For dynamic programming problems that are submitted for credit in this class, but are not part of programming assignments, you must do the following; missing one of these can cost you significant credit. I recommend practicing the problems as if these are going to be submitted for credit.

- Give a clear and precise English definition that describes the function you are implementing. Not how it works (yet), but rather what it solves.
- Give that function a meaningful variable name. This is not [just] me being pedantic; I have found it helps students with this topic if they do this. “OPT” is not a meaningful variable name, nor is “table.” Single letters are not meaningful variable names. Yes, I know you’re going to read question one first, and yes, I read it too.
- Give a clear recursive formula or algorithm in terms of smaller instances of exactly the same problem. If you aren’t solving exactly the same problem, you might need to go back to the previous step.
- Describe the iterative running time correctly. This can either be by writing the iterative algorithm (in which case, you can point out where the previous part is within the solution), or by taking your recursive solution, counting the cases, describing the order in which the table would be filled in, and analyzing the time accordingly.

The questions on this problem set are from a mixture of sources; some are warm-up questions to the undergraduate algorithms course. Others come from past homework and exams in past instances of CompSci 260P. These are not strictly in increasing order of difficulty, but the first few are deliberately chosen to be good starting points.

1. Suppose I am going to choose an integer between 1 and \( n \), inclusive, according to some probability distribution. For each integer \( i \), I have written \( p_i \), the probability that I select \( i \) as the chosen integer. You may assume that \( \sum_{i=1}^{n} p_i = 1 \).

   (a) Give an \( \mathcal{O}(n^3) \) time algorithm to compute a 2D-array \( X \), where \( X[i, j] \) is the probability that some integer in the range \([i, j]\) (inclusive) is chosen. You may assume that arithmetic operations take \( \mathcal{O}(1) \) time each.

   (b) Give an \( \mathcal{O}(n^2) \) time algorithm to solve the problem in part (a). If you are confident that your answer to this question is \( \mathcal{O}(n^2) \), you may elect to skip the previous part and count this as your answer to both.

2. Professor Shindler gives lots of homework assignments, each of which have an easy version and a hard version. Each student is allowed, for each homework, to submit either their answer to the easy version (and get \( e_i > 0 \) points) or the hard version (and get \( h_i > 0 \) points, which is also guaranteed to always be more than \( e_i \)) or to submit neither (and get 0 points for the assignment). Note that \( e_i \) might have different values for each \( i \), as might \( h_i \). The values for all \( n \) assignments are known at the start of the quarter.

   The catch is that the hard version is, perhaps not surprisingly, more difficult than the easy version. In order for you to do the hard version, you must have not done the immediate

---

\(^1\)This is not the actual policy.

© CompSci 260P Spring 2023– Michael Shindler– University of California, Irvine
This problem set may not be reposted without the express written permission of the professor teaching this course.
previous assignment at all: neither the easy nor the hard version (and thus are more relaxed, de-stressed, etc). You are allowed to do the hard version of assignment one if you want. Your goal is to maximize the number of points you get from homework assignments over the course of the quarter. Give an efficient dynamic programming algorithm to determine the largest number of points possible for a given quarter’s homework choices.

3. *Skittles* are small candies; each piece of candy is one of five flavors (grape, lemon, orange, strawberry, or green). An exciting new party game, *Skittle Trader*, is sweeping the nation, and the skills you are learning in CompSci 260P will help you to achieve a high score!

Here are the rules of the game. You are at a party with \( n \) of your friends. Each of you has one piece of *Skittles* candy. Your friends form a line. You walk down the line, starting at friend 1 and finishing at friend \( n \). At each friend, you have a choice: do you want to trade the candy you are holding for the one they are holding?

- If you do not trade candy, your score remains the same.
- If you and the friend have the same flavor candy, you earn one point.
- If you and the friend trade, but you had different flavor types, you lose one point. Your score can go negative.

You begin the game with zero points. Your goal is to finish the game with the most possible points.

For purposes of the input to this problem, the Skittles flavors are numbers 0, 1, 2, 3, 4. Your input to this problem is an array \( C[1 \ldots n] \), where each \( C[i] \in [0, 4] \) is the type of candy the \( i \)th friend in line is holding. You also have a parameter for the type of candy you begin the game with.

If you decide to actually play this game with friends, perhaps to test out your algorithm before submitting, please use individually wrapped candies instead, such as Starburst. This is nearly the most disgusting question I have given in a class\(^2\).

4. College students get a lot of free food at various events. Suppose you have a schedule of the next \( n \) days marked with those days when you get a free dinner, and those days on which you must acquire dinner on your own. On any given day you can buy dinner at the cafeteria for $6. Alternatively, you can purchase one week’s groceries for $20, which will provide dinner for each day that week. However, because you don’t have a fridge, the groceries will go bad after seven days (including the day of purchase) and any leftovers must be discarded. Due to your very busy schedule, these are your only three options for dinner each night.

Write a dynamic programming algorithm to determine, given the schedule of free meals, the minimum amount of money you must spend to make sure you have dinner each night. For full credit, the iterative version must have running time polynomial in \( n \).

*Hint:* Start by writing a recursive procedure to determine how you should eat dinner on the last night.

