Lab 9 (DP)

Collaboration level 0 (no restrictions). Open notes.

---

1. **Pharmacist problem:** A pharmacist has W pills and n empty bottles. Bottle i can hold p_i pills and has an associated cost c_i. Given W, $\{p_1, p_2, ..., p_n\}$ and $\{c_1, c_2, ..., c_n\}$, you want to store all pills using a set of bottles in such a way that the total cost of the bottles is minimized. Note: If you use a bottle you have to pay for its cost no matter if you fill it to capacity or not.

   Find the minimum cost for storing the W pills using the bottles.

   (a) Explain how the problem has optimal substructure.

   Answer: Consider an optimal solution O, and consider one of the bottles in it. Let's say this is bottle k, and it holds p_k pills. Then we know that the remaining bottles in O must be the optimal way to store .....

   (b) Define a subproblem and give pseudocode for a recursive function to compute it.

   (c) Extend your recursive pseudocode above to a recursive dynamic programming algorithm with memoization and analyze its running time.

2. **Unbounded knapsack:** We have n items, each with a value and a positive weight as given by two arrays: Item i has value $v[i]$ and weight $w[i]$. We have a knapsack that holds maximum weight W and we have *infinite copies* of each item.

   Given $W, \{w_1, ..., w_n\}$ and $\{v_1, ..., v_n\}$, we want to find the maximal value that can be loaded into the knapsack.

   (a) Suppose the optimal solution contains item i. Then the remaining part of the optimal solution must be an optimal solution for .... .

   (b) Come up with a recursive definition. Define your sub-problem and state clearly what its argument(s) represent and what it returns.

   Hint: we have infinite supplies of each item. So we don't need to keep track of which items we already included, only of the remaining space in the knapsack. Consider all the choices, and pick the best.

   (c) Develop a DP solution — either top-down or bottom up. What is the running time of your algorithm ?

(d) A friend proposes the following algorithm: start with the item with the largest cost-per-pound, and pick as many copies as fit in the backpack. Repeat.

Show your friend this is not correct by coming up with a small example where this algorithm leads to a solution that is not optimal.