

CS 161 (Stanford, Winter 2022) Homework 7

Style guide and expectations: Please see the “Homework” part of the “Resources” section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.

What we expect: Make sure to look at the “**We are expecting**” blocks below each problem to see what we will be grading for in each problem!

Exercises. The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

1 Greedy Strategies

Let $G = (V, E)$ be an undirected unweighted graph. Say that a vertex v can “see” an edge if v is an endpoint of that edge. Say that a set $S \subseteq V$ is “all-seeing” if every edge $e \in E$ is seen by at least one vertex in S . In this problem, we will try to greedily construct the smallest all-seeing set possible. Consider the following greedy algorithm to find an all-seeing subset:

Algorithm 1: findAllSeeingSubset

Input: G : undirected unweighted graph
 $S = \{\}$ **while** G contains edges **do**
 choose an edge $e = (u, v)$ in G $S.add(u)$ $S.add(v)$ remove u and all of its
 adjacent edges from G remove v and all of its adjacent edges from G
return S

1.1 (2 pt.)

Prove that $findAllSeeingSubset(G)$ always returns an all-seeing subset. [**We are expecting:** A short but rigorous proof.]

1.2 (2 pt.)

Give an example of a graph on which $findAllSeeingSubset(G)$ does not return a smallest all-seeing subset, for at least one way of choosing edges. [**We are expecting:** An example graph and a brief justification why it does not return a smallest all-seeing subset.]

Problems. The following questions are problems. You may talk with your fellow CS 161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
 - Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
 - If you collaborated, list the names of the students you collaborated with at the beginning of each problem.
-

2 Plucky's Slushies

Plucky the penguin has a cooler with a capacity of Q ounces. Plucky heads to the convenience store where there are n flavors of slushies. Each slushy flavor i has a cost per ounce $v_i > 0$ (measured in units of dollars per ounce) and a quantity $q_i > 0$ (measured in ounces). There are q_i ounces of slushy i available to Plucky, and for any real number $x \in [0, q_i]$, the total cost from x ounces of slushy i is $x \cdot v_i$.

Note that Plucky can take a fractional amount of each slushy. For example, perhaps there is 36 ounces of blue raspberry flavored slushy; Plucky can choose to put 12.345 ounces of blue raspberry slushy in the cooler.

Plucky wants to create the most expensive slushy concoction in the cooler. Thus, Plucky wants to choose an amount $x_i \geq 0$ to take for each flavor i in order to maximize $\sum_i x_i v_i$ the total cost while satisfying:

1. Plucky does not overfill the cooler (that is, $\sum_i x_i \leq Q$), and
2. Plucky does not take more of a flavor than is available (that is, $0 \leq x_i \leq q_i$ for all i).

Assume that $\sum_i q_i \geq Q$, so there always is some way to fill the cooler.



2.1 Which flavor? (0 pt.)

Suppose that Plucky has already have partially filled the cooler, and there is some amount of each slushy flavor left. Which flavor should Plucky choose next, and how much? **[We are expecting:** Nothing, this part is worth zero points, but it's a good thing to think about before you go on to the next part.]

2.2 Greedy Algorithm (4 pt.)

Design a **greedy algorithm** which takes as input Q along the tuples (i, v_i, q_i) for $i = 0, \dots, n-1$, and outputs tuples (i, x_i) so that conditions (1) and (2) hold and $\sum_i x_i v_i$ is as large as possible. Your algorithm should take $O(n \log(n))$. Note that the tuples may be returned in

any order. **[We are expecting:** Pseudocode AND an English explanation of what it is doing. A justification of the running time.]

2.3 Proof of Correctness (4 pt.)

Complete the inductive step below to prove that your algorithm is correct.

- **Inductive hypothesis:** After making the t 'th greedy choice, there is an optimal solution that extends the solution that the algorithm has constructed so far.
- **Base case:** Any optimal solution extends the empty solution, so the inductive hypothesis holds for $t = 0$.

[We are expecting: A proof of the inductive step: assuming the inductive hypothesis holds for $t - 1$, prove that it holds for t . A proof by induction conclusion.]

2.4 Ethics Questions

2.4.1 (2 pt.)

After making the most expensive slushy, Plucky realizes that it's actually quite tasty. Using some of the remaining flavors, he creates another slushy that is less expensive but not as tasty. Then Lucky comes along, hoping to buy one of Plucky's slushies. However, he doesn't know which one to choose: Slushy A, which is tastier, or Slushy B, which is less expensive.

Using the concepts of incomparability and incommensurability, explain why Lucky is having a difficult time choosing between the two slushies.

- Two *kinds of things* are incomparable when a one-unit increase in either wouldn't influence your choice.
- Two things are incommensurable when we lack a *common measure* of value. Incommensurability makes it difficult to establish ranking relationships, such as "more than" or "less than," "better than" or "worse than."

[We are expecting: Two to four sentences explaining whether you think that the two slushies are incomparable or their values incommensurable; and how this may affect Lucky's decision.]

2.4.2 (2 pt.)