\(^2\)The undergraduate data structures class has a popular practice problem that involves getting \( n \geq 44 \) pieces of fast food chicken, which is arguably more disgusting.
5. You’ve decided to play some music late night at your local club and want to impress. To
prepare your strategy, you have reviewed your notes and used statistical analysis to figure
out which moves get the most applause ending at certain musical measures. In your findings
you’ve found that at each measure, you can perform one of three different moves.

- **Jazzy riff:** *This is short and improvised.*
  How long it lasts is constant, but how you play depends on the ending measure.
  Played for 1 measure until the end of measure $t$, it awards you $j_t$ applause. Note that $j$
  is a vector: the measure on which you end the riff affects the applause earned.

- **Instrumental chorus:** *Your bread and butter.*
  Both how long it lasts and how you play is constant.
  Played for 2 measures until the end of measure $t$, it awards you $C$ applause. Note that
  $C$ is a constant: an instrumental chorus always earned you $C$ applause.

- **Solo:** *You play your heart out!*
  Both how long it lasts and how you play depends on the ending measure.
  Played for $k_t$ measures until the end of measure $t$, it awards you $s_t$ applause. Note that
  $k$ and $s$ are vectors: the measure on which you end a solo affects the applause earned.

Given these statistical models (the vectors $j, k, s$ and the constant $C$) are correct, you want
to figure out the **maximum amount of applause** you can obtain in $n$ musical measures.
*Solutions that omit the inclusion of the “solo” option but are otherwise correct will receive
reasonable partial credit.*

6. As a new professor, Shindler needs to set up his lab. However, nothing is free, and Shindler
has to pay quarterly rent for the building where he keeps his office. Shindler plans to be at
UCI for $n$ quarters (for a hopefully large value of $n$). Each quarter, he has the option to keep
his lab in Donald Bren Hall or in Engineering Hall. If he keeps his lab in Donald Bren Hall
during quarter $i$, he has to pay $d_i$. If he keeps his lab in Engineering Hall that quarter, he
has to pay $e_i$. If he is in one building during quarter $i$ and the other during quarter $i + 1$,
he has to pay $M$ to move offices.

For example, suppose $n = 4$ and $M = 10$, with the following costs:

<table>
<thead>
<tr>
<th></th>
<th>Quarter 1</th>
<th>Quarter 2</th>
<th>Quarter 3</th>
<th>Quarter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bren</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Engineering</td>
<td>50</td>
<td>20</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

In this case, the optimal solution would be to stay in Bren Hall for the first two quarters and
then move to Engineering Hall for the next two quarters.
7. Planning lectures can be quite the challenge. Luckily, I have already made a few key decisions already.

- Use at most $M$ minutes per lecture.
- Teach all $n$ topics and teach them in order (Topic 1 must be taught before Topic 3).
- Allocate topic $i$ exactly $t_i$ minutes.
- Topics cannot run over into another lecture (It is guaranteed that $\forall i \ t_i \leq M$).
- I can have as many lectures as I want.

If I was trying to minimize the number of lectures, I could just try to cover as many topics as possible per lecture until I finish them. However, my goal is to provide the best education experience and I have noticed that some topics work together better than others. According to my findings, I have found students receive $E_{ij}$ engagement when I cover topics $i$ through $j$ in the same lecture given that $i \leq j$.

If you would like to assume that $E_{ij} = -\infty$ whenever topics $i \ldots j$ do not fit into a single lecture spot, you may.

Given the topic time list $t$ and the engagement matrix $E$, design a dynamic programming solution to find the maximum total engagement I can achieve by teaching all $n$ topics in order without exceeding $M$ minutes per lecture.

8. Suppose you are helping to expand the popular chain of Algorithmic Pizza Parlor stores, who are now going to try to gain new customers by advertising. We have a budget of $B$ dollars to spend on advertising, and $n$ possible ways to spend it (billboards, television spots, radio ads, etc). For each possible spending method $i$, we can allocate anywhere between 0 and $B$ dollars, but must do so in even increments of one dollar (we cannot allocate partial dollars). Our marketing department has come up with a series of functions $\{f_i\}$ to determine how effective (in terms of customers gained) each advertising method will be. If we spend $d$ dollars on advertising method $i$, we will gain $f_i(d)$ customers. These functions are all non-decreasing: if $d < d'$, then $f_i(d) \leq f_i(d')$. You may assume that $f_i(0) = 0$ for all $i$, if you would like.

Given these functions, use dynamic programming to design an algorithm that will determine the maximum number of new customers we can gain within our budget.

Hint: The correct solution is not to figure out where to put the last marginal dollar.

9. In the MIN-COST FAST PATH problem, we are given a graph $G = (V,E)$ along with non-negative integer times $t(e)$ and non-negative costs $c(e)$ on each edge. Our goal is to find the minimum cost of a path $P$ from $s$ to $t$ such that $\sum_{e \in P} t(e) \leq C$. Give a dynamic programming algorithm to determine the minimum cost of such a path. Ideally, your algorithm’s running time should be $O(mC)$.

In addition, I recommend the following problems from the course textbooks. If you need help deciding which to try, I encourage you to try R-12.7, R-12.8, C-12.1, C-12.9, C-12.16, A-12.1, A-12.2, A-12.3, A-12.4, A-12.6, A-12.10 in the textbook of Goodrich and Tamassia or Chapter 3, problems 1, 2, 3, 6, 9, 11, 12, 13, 16, 17, 18, 23, 32, 33, 35, 46, or 49 in the textbook of Erickson. For a big challenge, try question 28 in that chapter.