CS 161 (Stanford, Winter 2022) Homework 6

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 Pruning Trees

Suppose you are given a **binary** tree with n nodes, and each node has an associated weight, which is a positive integer. You would like to remove nodes from this tree such that the resulting tree has at most k nodes, and you would like to maximize the sum of the weights of the remaining nodes. Additionally, at the end of the pruning, you must still have a tree rooted at the original root; in other words, if you remove a node, then all of that node's children/descendants must also be removed.

1. **(3 pt.)** For any node u of the original tree, and any positive integer $i \leq k$, let $A[u, i]$ denote the maximum weight of any subtree rooted at node u having at most i nodes. Letting r_u and ℓ_u denote the right and left children of node u (they are NULL if u does not have that child), and letting w_u be the weight of node u, formally describe the optimal structure by giving a recurrence that expresses $A[u, i]$ in terms of the quantities $A[r_u, 1], A[r_u, 2], \ldots$ and $A[\ell_u, 1], A[\ell_u, 2], \ldots$. **[We are expecting:** An expression for the recurrence relation for $A[u, i]$. If you find it convenient, you may additionally define $A[u, 0]$ and/or $A[NULL, i]$ to be an appropriate value, and use them in the recurrence.**]**
2. **(3 pt.)** Using the recurrence relation from the previous problem, define a dynamic programming algorithm that will efficiently solve the problem. (The algorithm only has to return the weight of the best tree. It doesn't have to return that best tree.) What is the runtime of your algorithm? **[We are expecting:** A brief description or pseudocode of your algorithm, no justification of correctness required. Additionally, the runtime and a brief description of how you arrived at your answer.**]**

2 Longest Increasing Subsequence

Let A be an array of length n containing real numbers. A *longest increasing subsequence* (LIS) of A is a sequence $0 \leq i_1 < i_2 < \ldots i_\ell < n$ so that $A[i_1] < A[i_2] < \cdots < A[i_\ell]$, so that ℓ is as large as possible. For example, if $A = [6, 3, 2, 5, 6, 4, 8]$, then a LIS is $i_0 = 1, i_1 = 3, i_2 = 4, i_3 = 6$ corresponding to the subsequence $3, 5, 6, 8$. (Notice that a longest increasing subsequence doesn't need to be unique). In the following parts, we'll walk through the recipe that we saw in class for coming up with DP algorithms to develop an $O(n^2)$-time algorithm for finding an LIS.

1. **(1 pt.) (Identify optimal sub-structure and a recursive relationship).** We'll come up with the sub-problems and recursive relationship for you, although you will have to justify it. Let $D[i]$ be the length of the longest increasing subsequence of $[A[0], \ldots, A[i]]$ that ends on $A[i]$. Explain why

$$D[i] = \max\left(\{D[k] + 1 \,:\, 0 \leq k < i, A[k] < A[i]\} \cup \{1\}\right).$$

   [We are expecting: A short informal explanation.**]**

2. **(2 pt.) (Develop a DP algorithm to find the value of the optimal solution)** Use the relationship about to design a dynamic programming algorithm returns the *length* of the longest increasing subsequence. Your algorithm should run in time $O(n^2)$ and should fill in the array D defined above. **[We are expecting:** Pseudocode. No justification is required.**]**

3. **(2 pt.) (Adapt your DP algorithm to return the optimal solution)** Adapt your algorithm above to return an actual LIS instead of its length. Your algorithm should run in time $O(n^2)$. **[We are expecting:** Pseudocode **AND** a short English explanation of what your algorithm is doing. You do not need to justify that it is correct.**]**

Note: Actually, there is an $O(n\log(n))$-time algorithm to find an LIS, which is faster than the DP solution in this exercise!

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.

---

3 Housing Layout

You own $n \geq 1$ consecutive plots of lands that you can build on, and you want to build a number of houses on these plots, with each plot having at most one house. However, due to some strange laws in your city, you cannot build two houses on two consecutive plots of lands. (The other plots of lands will just be wasted, unused.) Each plot of land is different, so building the house in the right plots is important. You have estimated the profit you would get from building a house on each plot of land to be $p[1], \ldots, p[n]$, where $p[i]$ is a positive integer representing the profit, in dollars, you would get from building a house on the i^{th} plot of land.

Example if the input was $p = [21, 4, 6, 20, 2, 5]$, then you should build houses in the pattern

and you would profit by $21 + 20 + 5 = 46$ dollars. You would **not** be allowed to build houses in the pattern

because there are two houses next to each other. In this question, you will design a dynamic programming algorithm which runs in time $O(n)$ which takes as input the array p and returns the maximum profit possible given p. Do this by answering the two parts below.

