Solutions for 19th of July

Accumulated Data Plan

This problem was supposed to be the easiest problem, but it seems we were wrong :(

As the problem statement says that we should calculate only how many megabytes I could use in the $(N+1)$-th month, let's try to focus on finding that.

From the 1st month to the $(N+1)$-th, the data plan gives me $X \cdot (N+1)$ megabytes. And I used $P_1 + P_2 + \cdots + P_n$ megabytes in total. So the remaining megabytes I can use is just the difference, $X \cdot (N+1) - \sum P_i$. We can just calculate the formula and print it out.

I wrote exactly this solution, why am I considered wrong?

From the statement, $X \leq 10^5$ and $N \leq 10^5$. So $X \cdot (N+1)$ could be big as $10^{10} + 10^5$, which cannot be stored in a 32-bit integer type variable. You had to use 64-bit integer type to store this value.

Bots Buying Beer

The order of desired prices are not important, so let's sort the sequence $p_1, p_2, \cdots, p_m$. Now, $p_1 \leq p_2 \leq \cdots \leq p_m$.

The most important thing you should know is that it is always optimal to **set the price as $\boldsymbol{p_i}$ for some $\boldsymbol{i}$.**

> Proof: Suppose the price is set to x such that $p_i < x < p_{i+1}$, and this is the optimal solution (maximizes the revenue) However, even if we increase the price to p_{i+1}, the number of bots that are willing to buy a can of beer doesn't change. Therefore the revenue increases, which leads to a contradiction.

So, suppose we set the price to p_i. The number of bots that are willing to buy the beer is $m - i + 1$. However, we have only n cans of beer. Therefore the formula of the revenue is

$$p_i \times \min(m - i + 1, n)$$

Iterate all i from 1 to n, and find the maximum value of the revenue, and print it out.

I wrote exactly this solution, why am I considered wrong?

$p_i \leq 10^{12}$, $n \leq 10^5$. So the revenue might be as big as 10^{17}, which is impossible to store in an 32-bit integer variable. You had to use 64-bit integer type to store this value.

Careful Multiplication

We actually wrote almost everything in the statement.

First, it is impossible to solve this problem by calculating `((p % m) * (q % m)) % m`, because $m \leq 10^{18}$ and $(10^{18} - 1)^2$ is too large to store in a 64-bit type integer.

So we gave an another approach in the Hints section on the statement. Try to think of a possible solution, and then take a look at our solution.

Let's define a function $f(a, b, m)$ which calculates the value of $(a \times b) \bmod m$. This function can be defined recursively, by this:

- Base case: $f(a, 1, m) = a \bmod m$.
- If b is odd, $f(a, b, m) = \{f(a, \mathbf{b\text{-}1}, m) + a\} \bmod m$
- If b is even, $f(a, b, m) = \{f(a, \mathbf{b/2}, m) \times 2\} \bmod m$

In this recurrence, we only add two numbers that are at most 10^{18}, so we can calculate the desired value only using 64-bit type integers. The code is simple too:

```
long long f (long long a, long long b, long long m) {
  if(b == 1) {
    return a % m;
  }else if(b % 2 == 1) {
    return (f(a, b-1, m) + a) % m;
  }else {
    return (f(a, b/2, m) * 2) % m;
  }
}
```

The time complexity of this solution is $O(\log n)$ since each transition only consists of a constant number of operations.

Division

The intended solution was to find the recursive structure from the problem and writing a recursive function that does the division.

Let's define a function $div(p, q, d)$. This function prints d digits after the decimal point of $\frac{p}{q}$, where $p < q$.

Refer to the picture in the problem statement. As $p < q$, we multiplied p by 10 and obtained $10p$, and then divided this number by q and obtained the quotient $\lfloor\frac{10p}{q}\rfloor$ and the remainder $10p \bmod q$. The quotient becomes the first digit after the decimal point of $\frac{p}{q}$, and the remaining $(d-1)$ digits comes from $\frac{10p \bmod q}{q}$.

