PA5: Greedy Choice(s)

This programming assignment is to be completed in either Java or Python (your choice) and submitted on Gradescope.

Collaboration is restricted to just 2 people (other than yourself), and it must be "Whiteboard Only". In summary, this means that you may discuss the assignment with 2 others, but you cannot preserve any artifacts (digital or physical) from these discussions. In particular, this means you may not take and keep photos, notes, video, audio recordings, whiteboard contents, etc. from any discussions.

All collaborators and other resources consulted should be listed as a comment at the top of your code submission. Course staff and official course materials do not need to be listed.

Motivation

The challenge in a greedy algorithm lies in a different place compared to any other algorithm design in that greedy algorithms are typically trivial to implement, but challenging to justify correctness. Accordingly, this programming assignment is structured to give you experience in gaining confidence in the correctness of greedy algorithms.

You will be asked to implement the greedy choice function for three different algorithms. All three algorithms have identical input/output formats, and the procedure of all three algorithms is precisely the same, except that they may differ by their greedy choice function.

For each algorithm the input will be a list of pairs of doubles/floats. The task of the algorithm is to determine the best way to order this list in order to minimize some objective. The implementation of each algorithm will work like a selection sort, which sorts a list by repeatedly finding the minimum element and placing that next in the list. Your greedy choice function essentially serves the role of the

"find minimum" step. Given a list of pairs of floats/doubles, your algorithm will return the index of the value which should be first in the optimal ordering. The rest of the code provided will then repeatedly apply your greedy choice to construct the final list.

Since there are relatively few options for what the correct greedy choice might be, we will *not* be providing you results from automated tests for this assignment. Instead, it is up to you to develop your own confidence in the correctness of your submission by either testing your code or else by proving correctness of your code via exchange argument. You will actually be requested to prove the correctness of the greedy choice functions for some of these algorithms as part of PS3, so this assignment will be giving you a head start for that! Until the deadline, the autograder will only run your code against the provided test cases primarily so that you can make sure it runs properly. Additional tests will be added and run against your code after the deadline has passed.

This pattern actually matches my recommendation for how to develop greedy algorithms. You come up with a hypothesis about what a correct greedy choice might be, try it on many test cases trying to break it, and when you break it try to develop an intuition for what might be better.

Below we describe the objective for each of the three problems.

Problem 1: Bakeoff

To make a cake, a baker must first make all of the cake's components (e.g. sponge, icing, jam, etc.). Once the components are made the baker will assemble the cake. Let's find an order to make all of the components to finish the cake as quickly as possible.

Each component of a cake requires "active time" followed by "passive time" to prepare. During its "active time", the baker must dedicate undivided attention. During its "passive time" the baker may be doing something else. For example, a Victoria sponge takes 20 minutes of active time mixing together the ingredients, then 60 minutes of passive time for it to bake then cool. Making jam takes 15 minutes of active time of stirring on the stovetop and then 25 minutes of passive time to chill in the fridge. A baker can only actively work on one component at a time and must finish that component's active work before moving onto the next.

There are n components of the cake. Each component $c_i = (a_i, p_i)$ requires a_i minutes of active time followed by p_i minutes of passive time to complete. Using your greedy choice function we will create an algorithm to compute the optimal order in which the baker should make their cake components to finish the cake as early as possible (i.e. it minimizes the time that the all the components' passive times are complete).

As an example, if there are two components c_1, c_2 where $a_1 = 4, a_2 = 2, p_1 = 10, p_2 = 5$, then the two possible preparation sequences are:

– c_1, c_2: In this case, c_1 will be ready for assembly after $4 + 10 = 14$ minutes and c_2 will be ready after $4 + 2 + 5 = 11$ minutes. Therefore the cake will be finished in 14 minutes $(max(14, 11))$

– c_2, c_1: In this case, c_2 will be ready for assembly after $2 + 5 = 7$ minutes while c_1 will be ready after $2 + 4 + 10 = 16$ minutes. Therefore the cake will be finished in 16 minutes $(max(7, 16))$.

The optimal order in this case is c_1, c_2 since the cake will be finished sooner.

Problem 2: Deadlines

You probably have, like, a lot of homework. Let's say you have n total assignments that we'll label $a_1, \ldots, a_n$ in your product. Each assignment $a_i = (t_i, d_i)$ has a deadline d_i when it is due and an estimated amount of time it will take to complete, t_i.

Let us assume that when you work on one assignment, you give it your full attention and do not begin any other assignment (yes, this is different from how I recommend you do CS3100 homework, but it makes the problem harder without this assumption). In some cases, your demanding professors ask too much of you. It may not be possible to implement all of your homework on time given the deadlines $d_1, \ldots, d_n$; indeed, some homework may *have* to be delivered late. Your goal as a responsible student is to minimize the lateness of your latest assignment.

