FIRST EUROPEAN JUNIOR OLYMPIAD IN INFORMATICS
SOFIA • BULGARIA • 2017

Analysis

The main thing that the contestants must have noticed in this problem was that **N** has at most six distinct prime divisors. Note that these 6 primes will be not only in the factorization of **N**, but also in the factorization of all of its divisors – the numbers, Elly is writing.

The first thing the contestants should do before getting to any solution is find the (up to) 6 prime factors of **N**. This can be done quite trivially, using the O(sqrt(N)) algorithm. Since **N** was up to 10^{15}, this would require around 32 million operations – a fraction of a second on a processor manufactured in the past 10 years.

Now, let's get to the main part of the problem. As with many other problems where the question is "how many" and the answer can be pretty large (thus, the modulus), here as well the solution is based on dynamic programming. We'll need to have a state, which tells us which divisors of **N** we can write at any given time.

In order two numbers in the list to be coprime (to have no common divisors greater than or equal to 2), they must share no prime number in their factorization. Thus, keeping track which of the primes have not been used, which have been used once, and which have been used twice should do the trick, right?

Well, almost. Unfortunately, it is not entirely enough and will most likely be a common problem the competitors face. It is fairly intuitive to go this way (use a ternary mask for the state), however we'll show that this is actually incorrect. Luckily for them, the examples are quite strong and reveal this!

The problem with the solution above is that it matters whether two primes (which were used once) were used in the same number or in two different numbers. Consider the following sequences:

1. (2, 3, 30)
2. (6, 30)

The first one is invalid, because 30 has common divisors with both 2 and 3 (two different numbers from the list). However, the second one is valid – 30 still has common divisors with 2 and 3, but this time they are used in the same number 6.

So, we need a state, which both:

- Keeps track which primes were used 0, 1, and 2 times.
- Keeps track which primes were used together.

We have several possible solutions. Let's start with the easiest one.

Approach 1 (keep all occurrences of each factor)
To get to this approach we need to make the observation that the list Elly writes can have at most 12 elements. This should be fairly obvious, as each prime can be used at most twice (and 6 * 2 = 12).

Instead of storing 0 (not used), 1 (used once), or 2 (used twice) for each prime, we can store not used (no restrictions), used twice (cannot use) or the position, at which it was used. Knowing the position where each of the primes was used helps us solve the problem we outlined above.

We will need 14 values for each prime: 0-11 will represent the position at which it was used, 12 will mean it was not used and 13 will mean it was already used twice. This means a DP with 14^6 = 7,529,536 states, which is okay from memory point of view (the answer fits into int, thus we'll need around 30 megabytes for the DP table).

In terms of time, for each state we have to choose which divisor of **N** to use. Since we have up to 6 primes, the number of divisors is 2^6 = 64. This looks bad at first, since, if we do the math, we'll need around $14^6 * 2^6$ = 481,890,304 operations in the worst case – more than we can afford for the given time limit. However, we can easily see that many of the states are invalid (can never be reached), thus shouldn't be counted towards that (see below for explanation why). In fact, running the last example (which has six prime divisors, i.e., should reach as many states as possible) reaches only 170,000 out of the 7,500,000 possible states. With that many actual states, the 64 operations per state are quite okay – this solution runs quite quickly on any input with these constraints, thus it gets 100 points.

Why are the valid states so few?
How can we deduce that many of the states will be invalid? Let's assume that as the first number in the list we use a divisor, which includes multiple factors. This drastically reduces the maximal length of the list. For example, if the first element is **N** itself (thus, uses all 6 factors), the length of the list is suddenly limited to 7.

Moreover, an average divisor of **N** has (1 + 6) / 2 = 3.5 (thus 3 or 4) prime factors. This means that the average length of a valid list is only around 12 / 3.5 = 3.42 (thus 3 to 4) numbers! From here we can conclude that, although there are valid sequences with 12 elements, *most* of the sequences will have only few. If we assume that in most cases the length of the list is no more than 6, our calculation for the number of reachable states would be $(6+2)^6$ = 262,144 – much closer to what we get empirically.

Approach 2 (map-based solution)
The contestants could exploit the low number of states, by using a map-based solution (instead of figuring out the perfect state). Since we don't care *where* in the list we have written each of the previous numbers, only which numbers we have written, we can instead use map <vector<int>, int> as a dynamic table. This approach is pretty simple and, as long as the vector is sorted, also relatively fast – it would get 80 points.

Approach 3 (smart solution)

We can do another observation (this time a hard one) to get a better state. We already saw that the number of reachable states is much lower than what our table covers. What if we make the table smarter (less sparse)?

We don't really care on which position each of the prime factors was used, only which pairs of prime factors cannot be used together anymore. For example, if we've already written 2 and 3, we know that we cannot use a number, which is divisible by 6. If we've written the numbers 6 and 35, we know that we cannot use the pairs (2, 5), (2, 7), (3, 5), and (3, 7) – thus the numbers 10, 14, 15, and 21 cannot be divisors of any new number.

How many pairs there are? Since (2, 3) and (3, 2) are the same, we'll only need information about 6 * 7 / 2 = 21 pairs. And we can use a bitmask for that.

This brings the table to 2^{21} = 2,097,152 states (slightly lower than the previous one). In fact, many of those are also not reachable. For example, if we know that the pairs (2, 3) and (5, 7) are forbidden, but (3, 5) is not, then we can conclude that (2, 5) should also be forbidden (because 3 and 5 were apparently used in the same number). Checking this empirically shows that only slightly over 3000 states were actually reached! It turns out that this solution is not only better in terms of memory efficiency, but also much faster as runtime as well. The number of states is so low, that even a map-based solution with this idea gets a full score.

I'd be interested in what is the best state the contestants come up with, since even this approach is very inefficient in terms of state.

Naive (bruteforce) approaches

Of course, the contestants could have tried other simpler approaches – for example bruteforce. Depending on how smart the backtrack was written, they could have gotten quite a lot of points. This was done on purpose, since the smart solutions are on the hard side. The trivial bruteforce, for example, was supposed to get around 40 points. An optimized one could get around 60.