Code Jam 2011 - Round 1C

Analysis: Perfect Harmony

This problem turned out to be very, very tricky, due to a number of cases to consider and the possibility of integer overflow. Thus, we will go over the solution with some care.

To solve the small data set for this problem, it is enough to iterate over all notes that can be played by Jeff, and for each of them check whether it is in harmony with all the other notes. Note that there are two cases to consider for each note - either its frequency has to divide the frequency of Jeff's note, or it has to be divisible by it.

For the large data set, this strategy will not be sufficient - there are too many notes that Jeff can play to check.

We will begin by sorting all the input frequencies. Now assume that the frequency of the note that Jeff will play (let us denote it by **F**) is somewhere between frequencies $\mathbf{f_k}$ and $\mathbf{f_{k+1}}$. This means, in particular, that all the frequencies $\mathbf{f_1}$, $\mathbf{f_2}$, up to $\mathbf{f_k}$ are no larger than **F**; so **F** has to be divisible by all of them. This means that **F** has to be divisible by their Least Common Multiple (which we will denote LCM($\mathbf{f_1}$, $\mathbf{f_2}$, ..., $\mathbf{f_k}$)). Similarly, **F** has to divide the Greatest Common Divisor of $\mathbf{f_{k+1}}$ up to $\mathbf{f_N}$.

Calculating the GCDs and LCMs

To make any use of this information, we need to calculate all the LCMs of sets $\mathbf{f_1}$, ..., $\mathbf{f_k}$ up to any **k**, and also the GCDs of sets $\mathbf{f_{k+1}}$, ..., $\mathbf{f_N}$ for any k.

Let us recall that the GCD of two numbers **a** and **b** can be calculated using the Euclidean algorithm in O(log(**a** + **b**)) time. To calculate the LCM of two numbers, we use the formula LCM(**a**, **b**) = **a** * **b** / GCD(**a**, **b**). Using this, we can calculate all the needed GCDs and LCMs in O(**N**) GCD operations, inductively. For instance, having already calculated the first **k-1** LCMs, we calculate the **k**th as follows: LCM($\mathbf{f_1}$, ..., $\mathbf{f_k}$) = LCM(LCM($\mathbf{f_1}$, ..., $\mathbf{f_{k-1}}$), $\mathbf{f_k}$), and the first of those numbers is already calculated.

Note that one can also calculate all the GCDs and LCMs directly, in O($\mathbf{N}^2$) GCD operations. With the limit of 10^4 on GCD this should also run in time.

One final comment to make here is that when calculating the LCMs, we should be careful to avoid overflow. It may turn out that the LCM of some of the input frequencies does not fit into a 64-bit integer (in general, the LCM of 10^4 numbers, each up to 16 digits, can have even 160,000 digits!). However, note that Jeff cannot play notes with frequency greater than 10^{16}, thus if the LCM of any numbers turns out to be greater than 10^{16} (or even greater than **H**) we can safely replace it by 10^{16} + 1 without changing the answer - Jeff will not be able to play a note with a frequency divisible by this LCM anyway. Thus, when we calculate the formula **a** * **b** / GCD(**a**, **b**) we should first divide any of the two numbers, say **a**, by the GCD, then check whether the resulting product exceeds **H** (e.g. by checking whether (**H** + 1) / **b** ≥ **a** / GCD(**a**, **b**), and perform the multiplication only if it does not.

Special cases first

Before going on, we should also consider that there are two special cases when the analysis above does not apply - when **F** divides all the input frequencies, and when **F** is divisible by all of

them.

There are a number of ways to take care of them. For the first, the easiest is to add another note with frequency 1 to the input. This will not make Jeff's task harder, as any number is divisible by 1, and on the other hand will assure that **F** is always divisible by at least one of the numbers in the input.

For the upper bound, no analogous trick exists (there is no number that is divisible by any **F** Jeff might choose; one might consider the LCM of all the numbers that Jeff's instrument can play, but this is usually too large. Thus, if we fail to find a solution **F** lying between any two of the input frequencies, we have to consider this case separately. Fortunately, it is not complicated - if **C** is the LCM of all the input numbers, then the result will be the smallest multiple of **C** that is greater or equal **L**, provided it is no larger than **H**.

