Homework 6: More Dynamic Programming

Version 3: We corrected typos in problem 3. (posted Thursday night)

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. **If you submit an answer substantially longer than 2 pages, the TAs are allowed to stop reading at the end of page 2.**

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 have a different box for each problem, please give yourself time to submit.

Note: This current version is slightly different from the version released on Tuesday, there is a typo correction in 3(b)

1. A Much More Efficient Itinerary [10 points]

Recall in Homework 2 Problem 3, you designed a recursive algorithm to find the number of different sequences of a activities satisfying a set of constraints. Our solution for that problem took $\mathcal{O}\left((a+r)^d\right)$ time (we strongly recommend looking at our solution if you did something different for this problem).

(a) In our proof, we said "Our function calculates, on input `ActivitySearch`(v, i) the number of activity lists of length i (i.e., lists with i activities) which end at Chess-boxing and start at v." That looks like an English description of a recurrence! Give a recurrence that will calculate this value (the recursive code in the solutions may also help). You'll need to make sure you're using the same graph description our solutions give.

(b) Now that we know DP, memoize it! What is your memoization structure?

(c) What's the filling order?

(d) Is it faster now?! What would the running time be (give 1-2 sentences of justification). WOW memoization makes a big difference!!

2. Programming: make a change and get the choices [25 points]

We've given you a (Java) function `LIS`, which returns the **length** of the longest increasing subsequence of an array. Your task is to write code for a *variant* of the longest increasing subsequence problem.

Specifically, given an array, you should return the longest **geometrically** increasing subsequence. A subsequence is **geometrically** increasing if for chosen elements where $i < j$, we have $3A[i] < A[j]$. I.e. each successive element must be more than three times larger than the previous. For simplicity, you may assume there are no indices i, j such that $3A[i] = A[j]$ and that there are no repeated elements.

Your code will return an ArrayList containing **in order** the **elements** of a longest geometrically increasing subsequence. You therefore will need to modify the code given both to handle the geometric requirement AND to build the subsequence itself. (You're also free to start from scratch, but we think you'll find the starter code helpful).

You will submit your code to gradescope, where it will be autograded. You may submit until your code is correct (your score will be the score of your last submission before the deadline).

3. Alchemis-Tree [25 points]

As an alchemist, you have a collection of bottles, each holding a specific type of mystic substance. You are planning to move to a different place and you found that it is difficult to carry all the bottles with you.

Due to their chemical properties, some of the substances, when put together, produce effects which would be...unfortunate. You have a note that tells which pairs of substances when mixed together, can cause damage, and you know the relationship can be modeled as a tree.

You have three large containers. Each is large enough to hold as many of the chemicals as you need. One of the three is new, though, and has not been extensively tested. You need to find a way to put all your substances into one of the three containers while minimizing the weight of the chemicals in the last (undertested) container.

Formally, given a tree T, you need to assign each vertex a target container "A", "B", or "C". Vertices cannot be assigned to the same container as its parent or any of its children. For each vertex, you can call $q(v)$ to get the mass of the chemical in that specific bottle. You want to minimize the mass in container "C" while not causing an unwanted chemical reaction (i.e., not putting adjacent nodes in the same container). You only need to return the final mass in container "C", (you do not need to record the assignment itself).

(a) Give pseudocode for an algorithm to run for this problem (Hint: You don't need DP here! We only needed one line). Give a 1-2 sentence explanation for why your algorithm is correct.

(b) Unfortunately, you later found that some substances can only be safely put into one of the containers. Luckily this is only happening to "leaf" nodes of the tree. You now have a tree where some leaves are pre-labeled with "A", "B" or "C", which is the **only** container that can safely hold them. All other nodes (internal nodes and unlabeled leaves) can got to any of the three containers.

We'll find a dynamic programming algorithm to solve the problem.

- Give an English description of what each of your recurrence(s) calculate and what each of the parameters means.
- Write a recurrence (or multiple recurrences) that describe the minimum mass in container C in a tree where you don't get to change any of the pre-labeled nodes.

  If there is no way to allocate all substances to containers (say one substance has children pre-labeled with all three possible labels), return ∞.

  Note: In a previous version of the assignment, we had a typo and asked to describe the minimum number of A labels in the tree; the recurrences should be similar, but if you've already solved the version with minimum number of A labels, make an explicit comment at the top of your submission and we will grade your problem for the original version.
- What will your final answer be (e.g. where in the memoization structure should we look)?

(c) What will your evaluation order be?

(d) What is the running time to evaluate your recurrence? Let n be the number of vertices in your tree. Give 1-3 sentences of justification.

4. Ro-bean-Hood [25 points]

After acquiring a large amount of beans from your local bean store, you decide to try your hand at day-trading beans at the local bean market. The market is especially rocky for bean-traders: the price of beans don't change during the day, but all of the prices can change significantly from one day to the next.

You plan to trade beans for n days and, due to insider knowledge[1] , you can predict the prices for the k different bean varieties for each of the next n days. Due to restrictions in the bean-trading sphere, you're only allowed to

[1]We, the 421 staff, do not condone insider-trading

hold at most a single bean during any day, and you're allowed to perform at most one action each day (either selling a bean, buying a bean, or not doing anything).

Determine the maximum amount of profit you can receive.

Formally, given k the number of distinct beans you can trade, n the number of days you trade, and k lists each of size n where the j^{th} term in the i^{th} list corresponds to the price of the i^{th} bean on the j^{th} day (this is a price that you can buy the bean at if you hold no other beans, or this is the price that you can sell the bean if you were previously holding this i^{th} bean). At any time you may hold at most one bean and on every day you may perform at most one action (buy, sell, or do nothing). Describe a dynamic programming algorithm to determine the maximum value you can get via some process of buying and selling the beans you have (you **must** use dynamic programming, even if other techniques also give correct algorithms).

(a) Clearly state **in English** what your recurrence(s) calculate. Be sure to mention any parameters you use.

(b) Write a recurrence (or multiple recurrences) that you will use to solve this problem.

(c) Give a brief explanation of your recurrence; we're not looking for a formal proof, but a brief explanation of what each of the cases represent, and why that set of cases is exhaustive.

(d) Describe a memoization structure.

(e) Describe a filling-order for your memoization structure. If you wrote (iterative) pseudocode to fill the structure, what would the running time be? Justify the running time with 1-2 sentences.

(f) What is your final answer going to be? (where in the memoization structure do we look?)