Plucky and Lucky are now business partners and are hoping to increase their revenue by selling slushies at the highest price that customers are willing to pay. Describe how this business problem may be translated to a computational problem. Explain whether and why this would constitute a "negotiated translation."

"Negotiated translation:" formalization of a business problem into an algorithmically tractable

problem, which is contingent on the discretionary judgement of stakeholders.

[We are expecting: Three to five sentences describing: (1) a computationally tractable problem that Plucky and Lucky might use as a translation of their business problem; and (2) what discretionary decisions may go into the translation, including alternative problem formulations, if there are any, what goals it prioritizes and what goals it sets aside.]

3 Monkey Island (pt.)

One day, you get stranded on a beach on an island with n monkeys. You will be here a while, so you want to make friends and introduce yourself to all the monkeys. However, you don't want to leave the beach since the rest of the island looks dangerous. Each monkey i will only be on the beach during one time interval $[a_i, b_i]$, inclusive. You can only introduce yourself to monkey i during this time interval. Your plan is to stand in the center of the beach and use the megaphone you brought with you to say "Hello" at certain times t_1, \dots, t_m . Any monkey on the beach at one of the times $t_j \in \{t_1, \dots, t_m\}$ will hear your introduction. It's okay if a monkey hears your introduction more than once, but you want to make sure every monkey hears your introduction at least once.

3.1 Greedy Algorithm (6 pt.)

The battery in your megaphone is low, so you want to minimize m , the number of times you use your megaphone. **Devise a greedy algorithm** that takes as input the list of intervals $[a_i, b_i]$ and outputs a list of times t_1, \dots, t_m such that m is as small as possible but every monkey hears your introduction. Your algorithm should run in time $O(n \log n)$ where n is the number of monkeys on the island. You can assume every monkey visits the beach at some point. **[We are expecting:** Pseudocode and an English description of your algorithm, as well as a short justification of the runtime]

3.2 Proof of Correctness (4 pt.)

Give a formal proof that your algorithm is correct. **[We are expecting:** A proof by induction]

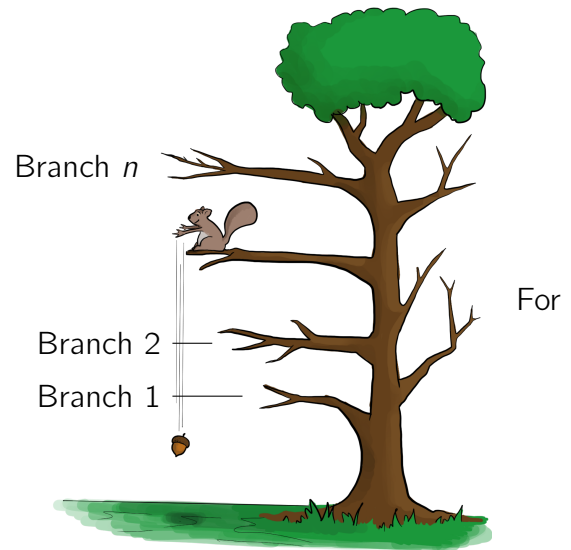
4 More Dynamic Programming!

Socrates the Scientific Squirrel lives in a very tall tree with n branches, and she wants to find out what the lowest branch $i \in \{1, \dots, n\}$ is, so that an acorn will break open when dropped from branch i . If an acorn breaks open when dropped from branch i , then an acorn will also break open when dropped from branch j for any $j \geq i$. (If no branch will break an acorn, Socrates should return $n + 1$).

The catch is that, once an acorn is broken open, Socrates will eat it immediately and it can't be dropped again.

Compute the minimum number of drops Socrates needs in the worst case, given that she has k acorns.

$n \geq 0$ and $k \geq 1$, let $D[n, k]$ be the *optimal worst-case number of drops* that Socrates needs to determine the correct branch out of n branches using k acorns. That is, $D[n, k]$ is the number of drops that the best algorithm would use in the worst-case.



4.1 (3 pt.)

For any $1 \leq j \leq k$, what is $D[0, j]$? What is $D[1, j]$? For any $1 \leq m \leq n$, what is $D[m, 1]$? (Note that if $n = 0$, then Socrates needs zero drops to identify the correct branch, since there are no branches). **[We are expecting:** Your answer. No justification required.]

4.2 (2 pt.)

Suppose the best algorithm drops the first acorn from branch $x \in \{1, \dots, n\}$. Write a formula for the optimal worst-case number of drops remaining in terms of $D[x - 1, k - 1]$ and $D[n - x, k]$. **[We are expecting:** Your formula and an informal explanation of why this formula is correct.]

4.3 (2 pt.)

Write a formula for $D[n, k]$ in terms of values $D[m, j]$ for $j \leq k$ and $m < n$. **[Hint:** Use part 4.2.] **[We are expecting:** Your formula and an informal explanation of why this formula is correct.]

4.4 Dynamic Programming Algorithm (5 pt.)

Design a dynamic programming algorithm which will compute $D[n, k]$ in time $O(n^2k)$. **[Hint:** Use parts 4.1 and 4.3.] **[We are expecting:** Pseudocode AND a brief English description of how it works, as well as an informal justification of the running time. You do not need to justify that it is correct.]