CSE 417 25au Homework 6: Greedy Algorithms

Released: Friday, November 7, 2025 @ 11:30am

First due by: Friday, November 14, 2025 @ 11:59pm

Last resubmissions by: Wednesday, November 26, 2025 @ 11:59pm

Instructions

For Problems 11, 11X.1, and 12, you have four options for submission:

- **Film a video in which you explain your solution.** See the Homework Guide for more details.
- **Use LaTeX to type your solutions.** A template is provided in the "Tasks" page of the course website, if you like.
- **Use Google Docs or Microsoft Word to type your solutions.** If doing so, please use the Equation Editor to ensure that any equations are legible and easy to read.
- **Handwrite your solutions on paper or digitally.** Please write neatly and if on paper, scan in black/white mode, not grayscale.

We prefer either video or LaTeX, but accept any of the 4 options.

A few more reminders:

- **Submit all problems on Canvas.** Each problem should have its own submission. *Do not* submit one large file containing answers to several problems.
- **Suggested word counts are rough guidelines.** We won't actually count, but if your writing is verbose to the point of obscuring your main argument, we may ask you resubmit more concisely.
- **Review the collaboration policy in the syllabus**. Collaboration is encouraged but strict rules apply, and remember to cite your collaborators.
- If you don't finish in time, we encourage you to be honest and just upload what you have so far. Resubmission won't cost you anything, and we can give you timely feedback on your partial progress by submitting on time.

Happy problem solving!

Problem 11: Company-wide decision

The purpose of this problem is to reinforce and extend the Huffman codes discussed in class.

A large company wants is at a critical junction and needs to make a decision about a topic. Since it's not a simple yes/no question and implementation details are critical to the plan, the company decides to use the following method to make a decision.

- Each team will first come up with their own plan. Each initial group consists of one team.
- Over the next several weeks, pairs of groups will have a joint 1-hour meeting to discuss their plans. They will compromise and modify their plans so that by the end of the 1-hour meeting, both groups are on the same page and support the same plan. From then on, the two groups merge into one group, for the purpose of going on to meet with even more groups.
- Even if a group is very large, it is critical that the entire group attends the meeting, so that all of its members are familiar with the plan.
- The process ends when the entire company is present at the same meeting, and agrees on a final plan at the end.

Suppose the company has n people in total, split up among m teams of various sizes. You can assume that each team has at least 1 person, and each person is on exactly one team. These sizes are given to you in an array $A[1 \ldots m]$. In what order should the teams meet up, so that the total person-hours spent in meetings is minimized?

1. Before solving the problem, consider a bad solution: Ignore the team sizes and have teams 1 and 2 meet and merge. Next, have the new group meet with team 3 and merge. Next, with group 4, and so on until the entire company is in the same group.

   Explain why if every team initially consists of 1 person, then this solution will take $\Theta(n^2)$ person-hours. (suggested 15–50 words)

2. Describe a greedy algorithm that outputs a meeting order that minimizes total person-hours spent in meetings. (Hint: It may not seem like it at first glance, but this is very closely related to Huffman codes!)

   It is enough to say by what criteria you will select the next two groups to meet, no pseudocode required. (Though if you prefer to give pseudocode, feel free.) (suggested 5–30 words)

3. Use an exchange argument to explain why your greedy choice must always be true in some optimal solution. (Hint: If you used an algorithm similar to Huffman codes, the argument should also be similar!) (suggested 100–200 words)

4. Using big-O (or big-Θ), how many person-hours will your greedy algorithm take if every team initially consists of 1 person? (suggested 50–100 words)

(Video solutions are suggested to take 3–6 minutes total.)

Problem 11X: Huffman coding (Extensions)

This is an extension problem that doesn't build so much on the ideas of Problem 11, but builds on the same Huffman coding background as Problem 11. **Pick one** of the following to complete:

1. In 1951, David Huffman was a student at MIT, taking a class taught by Robert Fano. Fano had been working on the problem of finding good prefix codes, and told the class that if they could solve the optimal prefix code problem, they could skip the final exam. Huffman struggled on this problem for the whole semester, and when he was just about to give up and begin studying for the final exam, he solved the problem and gave us the Huffman codes that we studied in class.

   This surprised Fano, because in fact, both he and the "father of information theory" Claude Shannon had just spent months struggling with the problem as well, to much less success. Shannon's algorithm is stated below, for input frequencies that sum to 1:

   1: Sort the frequencies so that $f_1 \geq f_2 \geq \cdots \geq f_n$.
   2: Let $\ell_i = \lceil \log_2(1/f_i) \rceil$ be the length of the encoding of character i.
   3: **for** $i = 1, 2, \ldots, n$ **do**
   4: Let the encoding of i be the lexicographically first code of length ℓ_i that does not contain already-chosen codes as a prefix.

