Code Jam 2020 - Round 3

Analysis: Pen Testing

How to approach this problem?

At first sight, it might seem impossible to achieve the number of correct guesses required in this problem. If we just pick two pens at random without writing anything, the probability of success will be 46.666...%. And whenever we write anything, the amount of remaining ink only decreases, seemingly just making the goal harder to achieve. And yet we are asked to succeed in 63.6% of test cases in Test Set 3. How can we start moving in that direction?

Broadly speaking, there are three different avenues. The first avenue is solving this problem in one's head or on paper, in the same way most algorithmic problems are solved: by trying to come up with smaller sub-problems that can be solved, then trying to generalize the solutions to those sub-problems to the entire problem.

The second avenue is more experimental: given that the local testing tool covers the entire problem (i.e., there is no hidden input to it), we can start trying our ideas using the local testing tool. This way we can quickly identify the more promising ones, and also fine-tune them to achieve optimal performance. Note that one could also edit the local testing tool to make it more convenient for experimentation; for example, one could modify the interaction format so that only one test case is judged at a time.

The third avenue starts with implementing a dynamic programming solution that can be viewed as the exhaustive search with memoization: in each test case, our state is entirely determined by how much ink have we spent in each pen, and by which pens are known to have no ink left. Our transitions correspond to the basic operation available: trying to spend one unit of ink from a pen. Moreover, we can collapse the states that differ only by a permutation of pens into one, since they will have the same probability of success in an optimal strategy. Finally, we need to be able to compute the probability of success whenever we decide to stop writing and pick two pens; we can do this by considering all permutations, or by using some combinatorics (which is a bit faster). This approach is hopelessly slow for **N**=15. However, we can run it for smaller values of **N** and then use it as inspiration: we can print out sequences of interactions that happen for various input permutations, and then generalize what happens into an algorithm that will work for higher values of **N**.

Test Set 1

How can we find a small sub-problem in which writing actually helps improve the probability of success? Suppose we have only three pens, with 1, 9 and 10 units of ink remaining. If we just pick two pens at random, we will have two pens with at least 15 units of ink between them only with probability 1/3: 9+10≥15, but 1+9<15 and 1+10<15. However, if we first write 2 units of ink with each pen, we will know which pen had 1 unit of ink since we will fail to write 2 units with it, and will therefore know that the other two pens have 7 and 8 units of ink remaining, so we can pick them and succeed with probability 1!

How can we generalize this approach to the original problem? We can pick a number K and write K units of ink with each pen. Then, we will know the identities of pens 0, 1, ..., K-1 so that we can avoid taking them to the South Pole. However, the other pens will have 0, 1, ..., **N**-K units of ink remaining, which seems to be strictly worse than the initial state. But we can include a small optimization: we can stop writing as soon as we find all pens from the set 0, 1, ..., K-1. For example, if the two last pens both have at least K units initially, we will stop before we reach them, so by the time we identify all pens from the set 0, 1, ..., K-1 and stop writing, we will not have touched those two pens at all. Therefore we will know that they both have some amount of ink from the set K, K+1, ..., **N**, and picking them gives us a much higher probability of success. Of course, this will not always be the case, and sometimes we will have only one or zero untouched pens. In this case we should pick the untouched pen if we have it, and add a random pen from those that have written K units successfully.

By running this strategy against the local testing tool with various values of K, we can learn that it succeeds in about 56.5% of all cases when K=3, and is good enough to pass Test Set 1. We will call this strategy *fixed-K*.

There are certainly many other approaches that pass Test Set 1 — for example, less accurate implementations of the solutions for Test Set 2 and Test Set 3 mentioned below.

Test Set 2

How do we make further improvements? Our current approach suffers from two inefficiencies. The first inefficiency has to do with the fact that we keep writing K units of ink with each pen even if we have already found the pen which had K-1 units of ink initially. Since we are now searching for 0, 1, ..., K-2, it suffices to write only K-1 units of ink with each subsequent pen at this point. More generally, if X is the most full pen from the set 0, 1, ..., K-1 that we have not yet identified, we only need to write X+1 units of ink with the next pen. Having identified all pens from the set 0, 1, ..., K-1, we stop writing and take the two pens not from this set that have written the least units to the South Pole. We will call this improvement *careful writing*.

