**Intermediate DP problems**

1. Dynamic programming is an art, and the only way to get better at it is practice. Up until this point, establishing a problem's recurrence relation has been tricky but somewhat intuitive. For higher-level DP problems, breaking the problem into sub-problems and establishing the recurrence relation is no small task. These are the basic steps for solving a DP problem:
   1. Define sub-problems
   2. Write down the recurrence that relates the sub-problems
   3. Recognize and solve the base cases
2. The unbounded knapsack problem (similar to the Thievery project you memoized): Given a knapsack weight **W** and a set of **n** items with certain value vali and weight wti, calculate the minimum amount that could make up this quantity exactly. This is different from "classical" Knapsack problem; here we are allowed to use unlimited number of instances of an item. Examples:

Input : W = 100

val[] = {1, 30}

wt[] = {1, 50}

Output : 100

There are many ways to fill knapsack.

1) 2 instances of 50 unit weight item.

2) 100 instances of 1 unit weight item.

3) 1 instance of 50 unit weight item and 50

instances of 1 unit weight items.

We get maximum value with option 2.

Input : W = 8

val[] = {10, 40, 50, 70}

wt[] = {1, 3, 4, 5}

Output : 110

We get maximum value with one unit of

weight 5 and one unit of weight 3.

1. The [UVa online judge](https://uva.onlinejudge.org/) site has *almost* *everything* an aspiring competitive programmer could ever want to learn.
2. Try [these](https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=648) intermediate dynamic programming problems (some of them you've already done). If the link ever breaks, you can get there by:  
   1. Going to the main site.
   2. Clicking the **"Browse problems"** link on the left.
   3. Click the **"Competitive programming 3: The New Lower Bound..."** link / folder.
   4. Click **"Problem solving paradigms"**.
   5. Click **"Dynamic programming"**.