earliest deadline), and schedule it at the earliest available time. Repeat.
- (c) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties arbitrarily), and schedule it at the latest available time before its deadline. Repeat.

4. (20 pts.) Timesheets

Suppose we have N jobs labelled $1, \dots, N$. For each job, you have determined the bonus of completing the job, $V_i \geq 0$, a penalty per day that you accumulate for not doing the job, $P_i \geq 0$, and the days required for you to successfully complete the job $R_i > 0$.

Every day, we choose one unfinished job to work on. A job i has been finished if we have spent R_i days working on it. This doesn't necessarily mean you have to spend R_i contiguous sequence of days working on job i . We start on day 1, and we want to complete all our jobs and finish with maximum reward. If we finish job i at the end of day t , we will get reward $V_i t * P_i$. Note, this value can be negative if you choose to delay a job for too long.

Given this information, what is the optimal job scheduling policy to complete all of the jobs?

Continued on Page 9

5. (20 pts.) Weighted Set Cover

In class we looked at a greedy algorithm to solve the *set cover* problem, and proved that if the optimal set cover has size k , then our greedy algorithm will find a set cover of size at most $k \log_e n$.

Here is a generalization of the set cover problem.

- *Input:* A set of elements B of size n ; sets $S_1, \dots, S_m \subseteq B$; positive weights w_1, \dots, w_m .
- *Output:* A selection of the sets S_i whose union is B .
- *Cost:* The sum of the weights w_i for the sets that were picked.

Design an algorithm to find the set cover with approximately the smallest cost. Prove that if there is a solution with cost k , then your algorithm will find a solution with cost $O(k \log n)$.

Continued on Page 10

Extra space for Problem 1

Continued from Page 1 adsfasdf

Extra space for Problem 2

Continued from Page 2

Extra space for Problem 3

Continued from Page 3

Extra space for Problem 4

Continued from Page 4

Extra space for Problem 5

Continued from Page 5