Code Jam 2016 - Round 3

Analysis: Teaching Assistant

Teaching Assistant: Analysis

Let's define N as the number of days in the test. Notice that since a student is not allowed to do nothing on a test day, the student must request at least N/2 problem sets. Further, there is no reason to request more than that, since the extras cannot be submitted.

We'll begin with a dynamic programming solution that is sufficient for the Small dataset, and then discuss a mathematical formula that solves the Large dataset.

A dynamic programming approach

Because we can only submit the problem we requested most recently, we can view the problems that we hold as a stack, where requesting corresponds to pushing and submitting the most recently requested problem corresponds to popping.

Consider what happens on day 1. Clearly, we must request a problem set, and we now have to pick a day to submit it. Now, between the request and the submission, we can push to and pop from the stack, but after we are finished, the stack must contain the problem we requested on day 1, and nothing else. That means we need to have an even number of days between our request on day 1 and the corresponding submission.

We can think of the period of time between our request and our submission as a subproblem, and the time after the submission as another subproblem. Each of those subproblems must leave the stack the way they found it, so we can use the same set of rules that we used above for every subproblem.

Now, we'll express how to solve the problem in terms of recursion. At each stage, we'll be looking at some range of days, from day i to day j. So the subproblem (i,j) is to find the best possible score we can get over the days from i to j. We'll also say we can't touch the existing entries on the stack (corresponding to any days before i), and must leave the stack as we found it. We can then solve the subproblem as we did the original problem: request a problem set on day i, then pick where in the (i,j) interval to submit it such that there's an even number of days between the request and the submission.

At this point, each of our choices divides the subproblem into two additional, possibly-empty subproblems. For a subproblem of length 6, for example, here are the ways to place the request and submission:

```
_ _ _ _ _ _
    ↓
R S _ _ _ _
R _ _ S _ _
R _ _ _ _ S
```

Given the pairs of request and submission days, how do we decide which type of problem set to request? If we request the type that doesn't match the assistant's mood on request day, the best score we can get is 5 points. But we always get at least that much by requesting the type that matches the assistant's mood, so we should always do that. Then, the score we get when we submit just depends on whether the assistant's mood on submit day matches the type of problem we requested: we get 10 points total if they match, and 5 if they don't.

The empty subproblems are the base cases of our recursion. For larger problems, we find, for each possible submission point, the score we get from requesting and submitting that problem set (10 if the assistant's mood on request day matches its mood on submit day, 5 otherwise), plus the optimal scores of the subproblems. We can use this to recursively find the solution to the original problem; memoizing the solutions for the subproblems yields a DP solution. Because the memoization table is size $O(N^2)$ and it takes O(N) iterations to fill each cell, this algorithm is $O(N^3)$, which is sufficient to solve the Small problem.

A formulaic approach

It's also possible to solve the problem with a simple mathematical formula based on the input.

Count the number of even- and odd-numbered days on which the assistant is in the mood for Coding and Jamming. That gives us four numbers, which we'll call CE, CO, JE, and JO (for instance, CE is the number of even-numbered days on which the assistant is in the mood for Coding). We will show that the maximum possible score is:

$$S = 10 * (\min(CE, CO) + \min(JE, JO)) + 5 * \text{abs}(CE - CO).$$

We'll first show that this is an upper bound (we can't get a score higher than S), then show that it is tight (we can get a score equal to S).

First, note that any request/submit pair for the same problem (which we'll just call a pair) must have one even and one odd day; that is, for any problem set, it was either requested on an odd day and submitted on an even day, or vice versa. That must be true because we need an even number of days between the request and the submission. Now, since we need an even Coding day and an odd Coding day to make a pair where both days are Coding, the amount of those pairs that we can make is equal to the minimum of CE and CO. We can apply the same logic to Jamming. Then, we can pair up the leftovers; the amount of leftover pairs is the amount by which CE and CO differ (you can show that JE and JO differ by the same amount). No matter how we pair up the leftovers, there is no way to form more matched pairs.

To show that this bound is tight, we will construct a solution that achieves that upper bound. We make a list of the days in order. Then, whenever we see two adjacent days with the same mood, mark the first one as a request and the second as a submission, then remove those two days from our list. Observe that we can continue making pairs on this list, and they'll be valid on the real list (it's impossible for a new pair to be halfway inside and halfway outside any pair that we've removed, since we only remove pairs when there's nothing inside them). Continue this process until we can't anymore. At this point, since there aren't any adjacent days with the same mood, the mood must alternate between days. But that means that all even-numbered days have the same mood, and all odd-numbered days have the other mood. So we can pair all of those up, completing the proof.

A greedy approach

If you're curious, it's also possible to use a greedy stack-based algorithm to solve the Large problem. Our algorithm proceeds from left to right, and at each point decides whether to request or submit. We maintain a stack of problem sets that we hold, where a request is a push and a submit is a pop, then select our action each day as follows:

1. If we have no unsubmitted problem sets, request.
2. If we've made N/2 requests thus far, where N is the number of days in the test, submit.
3. If the top of the stack contains a problem set that matches the assistant's mood, submit.
4. Otherwise, request.

It's fairly difficult to see why this is valid on its own, but we can show that this also obeys the formula that we give above.

Both algorithms (the stack algorithm and the adjacent pair algorithm) find an adjacent pair with the same mood some number of times; we'll call these hits. Pairs with a different mood are misses. It remains to prove that the stack algorithm gets the optimal number of hits.

In most cases, both algorithms hit and miss in the same places. Intuitively, steps 3 and 4 of the stack algorithm correspond to finding and removing adjacent pairs with the same mood. Then, steps 1 and 2 clean up the remaining unmatched pairs.

There are some corner cases, however, where the algorithms don't precisely match. Take the input CJCJJC, for example: the stack algorithm produces RRRSSS, while the adjacent-pairs algorithm results in RRSRSS.

All of these corner cases occur when there's an unmatched pair that the stack algorithm doesn't match, failing to see that there's a matched pair later on at the same level. In other words, they all look like the following. (You can swap the Js and Cs, but we'll use the case that we show below without loss of generality.)

```
... J ... C ... J ... J ...
```

Then, the stack algorithm requests for the first two and submits for the last two, while the adjacent pairs algorithm matches the second pair, then the first pair. Notice that these have the same result either way: there's always one unmatched pair and one matched pair. Hence, the algorithms both produce the optimal score, even though they may attain it in different ways.