It turns out that careful writing with K=4 increases our chances substantially to 61.9%, enough to pass Test Set 2!

The second inefficiency is that out of the pens that are determined to have at least K units, we take two to the South Pole, but the rest are useless. For example, we never end up taking any pen except the last K+2 pens to the South Pole! Therefore we could start by writing with the first **N**-K-2 pens until they are used up and have exactly the same success rate. We can improve the success rate if we use the information that we get from those pens to make potentially better decisions!

More specifically, we can do the following: initially, write with the first pen until it is used up (and therefore we know how much ink was in it). Then, write with the second pen until it is used up, and so on. In this manner, we will always know exactly which pens are remaining. At some point we should decide to stop gathering information and switch to the fixed-K strategy. The information we gathered can be used to pick the optimal value of K, which is potentially different in different branches.

This gives rise to a dynamic programming approach with $2^{\mathbf{N}}$ states: a state is defined by the set of pens remaining after we have used up some pens. For each state we consider either writing with the next pen until it is used up, or trying the fixed-K strategy with each value of K. Note that we only need to run the dynamic programming once before the interaction starts, and then we can use its results to solve all test cases.

We will call this improvement to the fixed-K strategy *pen exploration*. It turns out that pen exploration allows us to succeed in about 62.7% of all cases, which is also enough to pass Test Set 2. We expect that there are many additional ways to pass it.

Test Set 3

What about Test Set 3? You might have guessed it: we can actually combine careful writing and pen exploration! This yields a solution that succeeds in about 63.7% of all cases, which is close to the boundary of Test Set 3, so while it might not pass from the first attempt because of bad luck, it will definitely pass after a few attempts.

This is not the only way to solve Test Set 3, though. If we take a closer look at the careful writing solution, we can notice that we can achieve exactly the same outcome using a bottom-up approach instead of a left-to-right approach. More specifically, we start by writing one unit of ink using each pen in the order from left to right, until we find a pen that cannot write; we can conclude that it is the pen that started with 0 units of ink. Then, we can go from left to right again and make sure that each subsequent pen has written two units of ink (which will entail writing one more unit of ink from pens that have already written one, or writing two units of ink for pens that were untouched in the last round), until we find the pen that had 1 unit of ink in the beginning. We continue in the same manner until we have found pens 0, 1, ..., K-1, just as before. The amount written with each pen at this point will be exactly the same as the amount written with each pen in the original careful writing approach.

However, this formulation allows us to improve this approach: we no longer need to pick K in advance! We can instead make a decision after finding each pen X: either we continue by searching for the pen X+1 in the above manner, or we stop and return the two pens that have written the least so far. We will call this improvement *early stopping*. In order to make the decision of whether to continue or to return, we can either implement some heuristics, or actually compute the probability of success using a dynamic programming approach where the state is the amount written by each pen at the time we have found pens 0, 1, ..., X. This dynamic programming has just 32301 states for **N**=15, so it can be computed quickly.

Careful writing with early stopping works in about 64.2% of all cases if one makes the optimal decisions computed by the aforementioned dynamic programming, allowing one to pass Test Set 3 with a more comfortable margin. Once again, we expect that there are many additional approaches available for Test Set 3.

Ultimate solution

All three improvements — careful writing, pen exploration and early stopping — can be combined in one solution. It is a dynamic programming solution in which the state is once again the amount written with each pen, and we consider three options for every state:
1. Write with the leftmost remaining (=not known to be used up) pen until it is used up.
2. Write with all pens from left to right to find the smallest remaining pen.
3. Return the two rightmost pens that have not been used up yet.

Note that computing the probability of success for the third option is easy in this solution, as each of the remaining pens can be at any of the available positions. This means that any pair of remaining pens is equally likely to appear as the two pens we are taking to the South Pole. This means, in turn, that if the pens we are taking to the South Pole have written X and Y units, then the probability of success is simply the number of pairs of remaining pens with initial amounts A and B such that A+B>=**N**+X+Y, divided by the total number of pairs of remaining pens.

This dynamic programming has 1343425 states for **N**=15; therefore, it can run in time even in PyPy if implemented carefully. Its probability of success is around 64.4%, which is more than enough for Test Set 3.

