CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #2, due Friday, May 20th, 2016 at 2:15pm in Room x235, Box 32

One mark will be deducted if your solution uses multiple sheets of paper that are not stapled.

[10] 1. Let us consider how cycles, connectivity and searching works in directed graphs. (See Section 3.5 of the text.)

A path in a directed graph is much the same as in an undirected graph, except that all edges must be pointed in the same direction. So, a path is a sequence of vertices $v_1, \ldots, v_k$ such that $v_1, \ldots, v_k$ are distinct vertices, and (v_i, v_{i+1}) is a directed edge for every i.

A cycle in a directed graph is much the same as in an undirected graph, except that all edges must be pointed in the same direction. So, a cycle is a sequence of vertices $v_1, \ldots, v_k$ such that $v_1, \ldots, v_{k-1}$ are distinct vertices, $v_1 = v_k$, and (v_i, v_{i+1}) is a directed edge for every i. The case $k = 2$ also counts as a directed cycle.

A directed graph is called *strongly connected* if, for every ordered pair of vertices u and v, there is a directed path from u to v. (So there must be one from v to u as well.)

Breadth-first search (BFS) in a directed graph works much like BFS in an undirected graph, except that edges must be followed in the right direction: from their tail towards their head.

We say that a vertex v can "reach all vertices" if running a BFS starting at vertex v results in all vertices being visited.

[5] (a) Let G be a directed graph with at least two vertices. Suppose that there exist two different vertices s_1 and s_2, both of which can reach all vertices. Prove or disprove: G must have a directed cycle.

[5] (b) Let G be a directed graph with at least two vertices. Suppose that there exist two different vertices s_1 and s_2, both of which can reach all vertices. Prove or disprove: G must be strongly connected.

[15] 2. A small photocopying service with a single large machine faces the following scheduling problem. Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take t_i seconds to complete. Given a schedule (i.e., an ordering of the jobs), let C_i denote the finishing time of job i. For instance, if job j is the first to be done, then $C_j = t_j$; and if job j is done right after job i then $C_j = C_i + t_j$.

Each customer i also has a weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times:

$$\sum_{i=1}^{n} w_i C_i.$$

For example: suppose there are two jobs. The first job takes times $t_1 = 3$ and has weight $w_1 = 12$, while the second job takes time $t_2 = 1$ and has weight $w_2 = 3$. Then doing job 1 first would yield a weighted completion time of $12 \cdot 3 + 3 \times 4 = 48$. Doing job 2 first would yield a weighted completion time of $3 \times 1 + 12 \times 4 = 51$.

[5] a. Design an efficient algorithm to solve this problem. That is, you are given a set of n jobs with a processing time t_i and a weight w_i for each job, and you want to order the jobs so as to minimize the weighted sum of the completion times. Hint: the algorithm can be described in one line.

[10] b. Prove the correctness of your algorithm from part (a). Hint: what happens to the weighted completion time if you swap the first "out-of-order" job with the previous one?

[15] 3. This problem is related to the interval scheduling problem that was discussed in the lectures. Suppose that, instead of being a would-be attendee, you are now the person in charge of providing security for these events. To simplify the problem, let us assume that one guard is needed for each event. A guard can of course provide security at multiple events as long as they do not overlap. The cost of providing security will be computed as follows:

- There is a fixed cost c to hire a guard.
- There is a per-hour cost paid to provide security at each event.

Since the sum of the lengths of the events is fixed, the only control you have over the total cost is through the number of guards you hire. To minimize the total cost, you thus want to hire as few guards as possible.

[3] (a) One of your friends suggests using the following algorithm to assign guards to events:

while *there are events left* **do**
  hire a guard g
  find a largest set E of non-overlapping events
  assign g to guard the events in E
end

where the set E is found using the algorithm discussed in class. Prove that this algorithm does **not** always return a minimal set of guards.

[7] (b) Describe an algorithm that finds a minimal set of guards and runs in $o(n^2)$ time.

[2] (c) Analyze the worst-case running time of your algorithm from part (b).

[3] (d) Prove that your algorithm hires the fewest number of guards possible. Hint: the proof should be short.

[1] 4. (Bonus) How long did it take you to complete this assignment (not including any time you spent revising your notes before starting?)