Implicit Prime Factorisation - Codeforces

Those of us who have been doing cp for a long time, have come to love the representation of numbers as an infinite prime power vector i.e.

$$n = \prod p_i^{e_i} = [e_1, e_2, ...] = v_n$$

Where n is a number and p_i is the i th prime number. This has many useful properties for problems, such as mapping

$$\begin{gathered} n \times m \sim v_n + v_m \\ n \,/\, m \sim v_n - v_m \\ n \mid m \sim v_n \leq v_m \\ \gcd(n, m) \sim \min(v_n, v_m) \\ lcm(n, m) \sim \max(v_n, v_m) \end{gathered}$$

All the above operations are done elementwise. However in some cases, the problem authors may forbid you from factorising the numbers, by perhaps increasing n to around 10^{36} where there isn't any simple factorisation algorithm to do it in reasonable time. However using effective prime factorisation, we can use these properties freely on a set of numbers. If this set contained all integers, then it would require us to truly prime factorise the integer. However, we only need to use these operations on a subset of integers, making our task a lot simpler. We will make a finite array q of size t with the following properties.

(1) Any element in $x \in S$ is representable by q, i.e. there exists $[f_1, f_2, ...f_t] = v'_n$ such that $x = \prod q_i^{f_i}$

(2) For all the above operations, replacing v_n with v'_n should produce the same result.

Let us show, that if q satisfies (1) and $gcd(q_i, q_j) = 1$ for all $i < j$, then q satisfies (2).

Proofs

Let us use $[q_i]v_n$ as the power of q_i in the representation of n.

Divisibilty

Let us assume there exists q_i such that $[q_i]v_n > [q_i]v_m$. Then q_i must divide some product of q, with the power of $q_i = 0$. However all other elements in q are coprime to q_i, therefore it is not possible.

If there is no such q_i its trivial to see $n \mid m$.

GCD

Let us consider $min(v_n, v_m)$. Since $\min(v_n, v_m) \leq v_n$ and $\min(v_n, v_m) \leq v_m$, therefore $v_{min} = \min(v_n, v_m)$ is a common factor. If we show $\gcd(n \,/\, g, m \,/\, g)$

$= 1$, or $\gcd\,(v_n - min(v_n, v_m), v_m - min(v_n, v_m)) = 1$, we're done.

Notice that for any q_i, $[q_i]v_n - [q_i]v_{min}$ or $[q_i]v_m - [q_i]v_{min}$ is 0. If we look at any true prime p, it may only divide one element in q. However that element cannot be in both elements as shown above. Therefore their GCD must be 1. And thus proved.

LCM

By the basic relation between gcd and lcm we get $lcm(n, m) = \frac{nm}{\gcd(n,m)} = v_n + v_m - min(v_n, v_m) = max(v_n, v_m)$

Let us now consider how we can find q given S. Let us proceed element wise, and assume we have a valid solution, and we want to add another element.

Now let us add the next element $x \in S$. Go through q until we spot an element that has a common factor with x. Let that element be q_i.

Let $P(q)$ be the set of elements produced by some product of elements in q. Notice that if q has 2 elements that divide each other $a \mid b$, then a, b is replaceable by $a, \frac{b}{a}$, and P will not lose any elements. Let us do this recursively until neither divides the other, or an element becomes 1. Let us call this procedure `reduce_pair`.

(It maybe useful to look at the example implementation in parallel from here)

We run the procedure `reduce_pair` on the elements q_i, x, if x becomes 1, we're done. If q_i becomes 1, we remove it from the set and continue.

We then run an iterative algorithm on a list L, initialised with $[q_i, x]$. Let the elements before $L[ptr]$ be all co-prime. Initialise ptr with 1. Run `reduce_pair` on all elements before ptr with it. Then add their gcd to the end of the list if its not one, and divide both elements by it. This will make all elements upto $ptr + 1$ coprime. There can only be as many elements in this list as there are true primes in q_i, x. Therefore we can deduce that the size of the list will be at most $O(\log \log max(S))$.

Since this algorithm will be run $O(\log x)$ times, the total algorithm will run in $O(n^2 \log A + n \log A \times (\log \log A)^2)$ Where $A = max(S)$, and we will get our array q.

Example implementation : 159590539