Moreover, we have compared the probability of success for this approach with the optimal one (computed by the aforementioned exhaustive search with memoization, after *a lot* of waiting), and it turns out that for all **N**≤15 this solution is in fact optimal. Therefore, we suspect that it is optimal for higher values of **N** as well. We do not have a proof of this fact, but we would be very interested to hear it if you have one! Of course, neither this solution nor its proof are necessary to pass all test sets in this problem.

Constraints and judge

Finally, let us share some considerations that went into preparing the constraints and the judge for this problem. Given its random nature, the fraction of successfully solved test cases will vary for every solution. More specifically, thanks to the Central Limit Theorem we know that for a solution with probability of success P that is run on **T** test cases, the fraction of successfully solved test cases will be approximately normally distributed with mean P and standard deviation sqrt(P×(1-P)/**T**). In Test Sets 1 and 2 the standard deviation is therefore around 0.35%, and in Test Set 3 it is around 0.15%.

Considering the probabilities that a sample from a normal distribution deviates from its mean by a certain multiple of its standard deviation, we can see that for practical purposes we can expect to never land more than, say, five standard deviations away from the mean. This creates a window of about ±1.75% in Test sets 1 and 2, and a window of about ±0.75% in Test Set 3 such that whenever a solution is outside this window, it will always pass or always fail, but within this window the outcome depends on luck to some degree.

This luck window is inevitable and its size almost does not depend on the required success rate, only on the number **T** of test cases. Therefore it is impossible to completely avoid the situation when the success rate of a solution falls into the luck window, and therefore one might need to submit the same solution multiple times to get it accepted, with the number of required attempts depending on luck. Increasing the number **T** reduces the luck window, but increasing it too much requires the solutions to be extremely efficient and can rule out some approaches. Therefore, we decided that the best tradeoff lies around **T**=20000 for Test Sets 1 and 2, allowing less efficient approaches to still pass there. For Test Set 3, we felt the best tradeoff lies around **T**=100000 to better separate the solutions that are optimal or close to optimal from the rest and to reduce the luck window, while still not pushing the time limit to impractical amounts.

We have picked the required success rates in the following manner:
- For Test Set 1, we picked it such that the fixed-K strategy is outside the luck window and always passes.
- For Test Set 2, we picked it such that both careful writing and pen exploration are outside the luck window and always pass.
- For Test Set 3, we picked it such that the ultimate solution with all three improvements is outside the luck window and always passes, but solutions with careful writing and one of early stopping or pen exploration are within the luck window and therefore might require multiple submissions, but not too many. Lowering the threshold further in this test set would bring pen exploration without careful writing within the luck window. More generally, there is a continuum of solutions here if one uses inexact heuristics instead of exact dynamic programming to make the early stopping decision, and some of those solutions would inevitably land in the luck window.

Another peculiar property of this problem is that the judge is not deterministic, whereas typically in interactive problems, the randomness in each test set is fixed in advance, and submitting the same deterministic code always leads to the same outcome. This is necessary to avoid approaches in which one first uses a few submissions to learn some information about the test sets, for example by making the solution fail with Wrong Answer or Runtime Error or Time Limit or Memory Limit, to pass two bits of infomation back. This information can then be used to achieve a higher probability of success.

In order to see how this small amount of information can help significantly, consider a solution that has a probability of success that is five standard deviations smaller than the required one. Using the simple "submit the same code again" approach, one would need more than 3 million attempts on average to get it accepted, which is clearly not practical. However, we can do the following instead: we can submit a solution that just writes with all pens until they are used up, therefore gaining full knowledge of the test set. Then, it runs the suboptimal solution several million times with different random seeds, which by the above argument will find a random seed where the suboptimal solution in fact achieves the required success rate. Now the solution just needs to communicate back 20-30 bits comprising this random seed, which even at the two bits per submission rate requires just 10-15 submissions, which can be made within the space of a round. Finally, one could just submit the suboptimal solution with this random seed hardcoded, and pass.

We hope that this provides some insight into why this problem is designed the way it is, and we are sorry that some potential remained for contestants to be negatively impacted by the luck window or by the judge's non-determinism.