Sample DP & Greedy Solutions

Initial Version

Dynamic Programming: Essential Components

When writing out your DP solution, check the following:

1. Define the **high-level meaning** of states. For the dp function (e.g. $dp(i, j)$), explain the meanings of the parameters i and j, as well as what $dp(i, j)$ represents.
2. Define the recurrence relation mathematically. Supplement it by a simple textual explanation.
   (Note 1: It is a good practice to state **clearly the bounds**. Under what constraints can you apply a certain transition?)
   (Note 2: Do not mix up recurrence relations with recurrence formulas. We won't ask for recurrence formulas, e.g. $T(n) = T(n/2) + 1$, in DP questions.)
3. State the base cases, e.g. $dp(0, j) = 0$ for all j.
4. Give your algorithm, either top-down with memoization or bottom-up. If you use bottom-up DP, what are the states considered and what is the computation order?
5. If required: Give a proof of correctness by proving the **optimal substructure property** (you can phrase it using your recurrence relation).
6. Give a time complexity analysis by calculating the number of **distinct** states considered and the time needed to compute each state.

Note that writing pseudo-code does not exempt you from the items above. My recommendation is not to write a pseudo-code so that you must hit the above items properly.

Dynamic Programming Sample: Knapsack

For $n \in \{0, 1, \ldots, N\}$ and $w \in \{0, 1, \ldots, W\}$, denote $dp(n, w)$ as the maximum total value of items we can get among the first n items, without exceeding a total weight of w.

When we consider $dp(n, w)$, there are two choices: either we take the n-th item or we don't. If we include the n-th item, then we can include the optimal solution for the first $(n-1)$ items with a weight limit of $w - w_n$, with extra value v_n from the n-th item. If we do not include the n-th item, then we use the optimal solution for the first $(n-1)$ items. It follows that

$$dp(n, w) = \max \begin{cases} dp(n-1, w) \\ dp(n-1, w-w_n) + v_n \quad \text{if } w \geq w_n \end{cases}$$

When $n = 0$, then we have $dp(n, w) = 0$ for all w, since there are no items considered so far.

We can compute $dp(n, w)$ using a bottom-up approach, filling in the table in increasing order of n followed by increasing order of w. Since $dp(n, *)$ only depends on $dp(n-1, *)$, we will have the necessary values to compute $dp(n, w)$. The final answer would be $dp(N, W)$.

There are clearly $(N+1) \cdot (W+1) \in O(NW)$ states and computing each state takes $O(1)$ time. Hence, the total running time is $O(NW)$.

Greedy: Essential Components

1. State the **optimal substructure property**:
   - Consider any optimal solution OPT and let X be one item selected in the optimal solution. If we remove X and ..., OPT $\setminus \{X, \ldots\}$ must be an optimal solution to the subproblem.

   Prove it using a **cut-and-paste argument**:
   (a) Assume the contradiction: The solution to the subproblem is sub-optimal.
   (b) Cut and paste the solution to the subproblem to the whole problem.
   (c) Conclude that the solution to the whole problem is also sub-optimal.
2. State the **greedy-choice property**:
   - **There exists** an optimal solution with the item that ...

   Prove it using a **exchange argument**:
   (a) Take any optimal solution OPT.
   (b) If it contains the item specified in our greedy choice, we're done.
   (c) Otherwise, "exchange" it a solution that contains the item specified in our greedy choice, such that the solution is **not worse** than the original one.
3. State your algorithm (if required).
   - It must pick a greedy choice according to the greedy-choice property, then reduce to the subproblem following the optimal substructure.
   - Some optimizations (sorting) might be required.

Greedy Sample: Fractional Knapsack (given in lecture)

Optimal substructure: If we remove y kgs of one item j from the optimal knapsack, then the remaining load must be the optimal knapsack weighing at most $W - y$ kgs that one can take from the $n - 1$ original items and $w_j - y$ kgs of item j.
Proof.
1. Let X be the value of an optimal knapsack.
2. Suppose that the remaining load after removing y kgs of item j was not the optimal knapsack weighing at most $W - y$ kgs that one can take from the $n - 1$ original items and $w_j - y$ kgs of item j.
3. This means that there is a(nother) knapsack of value $> X - v_j \cdot \frac{y}{w_j}$ with weight $\leq W - y$ kgs, among the $n - 1$ original items and $w_j - y$ kgs of item j.
4. Combining with y kgs of item j gives knapsack of value $> X$ and weight at most W for original input, a contradiction.

Greedy-choice property: Let j^* be the item with the maximum value/kg, v_j/w_j. Then, there exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^*.
Proof.
1. Take any optimal solution OPT.
2. If it already contains $\min(w_{j^*}, W)$ kgs of item j^*, we are done.
3. Otherwise, if it contains $w_{\texttt{OPT}} < \min(w_{j^*}, W)$ kg of items, we can add $\min(w_{j^*}, W) - w_{\texttt{OPT}}$ kgs of item j^* and the value strictly increases, which is a contradiction.
4. Therefore, the optimal solution contains at least $\min(w_{j^*}, W)$ kgs of items. Suppose it contains x_1 kgs of item 1, x_2 kgs of item 2, ,x_n kgs of item n such that:
$$x_1 + x_2 + \cdots + x_n = \min(w_{j^*}, W)$$
5. Replace this weight by $\min(w_{j^*}, W)$ kgs of item j. Total weight does not change, and total value does not decrease because value/kg of is maximum. Hence this solution is equally optimal.

> **Warning:** There could be **multiple** items with the maximum value/kg ratio. We must prove that there exists an optimal solution containing the item for **each of** such items. Otherwise, our greedy algorithm would not be able to confidently pick any of them.
> The implication is that even if OPT includes another item with an equal value/kg ratio, we should still exchange it with our item to show that we **can** pick our chosen item in **some** optimal solution.