If you begin assignment a_i at time s_i, you will complete it at time $c_i = s_i + t_i$. The lateness value, denoted ℓ_i, for a_i is the value

$$\ell_i = \begin{cases} c_i - d_i & \text{if } c_i \geq d_i \\ 0 & \text{otherwise} \end{cases}$$

Using your greedy choice function, we will produce schedule, call it O, that specifies the order in which you implement features which minimizes the maximum ℓ_i for all assignments, i.e.

$$\min_O \max_i \ell_i$$

In other words, you do not want to complete in any assignment too late, so you minimize the lateness of the latest assignment you submit.

Problem 3: Mileage

A local car dealer sold n cars today! The dealership will now deliver the cars to the customers. The problem is that the dealership can only deliver one car per day, and gas prices are increasing fast! Help the dealership to select an order to deliver the cars in order to minimize the amount they must spend on gas.

Each car $c_i = (e_i, m_i)$ has a fuel economy of e_i (in miles per gallon of gasoline used) and must be delivered m_i miles away. On the first day of the deliveries gas costs \$2 per gallon, but it is increasing in value at a rate of 5 percent per day (compounding). This means that on the second day it will cost \$2.10 and on the third it will cost \$2.205. Using your greedy choice function, we will develop an algorithm which minimizes the total amount the dealership must spend to deliver all n cars.

Starter Code Summary

The starter code provided consists of two files:

- **GreedySolver.java / greedysolver.py**: This is implements the algorithm which will produce the optimal solution from your greedy choice. You should not need to make any changes here, but you will likely want to know what's going on for the purposes of debugging.
  - `Main`: The main method reads input from the console and then creates a "GreedySolver" object. it then repeatedly applies your greedy choice function and uses it to construct the final output. This final output is then printed along with the cost of the schedule produced.

  - `cost_problem`: For each of the three problems there is a cost function. This cost function calculates the cost of the schedule produced by repeatedly applying your greedy choice function. To summarize each: `cost_bakeoff` returns the time that the last component is completely finished,
  - `cost`: simply calls the correct `cost_problem` for the problem we're currently solving.
  - `solve`: This implements the "selection sort" algorithm to build a schedule using your greedy choice function. It will call your greedy choice function, place the item at the index you return as first in the schedule, call your greedy choice function again with that item removed, place the item at the index you return as the second in the schedule, and then repeat until all items have been added to the schedule.
- **GreedyChooser.java / greedychooser.py**: This is the file that you will modify and submit. Each function included is the greedy choice function for one of the problems described above.
  - `bakeoff`: The input is a list of pairs of values. Each ordered pair represents the active time and passive time, respectively, of each cake component. You should return the index of which component should be made first to finish all components earliest.
  - `deadlines`: The input is a list of pairs of values. Each ordered pair represents the time it takes to complete an assignment and its deadline, respectively. You should return the index of which assignment should be completed first to minimize the lateness of the assignment that is submitted the most late.
  - `mileage`: The input is a list of pairs of values. Each ordered pair represents the fuel economy of a car (in miles per gallon) and distance it must be driven to deliver it, respectively. You should return the index of which car should be delivered first to minimize the total cost of gas after completing all deliveries.

Input Format

Input will be given in the console (same as with the previous PAs). To provide an input with n pairs, you will have $n + 2$ lines in your input as follows:

- The first line contains the name of the problem you're solving ("bakeoff", "deadlines", or "mileage")
- The second line should have just the number n (e.g. "53")
- The remaining n lines should have a pair of doubles/floats separated by a single space (e.g. "13.5 1.234")

One sample input is provided to you per problem. You must determine for yourself what the correct solution would be for each.

Hopefully Helpful Hints

To help you to get started with thinking about this problem, keep these in mind:

- Each input is a collection of pairs. Consider what happens when you have small values paired with small values and big values paired with big values. In each case, what was the best solution?
- If your greedy choice would say that two choices were equally good then it should be that it doesn't matter what order to put them in. For example, if for Deadlines your greedy choice says to do the assignment that takes the longest first (ignoring the deadline) then if your input was a list of several assignments that took the same amount of time (but had different deadlines), it should not actually matter which assignment is completed first.

– Initially, the starter code will always return the first thing in the list. This means that the schedule produced by the algorithm just places all items in the same order they were given. Use this to play around with each problem by re-ordering the items by hand and seeing what that does to the cost.

– This is probably the biggest hint in the list. Each greedy choice is going to involve finding the maximum/minimum value produce by some expression using the ordered pair. For example, it may just be "pick the thing with the smallest first item in the pair" or "pick the thing with the largest sum of items in the pair". If your greedy choice function is more than just a for loop that finds max/min then you're on the wrong path.