The standard cases

Notice that as we are looking for the lowest frequency Jeff can play, we can investigate the possibilities one by one - first check if there is a solution between $\mathbf{f_1}$ and $\mathbf{f_2}$, if yes - return it (recall that $\mathbf{f_1}$ is 1, so there is no solution smaller than $\mathbf{f_1}$). If no solution was found, look between $\mathbf{f_2}$ and $\mathbf{f_3}$, and so on. Finally, if no solution is found between $\mathbf{f_{N-1}}$ and $\mathbf{f_N}$, we consider the special case analyzed above.

Now for the crux of the problem - how can we check (quickly) whether a solution is to be found between $\mathbf{f_k}$ and $\mathbf{f_{k+1}}$? Note that this interval can possibly contain up to 10^{16} numbers, so a brute force check is unsatisfactory.

Recall that if **F** is between $\mathbf{f_k}$ and $\mathbf{f_{k+1}}$, then it has to divide GCD($\mathbf{f_{k+1}}$, ..., $\mathbf{f_N}$) (we will denote this number by **D**), and it has to be divisible by LCM($\mathbf{f_1}$, ..., $\mathbf{f_k}$) (we will denote this number by **C**). Thus, in particular, if **C** does not divide **D**, we know there is no solution in this interval.

There are two more easy cases to consider. If the intervals [**L**, **H**] and [**C**, **D**] are disjoint, there is obviously no solution in this interval. If **C** lies in the interval [**L**, **H**] (and divides **D**, which we already checked), it is obviously the smallest solution in this interval, so we may safely return it.

Notice that there is at most one interval for which those easy (and in particular, constant time) checks will not suffice. Indeed, if for some **k** the intervals [**L**, **H**] and [**C**, **D**] are not disjoint, then for any subsequent interval [**C'**, **D'**] obtained for a different **k'** either **C'** lies in [**L**, **H**], or the two intervals are disjoint, as $\mathbf{L \le D \le f_{k+1} \le C'}$.

Finally, we can concentrate on this one interval. We want to find the smallest number **F** in the interval [**L**, **H**] that divides **D** and is divisible by **C**. For this, it is enough to consider all the divisors of **D** and check them one-by-one. The divisors of **D** can be enumerated in time proportional to the square root of **D** - for each divisor **d**, either **d** or **D**/**d** is no larger than the square root of **D**, thus to find all the divisors we check all the numbers no larger than the square root, and if a number **d** divides **D**, we add both **d** and **D**/**d** to the list of divisors. This algorithm returns the divisors almost sorted, so it is easy to consider them in ascending order and find the first that both is divisible by **C** and falls into the interval [**L**, **H**].

Summary

This was not an easy problem, and required quite a lot of care and attention. Let us enumerate the steps, to wrap it up:

- Sort all the input frequencies (in O(**N** log **N**) time)
- Add 1 to the beginning of the list of inputs (in O(1) time :))

- Calculate the prefix LCMs and suffix GCDs (in O(**N**) GCD operations, each taking O(log **H**) time (as we do not consider results greater than **H**)
- For each **k** from 1 to **N**-1 check whether the appropriate LCM divides the appropriate GCD (if no, proceed to next interval); if the LCM falls into [**L**, **H**] (if yes, return the LCM) and whether the intervals [LCM, GCD] and [**L**, **H**] intersect (if no, proceed to the next interval). This takes constant time for each interval, so O(**N**) time in total.
- If we are still analyzing this interval, find all the divisors of the GCD and check them one by one. This takes O(sqrt(**H**)) time.
- If no answer was found as yet, it remains to check the smallest multiple of the LCM of all the inputs that is greater or equal than **L**. This is done in constant time.

There are other approaches possible to this problem, too. For instance, one may analyze all the divisors of the largest input frequency, and for each of them use binary search to find the interval that contains it and (in constant time, using precalculated LCMs and GCDs) check whether it is a correct solution. We encourage you to analyze the details of this approach yourself.