Kick Start 2022 - Coding Practice with Kick Start Session #1

Analysis: Milk Tea

View problem and solution walkthrough video

We are given two sets of binary strings:

1. Set of preferences of size $\mathbf{N}$
2. Set of forbidden milk teas of size $\mathbf{M}$

Each binary string in these sets is of length $\mathbf{P}$.

Test Set 1

There are only $2^{\mathbf{P}}$ different combinations of milk tea available. Since $\mathbf{P}$ is at most 10 for this set, the maximum number of possible combinations for a milk tea is 2^{10} or 1024. This is a small enough number to check each combination and calculate how many complaints you will get. Take the combination that gives the fewest complaints and that is not forbidden and output the number of complaints.

Test Set 2

For the this test set, the above approach will take too long to compute, so we need a different strategy.

Notice the following:

- Each option is represented by a bit. We select the best string of bits that fits the requirements.
- Disregarding forbidden combinations, a decision for which bit to choose for an option is independent of other bit decisions. This is because the options are independent from each other - every option will have a complaint or not for each friend, regardless of other options picked.
- Disregarding forbidden combinations again, we can get the best milk tea combination by selecting the most occurring bit in the set of preferences for each position in the string of bits. For example, given the set of preferences:
  ```
  { 1100
    1010
    0000 } the best milk tea would be:
    1000
  ```

So how do we deal with forbidden combinations? First notice that there are at most 100 forbidden combinations. The size of the forbidden set is $\mathbf{M}$; if we make $\mathbf{M}+1$ milk tea combinations, at least one of these combinations is not in the set of forbidden combinations. Since $\mathbf{M}$ is at most 100, it is feasible to create the best $\mathbf{M}+1$ combinations and take the best one that is not forbidden.

To generate these combinations, we can preprocess how many complaints we would get for each option. Then, we build up the best combinations one option at a time. We iterate through the options, at each option adding both possibilities (0 and 1) to the previous set of binary strings we created and select the best $\mathbf{M}+1$ results. We can tiebreak the results arbitrarily.

Let's consider why this works. Let S_k denote the best $\mathbf{M}+1$ combinations when we consider only the first k options. The key idea is to note that each combination in S_{k+1} has a combination from S_k as a prefix.

This is easy to show by contradiction. Take any binary string X from S_{k+1} and remove the last bit to get the prefix X'. Suppose, for contradiction, that X' is not S_k, thus all strings in S_k must have fewer complaints than X'. Take any of these strings in S_k and append the removed bit. This gives a combination which will have strictly fewer (when considering the tiebreak) complaints than X. This results in $\mathbf{M}+1$ binary strings with fewer complaints than X, which contradicts S being in S_{k+1}.

The final algorithm is as follows:

1. Precompute the scores (i.e. the number of complaints) for setting each bit in $\mathbf{P}$ to 0. The scores for setting a bit to 1 can then be computed by subtracting the score for 0 from $\mathbf{N}$, so we do not have to store these. To do this, iterate through 0 to $\mathbf{P}-1$, and at each iteration, iterate through the set of preferences, summing the number of 1s at that position.
2. Start with an empty set S_0: $S_0 = \{\text{""}\}$
3. Iterate from 1 to $\mathbf{P}$, generating S_k. To generate S_k (for $k > 0$):
    i. Take each binary string in S_{k-1} and append both a 0 and a 1. Since the set S_{k-1} has at most $\mathbf{M}+1$ strings, this will give at most $(\mathbf{M}+1)\cdot 2$ potential answers.
    ii. Remove extra strings so that the set has (at most) the best $\mathbf{M}+1$. To do this, you can maintain a dictionary of precomputed scores for each binary string in S_{k-1} and add the score for the appended bit to it. Then you can sort these scores and remove the largest ones.
4. Take the best combination from $S_{\mathbf{P}}$ that is not forbidden.

Time complexity

1. Precomputing the scores for each bit: $O(\mathbf{PN})$
2. Generating the empty set: $O(1)$
3. Iterating from 1 to $\mathbf{P}$: $O(\mathbf{P})$; in total for generating S_k: $O(\mathbf{M}\log\mathbf{M})$. So the total for generating all sets: $O(\mathbf{PM}\log\mathbf{M})$
    i. For each string in S_{k-1}: $O(\mathbf{M})$
        a. Appending 0 and 1: $O(1)$
        b. Computing the score for these two binary strings (remember, these are precomputed, you just need to add the score for the last bit to the score from the dictionary) and storing them in a dictionary takes: $O(1)$
    ii. Sorting the set and removing extra strings: $O(\mathbf{M}\log\mathbf{M})$
4. Finding the best combination from $S_{\mathbf{P}}$ that is not forbidden: $O(\mathbf{M})$

Thus, the total time complexity is $O(\mathbf{PN}+\mathbf{PM}\log\mathbf{M}+\mathbf{M})$.