Kick Start 2022 - Round H

Analysis: Magical Well Of Lilies

Test Set 1

For the small test set, a simple backtracking algorithm is sufficiently fast. In each recursive call, we need to keep track of the number of lilies remaining in the well, the current state of well's memory, and the number of coins we have tossed so far. In each recursive step, we have at most three choices:

- Toss one coin for another lily;
- Toss two coins for as many lilies as last noted by the well;
- Toss six coins ($4 + 2$) for the well to remember the current state of the basket and then toss as many lilies.

Of course, we would consider an operation only if the well has that many lilies, for it would be a waste of coins otherwise. If the number of lilies left in the well becomes 0 at any time of the recursion, we update the minimum number of coins, if necessary, and return immediately from the recursion.

Since we have at most three choices at each recursion level, the time complexity of this exhaustive search might seem like a prohibitive $O(3^{\mathbf{L}})$, however, the algorithm is much faster in practice for two reasons. First, not all three choices are available at each recursion level. And second, the maximal recursion depth of $\mathbf{L}$ is possible only if we pick up the lilies one by one. In other words, the average recursion depth is much smaller than $\mathbf{L}$. For example, the algorithm needs just 5060 recursive calls for $\mathbf{L} = 20$.

Test Set 2

For the large test set, the exhaustive search is obviously too slow as the number of recursive calls for $\mathbf{L} = 50$ is already $7,821,316,841$. Fortunately, dynamic programming comes to rescue here.

But first, let us make one useful observation. Namely, we can always rearrange some operations such that a two coin operation never follows a one coin operation. If it does, we can always swap the two operations without affecting the net effect of the whole sequence of operations.

Now let us turn to our dynamic programming recurrence. Let $DP[i]$ be the minimum number of coins needed to fetch i lilies from the well. We have $DP[0] = 0$ and $DP[1] = 1$. For $i > 1$, there are two possible ways of arriving at i lilies — we either collect $i - 1$ lilies first and then use a one coin operation to get another lily, or the last operation must have been a two coin operation. The latter means that we had asked the well to remember the state of our basket at k lilies (where k is a divisor of i) and then applied $\frac{i}{k} - 1$ two coin operations. Formally,
$DP[i] = \min(DP[i-1] + 1, \min_{k|i}\{DP[k] + 4 + 2(\frac{i}{k} - 1)\})$.

In practice, instead of calculating each $DP[i]$ directly using the formula above and iterating over the divisors of i, we inverse the process as follows:

At the beginning we can say that all values of DP except for 0 and 1 are equal to infinity. Now let us iterate through all i from 2 to $\mathbf{N}$. One option to use one coin and get from $i - 1$ lilies to i lilies, so we can say that $DP[i] = min(DP[i], DP[i-1] + 1)$. Now when we already have i

lilies, we can also use four coins to remember the state of our basket and then later get $j \cdot i$ lilies from the well. So we should update DP for all multiples of i:
$DP[j \cdot i] = min(DP[j \cdot i], DP[i] + 4 + 2 \cdot (j - 1))$ for $i < j \cdot i \leq \mathbf{N}$.

As for the time complexity of the algorithm, for each index k, there are $O(\frac{\mathbf{L}}{k})$ multiples to be updated, which leads to $O(\mathbf{L}(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{\mathbf{L}}))$ updates. The factor $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{\mathbf{L}}$ is the $\mathbf{L}$-th harmonic number, which is $O(\log \mathbf{L})$, therefore, the overall time complexity of the algorithm is $O(\mathbf{L} \log \mathbf{L})$.

As a final note, we can precompute the DP table in advance, and then answer each test case in constant time.