COMP 285 (NC A&T, Spr '22) Homework 8

Due. Monday, April 11th, 2022 @ 11:59 PM!

Homework Expectations: Please see Homework.

Exercises The following questions are exercises. We encourage you to work with a group and discuss solutions to make sure you understand the material.
Points This assignment is graded out of 50 points. However, you can get up to 60 points if you complete everything. These are not bonus points, but rather points to help make-up any parts you miss.

Fun with Dynamic Programming

Written Problems The following questions are to be submitted in written/typed form to gradescope.

1 Longest Increasing Subsequence (10 pt.)

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. **(2 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.**]**

[1] If you're having trouble with this problem, take a look at Lecture 26 where we cover the Longest Common Subsequence problem. This problem is **not the same**, but is similar.

2. **(4 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. **(4 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! This algorithm uses divide-and-conquer! While not required for this exercise, you might find it helpful to think about this problem and see if you can come up with a solution!

2 Getting Greedy! (5 pt.)

Sometimes it can be tricky to tell when a greedy algorithm applies. Say whether or not the greedy solution would work. If it wouldn't, give a counter example.

You have unlimited objects of different sizes, and you want to completely fill a bag with as few objects as possible.

Proposed Greedy Solution: Keep putting in the largest object possible given the space you have left.

[We are expecting: Either 'WORKS' or 'DOES NOT WORK'. If it does not work, an example of where it doesn't work should be given as well.**]**

3 Housing Layout (20 pt.)

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.[2]

3.1 Step 1 - Optimal Substructure (4 pt.)

What sub-problems will you use in your dynamic programming algorithm? To get you started, let use define $P[i]$ (an array of length n) which will start will all `NULL` values. What does $P[i]$ represent?

[We are expecting: A clear **English** description of the meaning of $P[i]$**]**.

3.2 Step 2 - Recursive Relation (4 pt.)

What is the recursive relationship which is satisfied between the sub-problems? That is to say, how can we compute $P[i]$ using answers to "smaller" subproblems (depending on how you define $P[i]$, this expression will either involve $P[i-1], P[i-2]$ or $P[i+1], P[i+2]$).

[We are expecting: A mathematical expression for how to compute $P[i]$.**]**

[2] If you're having trouble with this question, revisit your 'pickBerries' problem from HW2. It's actually quite similar, but now you will implement it so that its efficient!

3.3 Step 2 - Base Cases! (2 pt.)

What are the base cases for this recursive relationship you defined above?

[We are expecting: The base cases for your recursion.**]**

3.4 Step 3 - The algorithm (5 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.**]**

3.5 Step 4 - Where do I build? (3 pt.)

Suppose we want to know not only the maximum profit possible, but I also want to know **which** houses to build. How would you modify your solution above to answer this question?

[We are expecting: A short description of what changes you'd make to your algorithm to track the houses we need to build**]**.

3.6 Step 5 - Saving Space (2 pt.)

What is the space-complexity of your proposal above? If it is not $O(1)$, how would you improve your algorithm to use less space.

[We are expecting: A solution above that uses $O(1)$ space, or a short description of how you'd modify the algorithm to use only $O(1)$ space.**]**

Coding Problems The following questions are to be submitted as a ".zip" file on Gradescope.

4 Coding (25 pt.)

After completing the written portion of the assignment, you should submit it to Gradescope.

For the coding portion, get your starter C++ code or Python code.

Note that the starter code also include a few test cases you can run on repl.it. However, the full test suite is the one run on Gradescope.

Please reference the `README.md` included in your starter code for detailed instructions.

Submitting the Assignment

This assignment is a combination of written and programming questions. Both portions of the assignment should be submitted through Gradescope.

The "Homework 8: Fun with Dynamic Programming" assignment is the written portion, for which you should submit a **typed** response to the non-coding questions (questions 1-3). Each response should clearly be marked with its corresponding number. You are free to use the provided templates, print the questions and write your answers, or to simply type your responses on a blank document (whatever works for you).

The "Homework 8: Coding" is the programming portion of the assignment. For this portion, download the ".zip" file from replit and upload this ".zip" file as your answer to Gradescope. You can upload the assignment as many times as you want.