Code Jam 2018 - Round 1C

Analysis: Lollipop Shop

This problem's acceptance condition is based on competitive analysis. The problem is a bit tough to test locally, even for an interactive problem. It is easy to generate a particular set of customer preferences, but determining the maximum possible number of lollipops we can sell requires bipartite matching. However, we do not necessarily need to do that; since there is only one test set, which is Visible, we have some room for experimentation and for small leaps of faith.

A solution that just picks random legal flavors is not good enough to pass, so we need a couple of insights:

- Selling a lollipop, if possible, is never worse than not selling a lollipop. Assume that there is some good solution that involves not selling a lollipop now, and selling L lollipops in total later. Selling a lollipop now can only stop at most 1 lollipop from being sold later, so we can still sell L-1 more lollipops.
- If we have a choice of lollipops to sell, the best we can do is sell a lollipop that has the lowest probability of being liked. Intuitively, this saves the flavors that are more likely to show up later. (This intuition can be proved mathematically.)
- But we don't know the true probability for each flavor. The best estimate we can get for a flavor at any point is the number of customers that have liked that flavor so far, divided by the total number of customers seen so far.

Therefore, our best strategy is to always sell a lollipop if possible; whenever there are multiple flavors to choose from, we choose one of the flavors that we have seen the minimal number of times among previous customers' preferences. In problems like this, when we have to break a tie, we should do so at random just in case the problem setters have tried to anticipate and thwart particular strategies like always choosing the smallest ID number. (In this problem, it would have been impossible for the setters to penalize that, though!)

Notice that our solution would not work for an arbitrary list of probabilities, even with 200 customers. If every probability is 0.02, for example, our strategy loses its power (since all flavors are equally likely), and each flavor appears frequently enough to make our strategy do much worse than matching, but infrequently enough that we have very few chances to make the right choices. However, the problem guarantees that the probabilities are drawn randomly from the range [0.005, 0.1]. This should give us enough confidence that the probabilities will be different enough for our strategy to exploit.

The algorithm we have described can be shown to be at least as good as any other solution. But what if we had been less certain? In a problem with only Visible test sets, it is generally better to try (at the risk of a 4 minute penalty) than to spend more than 4 minutes worrying.