Homework 3: Greedy

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less.

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to Gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Find a Counterexample [10 points]

For a graph $G = (V, E)$, a set $S \subseteq V$ a "vertex cover" if $\forall (u, v) \in E : (u \in S \vee v \in S)$. In English, a vertex cover is a subset of the vertices so that every edge has at least one of its endpoints in the set.[1] We are usually interested in small vertex covers (since V is always a vertex cover). Consider the following greedy algorithm for finding a small vertex cover:

function GREEDYVERTEXCOVER($G = (V, E)$)
  $S \leftarrow \varnothing$
  while E is not empty **do**
    Let u be the highest degree vertex ▷ not counting deleted edges
    Add u to S
    Delete all edges incident to u from E ▷ we know these are covered by u and can ignore them
  return S

Prove that this algorithm is incorrect by counterexample. Specifically, give an example of a graph on which this algorithm does not produce the smallest vertex cover. Show both the vertex cover that the algorithm finds and a smaller cover.

2. Opening up the STP [25 points]

Every summer on a weekend in July there is a bike ride from Seattle to Portland (the STP) beginning at the Montlake parking lot and ending in Portland. Registered finishers get a T-shirt when they arrive at the end.

You have been asked to figure out how to open up the ride to more participants who do not have to register for the whole distance. Instead, participants only need to register with a start milepost and end milepost (each a decimal number) along the route and, instead of getting finishing T-shirts, participants can pick up STP logo participant water bottles at some station along their route.

Your job is to set up a number of stations so that you can hand out water bottles to every participant at some point along their registered portion of the STP.

Describe an efficient algorithm that, given the start and finish mileposts of n registrants, finds a sequence of mileposts for stations that minimizes the number of stations that you have to set up between Seattle and Portland. (It is OK for there to be more than one station along someone's route; each participant will be told which of the stations along their route they can collect their water bottle.)

You should be able to do this using $O(n \log n)$ time; of course you need to prove your claims.

[1] The name is somewhat counter-intuitive; the set contains vertices – we say "the vertices cover the edges" so it's the **edges** that are covered (by the *vertices*) in a *vertex* cover.

3. The Waiting is the Hardest Part [25 points]

You are in the business of renting a high end video-conferencing studio for those who are very tired of Zoom and want much better production values. Each reservation request you receive for your studio is for a fixed duration t_i in hours and involves p_i people participating. You happen to know that no person participates in more than one reservation request.

You want to schedule all of these requests at your studio, which can only handle one request at a time, in the best way possible according to the following criterion: Each of the people participating wants to have their respective conference over as soon as possible so you want to schedule them to minimize the total person-hours that participants need to wait until their video conference is finished. If you think of the start of your schedule as time 0 and have scheduled request i to finish at time f_i, the total person-hours of waiting for that request would be $p_i f_i$.

Design an efficient algorithm that takes as input n pairs consisting of the duration t_i and number of participants p_i for the i-th request and produces a schedule that minimizes the total person-hours of waiting summed over all requests.

4. Beans [25 points]

As the lucky millionth customer of your local bean store, the owners have allowed you to fill up a bucket with any beans for free. They give you a comically-large bucket, so you don't need to worry about volume, but weight may be a problem: you decide you'll only be able to carry up to b pounds of beans. At the bean store, there are n different types of beans to choose from and bean i costs c_i dollars per gallon and weighs w_i pounds per gallon. Unfortunately, because of supply chain issues, the store is running low on beans. Each type has s_i gallons currently in stock. Note that since you're dealing with beans, you can fill your bucket with non-discrete quantities of beans (e.g., 0.5 gallons or $\sqrt{2}$ gallons). You wish to fill this bucket with the highest cost combination of beans that won't go over your weight limit.
As former Algorithms students themselves, the store owners clue you in that a greedy algorithm might work. In this problem you will describe some "greedy rules" to summarize algorithm ideas. A "greedy rule" is a clear description someone could follow to select your beans (e.g., an ordering of bean types and rule for how much to include of each).[2]

(a) Describe a greedy rule that would **not** produce the optimal value. Present a counterexample where the greedy rule chooses a suboptimal assortment of beans, also give both the selection made by the greedy algorithm and the better selection.

(b) Describe a greedy rule that would produce the optimal value. Present a proof showing that the greedy algorithm with the described rule always chooses an optimal assortment of beans.

[2]For an example: "Sort the edges in increasing order and add an edge as long as it doesn't create a cycle" is a greedy rule for MSTs.