3.1 Sub-problems (6 pt.)

What sub-problems will you use in your dynamic programming algorithm? What is the recursive relationship which is satisfied between the sub-problems? What are the base cases for this recursive relationship?
[We are expecting: A clear description of your sub-problems, a recursive relationship that they satisfy, along with a base case, and an informal justification that the recursive relationship is correct.**]**

3.2 The algorithm (6 pt.)

Write pseudocode for your algorithm. Your algorithm should take as input the array p, and return a single number which is the maximum profit possible. Your algorithm does not need to output the optimal way to build houses.
[We are expecting: Pseudocode **AND** a clear English description. You do not need to

justify that your algorithm is correct, but correctness should follow from your reasoning in the previous part.**]**

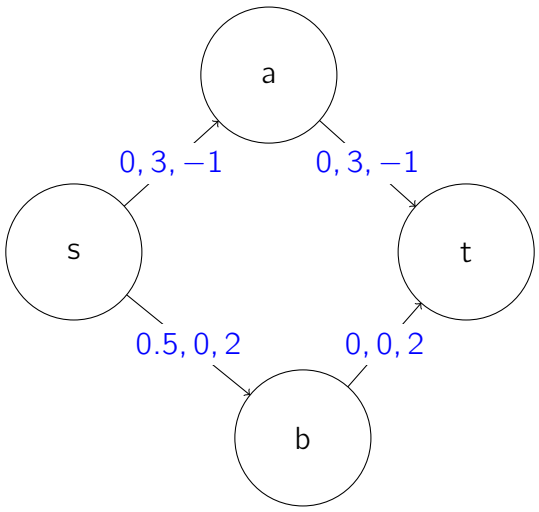
4 Most Reliable Path II

(Plot continued from HW5 Q4) After the trip from town s to city t, you decided to open a tourism company and provide the same trip plan to tourists. There is no direct flight from s to t so you need to take multiple flights to reach t, and the probability of cancellation for each flight varies from month to month. You created a directed graph $G(V, E)$ where each node is an airport and each edge is a flight from one airport to another. You have a map with *reliability scores* calculated based on history of cancellation and time efficiency for the flights each month (can be either positive or negative or zero), and you want to maximize the total reliability score along the trip. However, your customers expect to visit exactly the same stops as they see on social media posts from you, so they don't like changes. The reliability score of a route R is denoted $S_i(R)$, and is the total reliability score of flights in the route corresponding to the i-th month. $Different(R_1, \ldots, R_k)$ is the number of months $i > 1$ such that $R_i \neq R_{i-1}$. The cumulative tourist satisfaction from a sequence of routes is given by:

$$Satisfaction(R_1, \ldots, R_k) := \left(\sum_{i=1}^{k} S_i(R_i)\right) - Different(R_1, \ldots, R_k).$$

Assumption $S_i(R)$ is finite for every i and any route R from s to t (i.e. no positive cycles).

Example



In this example, the tuples represent reliability scores of the flights over months. The optimal tour is to use route $(s \to a \to t)$ the first two trips, and then $(s \to b \to t)$ in the third trip. The total satisfaction is:

$$(0+0)+(3+3)+(2+2)-1=9.$$

Notice that $(s \to b \to t)$ would have a higher score (0.5) for the first trip, but it wouldn't offset the cost (1) of adding another switch.

1. **(4 pt.)** If all routes had to be exactly the same, describe an algorithm that would maximize only the first term, namely $\sum_{i=1}^{k} S_i(R_i)$. **[We are expecting:** A short English description**]**

2. **(8 pt.)** Design a $O(k^2mn)$ dynamic programming algorithm to find the routes that maximize the satisfaction score, where m is the number of flights and n is the number of airports.
   [We are expecting: A short English description, pseudocode, and running time analysis**]**

5 Taking Stock

Suppose you are given reliable insider information about the prices for k different stocks over the next n days. That is, for each day $i \in [1, n]$ and each stock $j \in [1, k]$, you're given $p_{i,j}$, the price at which stock j trades at on the ith day. You start with a budget of P dollars, and on each day, you may purchase or sell as many shares of each type of stock as you want, as long as you can afford them. (Assume that all prices are integer-valued and that you can only purchase whole stocks.) You have an earning goal of Q dollars. Here, we will design an algorithm to determine whether you can meet your goal, and if not, how much money you can earn.

1. **(8 pt.)** Suppose we are only looking at prices over two days (i.e. $n = 2$). Design an $O(kP)$ dynamic programming algorithm that computes the amount of money you can make buying stocks on the first day and selling stocks on the second day. Prove the runtime and correctness of your algorithm.
   (*Hint: Let $M[l]$ be the most amount of money you can make by buying l dollars of stock on the first day. Write a recursive relationship for $M[l]$.*)
   [We are expecting: Pseudocode or a clear English description of the algorithm, a runtime analysis, and a rigorous proof of correctness**]**

2. **(8 pt.)** Now, suppose you are given prices over n days. Using your solution to part (a) as a guide, design an $O(nkQ)$ time algorithm that determines whether you can reach your goal, and if not, reports how much money you can make. Prove your algorithm's runtime and correctness.
   (*Hint: ItâĂŹs helpful to reframe each day as 1) selling all the shares you own and 2)*

then buying a set of shares that you can afford.)
[We are expecting: Pseudocode or a clear English description of the algorithm, a runtime analysis, and a rigorous proof of correctness**]**