

Due March 20, 2:45pm

**Instructions:** See “Instructions for writing up homework” on the course web page.

**1. (25 pts.) A greedy algorithm—so to speak**

The founder of LinkedIn, the professional networking site, decides to crawl LinkedIn’s relationship graph to find all of the *super-schmoozers*. (He figures he can make more money from advertisers by charging a premium for ads displayed to super-schmoozers.) A *super-schmoozers* is a person on LinkedIn who has a link to at least 20 other super-schmoozers on LinkedIn.

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 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  $G$ .

Hint: I bet there are some vertices you can rule out immediately as not super-schmoozers.

**2. (25 pts.) A funky kind of coloring**

Let  $G = (V, E)$  be an undirected graph where every vertex has degree  $\leq 51$ . 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 this algorithm terminates, 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.

(b) Prove that the algorithm terminates after at most  $|E|$  iterations of the loop.

Hint: I suggest that you figure out what is the largest value the potential could take on.

Comment: You might enjoy thinking about how to implement the algorithm so that its total running time is  $O(|V| + |E|)$ —but we will not grade it.

### 3. (50 pts.) Scheduling

This problem will give you a tour through several scheduling algorithms with interesting practical applications.

(a) You are writing a scheduler for an operating system running on a machine with a single CPU. You have  $n$  jobs to schedule, and the  $i$ th job will take  $t_i$  seconds. Only one job can run at a time, so you must decide what order to run them in. There are no dependencies between the jobs, so there are no constraints on what order you can schedule them in. The goal is to find an ordering that minimizes the average time at which a job finishes, averaged over all jobs.

For instance, suppose we have three jobs, and suppose you decide to execute the jobs in the order 1, 2, 3. Then job 1 completes at time  $t_1$ , job 2 completes at time  $t_1 + t_2$ , and job 3 at time  $t_1 + t_2 + t_3$ . Therefore, in this example the average is  $[t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3)]/3$ .

Design an efficient algorithm to find such an ordering, given  $t_1, \dots, t_n$ .

(b) Now suppose that in addition to the times  $t_1, \dots, t_n$ , we are also given weights  $w_1, \dots, w_n$  that represent how important each job is (the higher the weight, the more important the job is). Assume that  $w_1 + \dots + w_n = 1$  and  $0 < w_i \leq 1$  for each  $i$ . The goal is to find an order that minimizes the *weighted* average. For instance, if we execute  $n = 3$  jobs in the order 1, 2, 3, the weighted average will be  $w_1 t_1 + w_2(t_1 + t_2) + w_3(t_1 + t_2 + t_3)$ .

Design an efficient algorithm to find such an ordering, given  $t_1, \dots, t_n$  and  $w_1, \dots, w_n$ .

(c) We buy another identical machine, so now we have two machines. Each job must be assigned to one of the two machines. Suppose what we care about is the final completion time when the last job is finished (the weights are irrelevant now). To put it another way: let  $S_1$  denote the set of jobs assigned to the first machine, and  $S_2 = \{1, 2, \dots, n\} \setminus S_1$  the set of jobs assigned to the second machine. The completion time for the first machine is  $C_1 = \sum_{i \in S_1} t_i$ , and the completion time for the second machine is  $C_2 = \sum_{i \in S_2} t_i$ . The overall completion time is  $C = \max(C_1, C_2)$ , and we'd like to find an assignment of jobs to machines that makes  $C$  as small as we can.

Consider this greedy algorithm. We process the jobs one by one, assigning job  $i$  to whichever machine completes earlier given the assignment of the first  $i - 1$  jobs. In pseudocode:

1. Set  $c_1 := 0$  and  $c_2 := 0$ .
2. For  $i := 1, 2, \dots, n$ :
3.     If  $c_1 < c_2$ , then assign job  $i$  to machine 1 and set  $c_1 := c_1 + t_i$ .
4.     Otherwise, assign job  $i$  to machine 2 and set  $c_2 := c_2 + t_i$ .

Prove that the completion time of the solution output by this algorithm is at most 1.5 times larger than the completion time of the best possible ordering.

Hint: Let  $a = \frac{1}{2} \sum_{i=1}^n t_i$  and  $b = \max_i t_i$ . Let  $C^*$  denote the overall completion time of the optimal ordering. Show that  $C^* \geq a$ , and  $C^* \geq b$ . Give an upper bound on  $|C_1 - C_2|$ , as a function of  $b$ ; an upper bound on  $|C_1 - a|$ , again as a function of  $b$ ; and similarly for  $|C_2 - a|$ .

(d) We buy a whole bunch of identical machines, so that we have a total of  $m$  machines. Suppose we apply the greedy algorithm in part (c). In pseudocode:

1. Set  $c[i] := 0$  for each  $i := 1, 2, \dots, m$ .
2. For  $i := 1, 2, \dots, n$ :
3. Find  $j$  that minimizes  $c[j]$  (i.e., choose  $j$  so that  $c[j] \leq c[k]$  for all  $k$ ).
4. Assign job  $i$  to machine  $j$ , and set  $c[j] := c[j] + t_i$ .

Give a good upper bound on the ratio between the completion time of the greedy algorithm above and the completion time of the optimal ordering, as a function of  $m$ . In other words, you should come up with a small constant factor  $r_m$  (which may depend upon  $m$  but should be independent of  $n$ ), so that the completion time of the greedy algorithm on any input is at most  $r_m C^*$ , where  $C^*$  is the completion time of the optimal ordering on this input. Make sure to prove your answer, and to provide a formula for  $r_m$  as a function of  $m$ .

(In part (c) we saw that  $r_2 = 1.5$ . Generalize the analysis from part (c) to arbitrary  $m$ .)