Programming Assignment #3

Some Preliminaries

Programming assignments are to be done individually. Do not make your code publicly available (such as a Github repo) as this enables others to cheat and you will be held responsible. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet, or from any internet tools such as ChatGPT. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

The Goals

This programming assignment is due **Monday, November 27th at 11:59 PM.** If you are unable to complete the assignment by this time, you may submit the assignment late until Wednesday, November 29th at 11:59 PM for a 20-point penalty.

The goals of this lab are:

- Recognize and deploy algorithms you have learned in class in new contexts.
- Ensure that you understand certain aspects of dynamic programming.
- Recognize and reason about social implications of employing algorithms in a medical application.

The Problem: Design of Combination Chemotherapy Treatments

Combination Chemotherapy[1] is the use of two or more medications simultaneously to treat a disease such as cancer. Recently, data analytics and (you guessed it!) algorithms have been used to aid in the design of combination chemotherapy regimens[2]. In this programming assignment, we'll explore a simple approach to exploring the design space of combination chemotherapy using dynamic programming.

We will consider the situation where we have available to us n different medicines. Each medicine has a potential (non-negative) impact on the treatment plan. We assume for simplicity that these impacts are independent and additive, and our goal is to select a dosage of each medicine that maximizes the total impact. But there's a catch—we have a finite amount of time H (in hours) that we can use to deliver the medication. For each of the n medicines ($0 \leq i < n$), we have analytically derived a non-decreasing function $f_i(h)$ that gives a value of the impact of medicine i if delivered for h hours. The fact that the functions are non-decreasing is important—spending more time delivering a medicine will not *decrease* the overall impact, but it might not increase it. You can make no assumptions about the growth rates (or lack of growth rates) of these functions,

[1] `https://www.verywellhealth.com/what-is-combination-chemotherapy-2248995`
[2] `https://pubsonline.informs.org/doi/abs/10.1287/mnsc.2015.2363`

other than that they are non-decreasing.

Your goal is to determine a number of hours h_i to allocate to each medicine such that each medicine such that $0 \leq h_i \leq H$ and $\sum_{i=0}^{n-1}(h_i) \leq H$. Note the $\leq$ sign in the previous: if the benefit of all medicines maxes out at some $h < H$, you should stop delivering medicines. It won't *hurt*, from an impact on health perspective, to deliver more medicine, since the functions are non-decreasing, but it is undesirable from a treatment perspective. Note that medicines are assumed to be given for round number of hours, so h_i is always an integer.

We will provide you with the following classes:

- `ImpactCalculator`
  This class has already been implemented for you. It takes in a text file representing the medicines' impact functions and parses them for use in your program. Do not modify this class to make your solution work. We will be using our own version of `ImpactCalculator` during grading. Treat it as a black box to access the impact functions $\{f_0, f_1, ..., f_{n-1}\}$.

- `Program3`
  The class that you will be implementing your algorithms in. Contains an ImpactCalculator object provides parameters that define the problem. Also contains an int array treatment_plan, which must be updated to represent a correct treatment plan. *More explained below.*

- `Driver`
  Reads file input and populates `ArrayList regions`. You can modify `testRun()` for testing purposes, but we will use our own `Driver` to test your code.

** Part 1: Implement the Algorithm ** [60 points]

Implement an algorithm to find the optimal treatment plan, given the constraints defined above. You should create an algorithm with the fastest runtime possible, given what we have learned in class so far. The specific inputs provided and outputs expected are provided below:

Input(s):

- `ImpactCalculator calculator`
  In the Program3 class, we provide you an ImpactCalculator object. This object will provide all the necessary info to define the problem. you can call calculator.getNumMedicines() to determine the total number of medicines for the problem and calculator.getTotalTime() to determine the total length of time that medicine can be administered. Additionally, you can determine the impact of a medicine M_i for a specific length t_i by calling calculator.calculateImpact(M_i,t_i).

Output:

- `return totalImpact`
  ComputeImpact() should return the maximum amount of impact the set of medicines can have over the limited time defined by the ImpactCalculator. This will be evaluated in some test cases.

- `int[] treatment_plan`
  Treatment_plan is an int array in Program3.java that represents the treatment plan for the given medicines. Each index represents a medicine, and the value at the index is the length of time that that medicine is applied. ComputeImpact() should update the treatment_plan array in Program3.

**** Part 2: Write a report **** [40 points]
Write a short report that includes the following information:

(a) Solve the dynamic programming problem with the following steps: characterize the structure of the problem, write a recursive definition of the value of the optimal solution, write the pseudocode for filling in the table, and describe how you will find the optimal solution itself (i.e., how many hours to spend on each medicine).

(b) Give the runtime of your algorithm as a function of hours and medicines (H and M) and justify. Is your algorithm *polynomial time* by the definition given in class? Why or why not?

(c) Modern healthcare systems entail a *payer*, which is not always solely the patient themselves and usually instead involves an insurance company or the government. In such instances, the payer may have its own constraints on treatment, often based on allowable cost[3].

Consider the desire to add a cost constraint to the problem in addition to a time constraint. That is, for each medicine, we have a second function $c_i(h)$ that we must pay for each hourly dose of the medicine, and the payer has set a total budget of C.

How would such a change impact your solution? *You do not have to implement this change*, but you should consider how it might change your dynamic programming algorithm and describe potential updates, including updates to the runtime.

(d) Access to cancer treatments varies widely based on a variety of factors[4]. Find (via research) or brainstorm (based on research) one or more ideas on other ways algorithms we have learned so far could be employed to aid in analyzing or equalizing access to cancer treatments. Be sure to include your citations!

What To Submit
You should submit only `Program3.java` to Gradescope. Please do not submit any other Java file. Failure to follow these instructions will result in a penalty of up to 10 points.

NOTE: **DO NOT USE PACKAGE STATEMENTS.** Your code may fail to compile if you do.

Your PDF report should be legibly scanned and submitted to Gradescope. Both your code and PDF report must be submitted by 11:59 PM on November 27, 2023. You may submit to Gradescope as often as you want up to the deadline with no penalty. Your code will be evaluated on public and private/hidden test cases. In order to receive a score, your score must not timeout in Gradescope.

[3] `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6652174/`
[4] `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8120029/`