1 Chinese Remainder Problem

Claim
The task is only possible if $k < a_i$ for all i.

With that annoying edge case out of the way, let's assume that $k < \min(a)$ and solve the problem.

Problem Solving Technique: Solve an easier version first
It can be easier to reckon with, and might give you insight to the full version of the problem.

Always do the subtasks first. Even then, don't feel limited to needing subtasks—make up your own!

Forget about ℓ and r for now.

Is there *any* integer x such that $x \bmod a_i = k$ for all i?

Well, yes! Here's an obvious answer. Take k itself. Since $k < a_i$ for all i, it is guaranteed that $k \bmod a_i = k$ always.

How about other solutions?

Standard Toolbox: Solution families
In math, a common technique is to ask:

- If I have *one* solution to a problem, can I use that to generate *more* solutions?

If you do, then starting from one "primitive" solution, you can produce an entire *family* of other solutions.

We know something about k (it has our desired property), so let's try to express other solutions in terms of k. For an arbitrary integer x, let $d := x - k$, allowing us to write $x = k + d$.

Then, if we desire $x \bmod a_i = k$...

$$\begin{aligned} x \bmod a_i &= k \\ (k+d) \bmod a_i &= k \\ (k \bmod a_i) + (d \bmod a_i) &\equiv k \\ k + (d \bmod a_i) &\equiv k \\ d \bmod a_i &\equiv 0. \end{aligned}$$

Thus, we see that d has to be *divisible by all the* a_i, and there is a standard characterization of such integers: they are the multiples of the **least common multiple** of all the a_i.

Claim

An integer x satisfies $x \bmod a_i = k$ for all i *if and only if* it can be written as $k + mt$, where $m = \text{lcm}(a)$ and t is a non-negative integer.

To find any integer within the range $[\ell, r]$, we note that k and m are fixed values (the LCM can be computed from all the a_i), and our only degree of freedom is t.

Standard Approach: Algebra

Precisely mathematically describe the problem or scenario, in symbols. This enables you to use algebra—we teach that in school for a reason.

Say we wanted to find the largest valid integer x that is $\leq r$. So, we write:

$$k + mt \leq r,$$

and solving for t,

$$t \leq \frac{r-k}{m}$$

Since we want the largest non-negative t which is *also an integer*, we get:

$$t = \left\lfloor \frac{r-k}{m} \right\rfloor \text{ if this is non-negative, else 0.}$$

So, let $x := k + mt$ using this t. If $\ell \le x \le r$, then that is our answer. If not, then no solution can exist.

Implementation

Note that the product of two numbers $\le 10^{10}$ can reach up to 10^{20}, which **does not fit in a 64 bit data type**.

So, when computing the LCMs in C++, you must either use `__int128`, or otherwise have some special check *before* multiplying for if the product would overflow (instead of `a*b < LIM`, you can test `a < LIM/b` to dodge overflow).

Implementation

Note that the LCM can be **massive**. Even if you used Python or some other BigInteger class, you cannot literally compute $m = \text{lcm}(a)$ without running out of time, because the numbers are so large that *performing the arithmetic* will cause you to run out of time.

Note that in the worst case, if a were the 10^5 largest primes $\le 10^{10}$, then their LCM would just be their product, so your code would deal with values up to $\approx 10^{50}$. GCD and LCM between integers of this size is too slow for us to do 10^5 times.

So, we'll need a cute trick.

We note that $\left\lfloor \frac{r-k}{m} \right\rfloor = 0$ whenever $m > r$ (and $r - k \ge 0$).

- Try to compute $\text{lcm}(a)$ naively.
- But, if a running value ever becomes $> r$, just stop immediately and set $t := 0$.