Constructor Spring Challenge sample problems

Sample MATH problem

The value of the expression

$$10 \cdot 20 \cdot 30 \cdot 40 \cdot 50 \cdot 60 \cdot 70 \cdot 80 \cdot 90$$

is **not** a perfect square. What is the minimum number of factors that need to be removed so that the product becomes a perfect square (i.e. equal to a square of an integer)?

Solution/Proof:

Notice that only one of the factors is divisible by 7 (and 7 is a prime number). Therefore, we certainly must remove "70".

The next step is a bit more complicated. Observe divisibility by powers of 2, 3, and 5 (i.e., the amount of 2, 3, 5 in the prime decomposition of the factors):

$$\begin{aligned}10 &= 2^1 \cdot 3^0 \cdot 5^1\\ 20 &= 2^2 \cdot 3^0 \cdot 5^1\\ 30 &= 2^1 \cdot 3^1 \cdot 5^1\\ 40 &= 2^3 \cdot 3^0 \cdot 5^1\\ 50 &= 2^1 \cdot 3^0 \cdot 5^2\\ 60 &= 2^2 \cdot 3^1 \cdot 5^1\\ 80 &= 2^4 \cdot 3^0 \cdot 5^1\\ 90 &= 2^1 \cdot 3^2 \cdot 5^1\end{aligned}$$

So a total of 15 twos, 4 threes, and 9 fives. Yet, in order for the product to be a perfect square, the total amount of each its prime factor should be even. So we need to somehow get rid of:

1 or 3 twos
0 or 2 threes
1 five

We could remove "10". The result would become equal to $(720000)^2$.
Or we could remove "90". The result would become equal to $(240000)^2$.
Or we could remove "40". The result would become equal to $(360000)^2$.
Either way the minimum total number of factors that should be removed (so that what remains is a perfect square) is 2.
■

Sample PROGRAMMING problem

Consider the expression

$$10 \cdot 20 \cdot 30 \cdot 40 \cdot 50 \cdot 60 \cdot 70 \cdot 80 \cdot N$$

where N is an integer, $1 \leq N \leq 10000$, given as the only input. Find and output the minimum number of factors that need to be removed so that the product becomes a perfect square (i.e. equal to a square of an integer). Remark: the product of 0 factors is equal to 1.

Example

Input
90

Output
2

Here is an example of a solution in Python based on a mathematical reasoning. Namely, the product of all numbers except for "70" and "N" is a perfect square. Of course, this is not obviuos just by looking at the expression, but some analysis of powers of primes can lead to this conclusion (just like in the solution of math problem above). So all we need to do is to check whether entire product is a perfect square (if it is, then the answer is 0, nothing to remove), and if it is not, then try to get rid of just one of the factors (if successful, then the answer is 1).

```python
import math

def is_perfect_square(number: int) -> bool:
    if number < 0:
        return False
    return math.isqrt(number) ** 2 == number

def check_squareness(N: int) -> int:
    M = 70 * N

    if is_perfect_square(M):
        return 0

    for divisor in (10, 20, 30, 40, 50, 60, 70, 80):
        if M % divisor == 0 and is_perfect_square(M // divisor):
            return 1

    return 2

N = int(input())
print(check_squareness(N))
```

Or, if you don't feel like digging into math, and even prefer not to optimize your code, this very problem can be solved with the use of brute force. Simply exhaust all possible sets of numbers to be removed:

```python
import math

def is_perfect_square(number: int) -> bool:
    if number < 0:
        return False
    return math.isqrt(number) ** 2 == number

def find_largest_square_subset(N: int) -> int:
    elements = [10, 20, 30, 40, 50, 60, 70, 80, N]
    max_size = 0

    def backtrack(index, current_product, count):
        nonlocal max_size
        if index == len(elements):
            if is_perfect_square(current_product):
                max_size = max(max_size, count)
            return

        backtrack(index + 1, current_product, count)
        backtrack(index + 1, current_product * elements[index], count + 1)

    backtrack(0, 1, 0)
    return 9 - max_size

N = int(input())
print(find_largest_square_subset(N))
```

However, keep in mind that brute force might not work for large data.

Good luck!