Code Jam 2021 - Round 1B

Analysis: Digit Blocks

Let $D_{i,j}$ be the digit on the block that has exactly j blocks under it in the i-th tower. The number written on the i-th tower is then

$$\sum_{j=0}^{\mathbf{B}-1} 10^j \cdot D_{i,j}$$

and the total score is

$$\sum_{i=1}^{\mathbf{N}} \sum_{j=0}^{\mathbf{B}-1} 10^j \cdot D_{i,j}.$$

Notice that each term in the last sum is independent from others, that is, each position in a tower has a particular value that gets multiplied by the digits, and exchanging blocks at the same heights that go on different towers does not change the final outcome. Our job to maximize the score is to get the largest digits paired up with the largest values. That means, putting large digits near the top of the towers. Moreover, the value of a position is more than 9 times the value of all the positions below it combined. Therefore, delaying the use of a position until we can get the maximum possible digit can be worth sacrificing the ability to decide on the order of all positions below it for up to 9 towers, and possibly a lot more (as even a random order gets some value).

All the approaches we present are based on trying to reserve high-valued positions for high-valued digits, sacrificing other positions.

Greedy strategies

The simplest greedy strategy is: try to place 9s in the top position of every tower, and sacrifice everything else. That means, when a block comes, if it is a 9, place it in the highest non-finished tower we have. If it is not, place it in the highest tower we have that has fewer than $\mathbf{B}-1$ blocks on it. This reserves the top spots for 9s and accelerates the availability of newer top spots by trying to build towers up as soon as possible.

There are other similar strategies, like doing the same but trying to place 9s in the top two spots of every tower, possibly pivoting to just the top spot when we are running out of time. Similarly, we can pivot to accepting lower digits for second-from-the-top spots or for top spots when there are few blocks left and the likelihood of enough 9s coming becomes low.

These simple strategies oscillate between getting a score of 87% and 91% of S, which means many of them are enough to pass Test Set 1, but none seems to be close to pass Test Set 2. More sophisticated heuristics can pass Test Set 2, but they need to consider at least the top 2 positions and also deal with tactically placing not just 9s but also 8s. These type of solution need careful tweaking and also comes with some uncertainty that it will eventually work out. The solutions in the next section address both those issues, ensuring the points and quite possibly saving us time.

Dynamic programming strategies

An alternative way is to try to maximize the expected score. This does not exactly maximize the probability of getting above a specific threshold (being more aggressive or conservative near the end depending on how close you actually are to the threshold might be better), but it's really close and much simpler.

The optimal such solution (the one that gets S as its expected score) is too slow, but it is worthwhile to consider it as a starting point. By linearity of expectation, when considering where to put a block, only the heights of all current towers matters, but not the actual values that were put there (since that is the score we will get regardless of all future decisions). We can therefore consider the multiset of tower heights the status, and optimize a function f(status, next_digit) that checks every possible placement for each possible next digit. The problem with this idea is, of course, the status space is too big for the time limit. This is what we did to calculate S, but we ran it for a lot longer than the time limit.

As we mentioned before, we can improve on the time limit by not caring too much about what happens with the low-value positions. That is, consider only a subset of the statuses. One example is this: for each digit we consider only the options of the first greedy strategy. We either put it in the top-most position of some tower, or we put it in the highest tower that has fewer than $\mathbf{B}-1$ blocks on it. This severely reduces the number of possible statuses, as there is only one tower that can have a height other than 0, $\mathbf{B}-1$ or $\mathbf{B}$. Instead of making the decision greedily, we leave the decision of which option to choose to the dynamic programming part. This solution is fast enough and gets about 96% of S as its score. Not enough for Test Set 2, but getting closer. Doing the same but leaving the top 2 positions up for optimization, on the other hand, gets over 99.5% of S and passes Test Set 2.

Since the top position represents close to 90% of the value, it makes sense that optimizing it gets 90% of the score. Similarly, the top two positions represent close to 99% of the value. More formally, consider two modified problems in which the top position or top two positions retain their value, but all other positions are valued as 0. Our proposed solutions get the maximum expected scores S_1 and S_2 for those modified problems, and we know that $S_1 > 0.9 \cdot S$ and $S_2 > 0.99 \cdot S$. Therefore, our solutions' expected score is guaranteed to be above each respective threshold, and the extra score we get from the lower positions gives us an additional gap to make the probability of success extremely high: our estimates are that each solution has less than 10^{-10} probability of failing the respective test set.