Below are three dynamic programming problems. You might recognize one or more. Before Friday's 6:00 PM lab, I encourage you to try each of these, either alone or with friends in the class. We will present solutions to these in lab, along with tips for how to solve problems like them.

It is possible you have seen one or more problems like these, or even possibly these questions themselves, in your undergraduate algorithms course. The goal is *not* to have an answer memorized – it's far more important if you know how to come up with one, on your own, if presented with one you haven't seen before.

We are moving onto more complicated dynamic programming uses in weeks 3 and 4, so now is a great time to be sure you understand the fundamentals!

1. Suppose a hiker is about to go on a trek through a rainforest carrying a single knapsack. She knows the maximum total weight $W$ she can carry and has a set $S$ of $n$ potentially useful items to take with her, such as a folding chair, a tent, or a copy of the CompSci 260P textbook. Each item $i$ has a weight $w_i$ (a positive integer) and a benefit value $b_i$ (positive, not necessarily integer). The goal is to select a subset of the items such that the total weight is at most $W$ while maximizing the total benefit of the items. This is known as the **0-1 Knapsack Problem**.

   Give a dynamic programming algorithm to compute the largest subset of items she can carry with her, subject to the total weight constraint.

2. Professor Shindler gives lots of homework assignments, each of which have an easy version and a hard version[1]. 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 semester.

   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 previous assignment at all: neither the easy nor the hard version (and thus are more relaxed, de-stressed, etc). Your goal is to maximize the number of points you get from homework assignments over the course of the semester. Give an efficient dynamic programming algorithm to determine the largest number of points possible for a given semester's homework choices.

3. Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee $j$ has a value $v_j$ (a positive integer), representing how enjoyable their presence would be at the party. Produce an algorithm that will determine which employees should be invited, subject to the above constraints.

[1] This is not the actual policy.