Dynamic Programming: Take It or Leave It Paradigm

Many problems with dynamic programming solutions can be characterized as "take it or leave it" problems. Consider the two following problems:

Donation Collector

Imagine walking door to door to get donations. Unfortunately, if one neighbor gives to a cause, the neighbor next to him will not want to. The problem is as follows: given a list of how much each person on a street, in order, is willing to donate, what is the maximum amount of donations you can garner?

For example, if the street had six houses and the numbers below represented how much each neighbor would be willing to give,

10, 8, 2, 6, 7, 9

Then the most that could be collected is 25 dollars, from houses 0, 3 and 5 (highlighted).

Let the input array be array. We can create a second array, dp, where dp[i] stores the most we can collect from the subarray of houses 0 through house i.

If we want to solve for dp[i], we have two choices for what to do with the donation from house i:

1) Take this donation.

2) Leave this donation.

If we do the former, the most money we could garner is array[i] + dp[i-2]. This is because if we take money from house i, we CAN'T take money from house i-1, so we just want to build off of the best solution for the first i-2 houses.

If we do the latter, the most money we could garner is dp[i-1].

Thus, our solution looks like this (after array is filled and dp is allocated, and n is the size of the arrays):

```
dp[0] = array[0];
dp[1] = array[1];
for (int i=2; i<n; i++)
     dp[i] = max(array[i]+dp[i-2], dp[i-1]);
```

where the first option represents how much money we get for "taking item i" and the second option represents how much money we get for "leaving item i".

Desert Thirst

In this problem, you are walking through a desert with the goal of getting out of the desert. You start with an initial hydration level of H. Each minute you walk, you lose 1 unit of hydration. On your walk out of the desert, there are various stands set up, selling water. Stand i is t_i minutes of walking time away from your starting point, and sells you x_i units of hydration for d_i dollars. It's imperative that you never go below 0 hydration. (Assume that these hydration stations are listed in order of distance from your starting poing.) Luckily though, as soon as you buy water, you can drink it instantly, and it does not slow down your walk out of the desert. (Also, you can make the purchase instantly!) The goal is to determine the least cost necessary to get you out of the desert.

Define f(a, h) = the minimum cost to get you out of the desert from location a (a minutes from your starting point) with a hydration level of h.

Our goal will be to find f(0, H). (We assume there is no water stand at location 0.)

To solve for $f(t_i, h)$ in general, we consider moving towards the next hydration station located after position x_i. There are two options:

(1) Take the water at station i+1.

(2) Leave the water at station i+1.

In the first instance, our cost would be $d_{i+1} + f(t_{i+1}, h - (t_{i+1} - t_i) + x_{i+1})$.

In our second instance, as long as $t_{i+1} - t_i \leq h$, then the cost would be $f(t_{i+1}, h - (t_{i+1} - t_i))$.

We simply want to take the smaller of these two options.

In implementing this solution, we write the function f recursively, adding a memoization table to avoid redundant recursive function calls. If we let n = the number of hydration spots and D = the total distance of the walk, the total run time of the solution is O(nD), since there are a total of at most nD unique input values to the recursive function. (Note: if our hydration level ever exceeds our distance out of the desert, there no need to call the recursion, because we can get out without buying any more water.)