So, we can write a recursive function like this. Do not forget to print the string "0." first.

```
void div (int p, int q, int d) {
  if(d == 0) return;
  printf("%d", p * 10 / q);
  div(p * 10 % q, q, d - 1);
}
```

Why converting into double and then dividing doesn't work?

`double` use only 8 bytes for storing a real number. In this problem, we are asked to print at most **100 digits** after the decimal point, so we should be accurate to 10^{100}. However, 8 bytes = 64 bits, so a `double` can only distinguish 2^{64} different states. $2^{64} \ll 10^{100}$, so it was impossible to store all digits correctly in a `double`.

Endless Waiting

We think this problem is the hardest one.

Let's define a function $f(t)$, which calculates the *maximum number of people that can do the banking in t seconds.* It is obvious that $f(t)$ is monotonically increasing (if the time is extended, more people can come!). Therefore we can use a binary search to find the minimum t that satisfies $f(t) \geq M$.

So, how to calculate $f(t)$? Let's try to see each desks independently, because each desk just process banking and don't interrupt each other. If desk $\#i$ doesn't rest and keep banking for t seconds, the desk can process $\left\lfloor \frac{t}{T_i} \right\rfloor$ people. Then, we can just sum the values up to calculate $f(t)$

$$f(t) = \sum_{i=1}^{n} \left\lfloor \frac{t}{T_i} \right\rfloor$$

Now, let's apply the binary search on integers to get the value. This is an excerpt of the code. We set the upper bound of the interval as $10^5 \times 10^5$ because there might be a case that there is only one desk that takes 100,000 seconds per person and there are 100,000 people waiting.

```
long long low = 0, high = (ll)1e5 * (ll)1e5;
long long ans = high;

while(low <= high) {
     long long mid = (low + high) / 2;
     if(f(mid) >= M) {
            ans = mid;
            high = mid - 1;
     }else {
            low = mid + 1;
     }
}
```

Flowering Plants

Given f_1, f_2 and f_3, the problem asks to calculate the value of $\operatorname{lcm}(f_1, f_2, f_3)$. There are 10,000 test cases to solve in 1 second, so the algorithm should be super fast.

As we know how to calculate $\gcd(a, b)$ in $O(\log \max(a, b))$ time (Euclidean algorithm), we are going to use this function. We can calculate the least common multiple of a and b using the formula below:

$$\operatorname{lcm}(a, b) = \frac{a \times b}{\gcd(a, b)}$$

And then, it is true that

$$\operatorname{lcm}(a, b, c) = \operatorname{lcm}(\operatorname{lcm}(a, b), c)$$

We will not prove the statement here, maybe you could find it by yourself.

So we can use this formula and print the answer. Of course, you had to use 64-bit integer type because the product of three numbers is as large as 10^{18}.

GOLDBACH'S CONJECTURE

The problem statement says there always exists an answer, so let's try to find an answer such that the difference between two primes are the smallest.

Suppose the answer is p and $n-p$, and WLOG $p \le n-p$. The difference between these two numbers is $(n-p)-p = n-2p$. Therefore to minimize the difference, p should be close to $\frac{n}{2}$. And as $p \le n-p$, $p \le \frac{n}{2}$ must always hold. Therefore we can iterate all p from $\frac{n}{2}$ to 2, and try to find whether p and $n-p$ are all prime, and if they are both prime, we break the loop and print the answer.

```
for(int p = n/2; p >= 2; p--) {
    if(is_prime(p) && is_prime(n - p)) {
           printf("%d %d\n", p, n - p);
           break;
    }
}
```

There are several ways to write the `is_prime` function. These are some possible methods:

- Use the $O(\sqrt{n})$ time primality test.
- Precalculate all primes under 10^5 using the Sieve of Erathosthenes.

This was working because actually we can find the answer really fast. (p is almost always near $\frac{n}{2}$, so we don't iterate much)