   For example, with frequencies $1/2$, $1/4$, $1/8$, and $1/8$, Shannon's algorithm would first compute the lengths to be 1, 2, 3, and, 3, respectively. The lexicographically first code of length 1 is "0". The lexicographically first code of length 2 that does not start with "0" is "10". Then the third symbol gets "110" and the fourth symbol gets "111".

   (a) Explain why Shannon's algorithm produces a feasible solution. This means that for Shannon's choice of lengths ℓ_i, you will not run out of codes as you choose them. (That is, for example, it will not ask you to pick 3 different codes of length 1.) (suggested 50–100 words)

   (b) Shannon's algorithm is *not* optimal. Find an example input for which Shannon's algorithm produces a worse prefix code than Huffman's optimal algorithm. Show and explain all calculations needed to support your claim. (no suggested word count, just give the input and show the calculations)

   (Video solutions are suggested to take 3–6 minutes total.)

2. When you write a program, it is always important to test it on a few inputs and verify that they produce the correct outputs. Testing is especially important for greedy algorithms, since it is easy to generate many correct-sounding greedy strategies that are actually suboptimal. However, it is easy for manual tests to miss edge cases, or in general, fail to catch bugs.

   If it is crucial for your application that the code has zero bugs, one better technique is to do exhaustive testing over small inputs. We will do this for the optimal prefix code problem. Breaking this down into parts, you will do the following, some of which will be provided as starter code:

- Given a number of symbols n and a maximum string length N, generate all strings of length $\leq N$ on n symbols. (These are the "small inputs".)
- Calculate the frequency of each symbol in the string.
- Suggest a reasonable greedy attempt at computing prefix codes that does NOT always produce the optimal prefix code. It should still produce a prefix code, just not the optimal one. Do not copy the method in Problem 11X.1. **For this part, leave a comment in your code that explains in one or two English sentences what your greedy strategy is, to aid grading.**
- Find a counterexample within the set of small inputs generated by comparing the attempt with a brute-force solution.

The starter code and a more detailed spec is available both on the course website and Canvas. Submit your code in the assignment on Canvas (which redirects to Gradescope) or Gradescope directly, where it will be autograded. We will manually verify that your attempted strategy for computing prefix codes is a greedy strategy. Do not use libraries beyond the standard Java API.

Problem 12: Eating ice cream

Nathan's favorite ice cream in town is Molly Moon's ice cream. A special thing that you can actually order at Molly Moon's is called an "ice cream flight", which is a tray filled with a small scoop of every one of their ice cream flavors! There is one (well, at least one) clear drawback from ordering this ice-cream flight. The different flavors melt at different rates! (This is realistic, since cream and sugar ratios affect melting rate.) And to complicate things further, the different add-ins to each flavor (possibly real bubble gum, for instance) cause each flavor to take a different amount of time to eat as well. To avoid having a lukewarm liquid mess before we get a chance to try them all, we need to develop a strategy for tackling the frozen confection.

We have done our research and know the rate at which each flavor melts and the amount of time it will take us to eat the scoop of each flavor. The input to the algorithm will be a list of n pairs (m_i, t_i) representing the melt rate (in milliliters per second) and eating time (in seconds) of each of the n ice cream flavors. Melting does not affect the time it takes to eat the ice cream (you would eat the liquid melted ice cream). Feel free to also assume that you do not take breaks, and are constantly eating ice cream until you are finished. Therefore, you finish at time exactly $t_1 + t_2 + \cdots + t_n$.

Assume that no scoop can completely melt before time $t_1 + t_2 + \cdots + t_n$, and that the ice cream continues to melt while you are eating it. So if we finish eating flavor i at time f_i, then the amount melting of flavor i will have been $m_i f_i$. The total melting is therefore given by $m_1 f_1 + m_2 f_2 + \cdots + m_n f_n$. Our goal is to decide on the order to eat the ice cream to minimize the total amount of melting that occurs.

Your job is to design a greedy algorithm that takes as input n pairs consisting of the eating time t_i and melting rate m_i for the ith ice cream flavor and produces an eating order that minimizes the total melting.

1. The following attempt at a greedy algorithm does *not* work: Eat the ice cream takes the lowest time to eat t_i first. Give a counterexample where this algorithm produces a suboptimal solution. (That is, state your example's input pairs, calculate the algorithm's solution, and show one feasible solution that is better.) (no suggested word count)

2. The following attempt at a greedy algorithm does *not* work: Eat the ice cream has the highest melting rate m_i first. Give a counterexample where this algorithm produces a suboptimal solution. (That is, state your example's input pairs, calculate the algorithm's solution, and show one feasible solution that is better.) (no suggested word count)

3. Design a greedy algorithm that solves this problem. It is enough to say by what criteria you will select the next ice cream flavor to eat, no pseudocode required. (Though if you prefer to give pseudocode, feel free.) (suggested 5–30 words)

4. Use an exchange argument to explain why your solution minimizes the total melting. Be sure to explain mathematically why swapping your choice into another solution that differs would decrease the other solution's total melting (or stay the same). (suggested 50–100 words)

(Video solutions are suggested to take 3–6 minutes total.)