Greatest Common Divisors

Hello, and welcome to Competitive Programming! Today we are going to talk about using the greatest common divisor in contests. In particular, we are going to talk about Euclid's algorithm and the extended Euclid's algorithm.

Objectives

When this is done, you'll be able to implement Euclid's algorithm to find the GCD of two numbers. You'll also be able to make use of the extended Euclid's algorithm to solve Diophantine equations. As an extra added bonus you'll even be able to explain what a Diophantine equation actually is.

The GCD

Calculating the GCD

You can calculate the GCD using the very well known Euclid algorithm.

You start with two integers (a) and (b), and we get to assume that they are both positive and that (a>b).

The formula is this: that the gcd of a and b is the gcd of b and the modulus of a and b.

It will be helpful if you understand why that is the case.

(next)

So first, let's consider this property of gcd. If g divides a and g also divides b, then g also divides the sum of a and b and also the difference of a and b. In fact, any linear combination of a and b will work.

So we could make a recursive solution like this one: gcd of a and b is the gcd of a-b and b. You may need to swap the position in the recursive call so that the larger of the two is in the (a) position.

This works, but it could be very slow if (a) is large compared to (b).

(next)

So instead of repeatedly subtracting off b from a until a is the smaller element, let's find the largest number (n) such that (a - b n) is still non-negative.

(next)

Well, we know how to get that: that is just the modulus.

Let's look at an example now.

Example

Here's an example, the GCD of 90 and 25. This is equal to GCD of 25 and 15, which is equal to the GCD of 15 and 10, which is equal to the GCD of 10 and 5, which is equal to the GCD of 5 and 0, which is just 5.

You have a choice now. On the next slide we have a small example worked out for you, and you will probably be fine going forward, but it will make your problem solving skills better if you pause this and verify these formulae for yourself.

Extended Euclidean Algorithm

Diophantine Equations

A diophantine equation is a polynomial equation in which we are only interested in integer solutions.

A diophantine equation of the form ax + by = 1 is called a linear diophantine equation. These are interesting in part because we can usually solve for both x and y with a single equation.

As an example, let's suppose that you went to a store and bought (x) apples for 72 cents each and (y) oranges for 33 cents each. I suppose from the prices you could deduce that we are in a more tropical country, since oranges cost less than apples, but anyway….

You end up spending $5.85, or 585 cents in total, and we want to know how many apples and how many oranges we purchased.

To solve this we are going to have to use an extended version of the Euclidean algorithm.

Extended Euclidean Algorithm

So, the idea is that we want to solve an equation (a x + b y = 585). Well, we don't have a direct formula for that, but we can solve for (ax + by = g = gcd(a,b)). Since we know (a) and (b) at the start, we can calculate (g) quickly. But how can we get (x) and (y)?

It turns out we can make use of the recursive structure of the GCD algorithm to solve this inductively. Here's how it works.

(next)

Suppose we know the values for (x) and (y) for one step. At the base case, for instance, we will have 1 for (x) and 0 for (y).

But now instead of writing ax + by = g, we undo one level of recursion from our original GCD algorithm. This would give you ($bx_1 + (a \mod b) y_1 = g$).

(next)

The definition of a mod b is a - floor of a over b times b. If we substitute that into the a mod b part, …

(next)

… and then rearrange things …

(next)

and after a final cloud of algebra you get this.

This new x and y will be the factors of the caller's version of a and b.

The code

It might be easier to understand what we are doing if you see the code. This computes the CGD recursively, and on the way out it computes the x and y factors.

ICPC allows you to bring reference material to the contest, and this code is something you might want to have, though it's probably not that bad to memorize it, especially if you are able to understand the math.

The Example

Let's look at the example again. The table shows the values for (a) and (b) as we recurse through the gcd algorithm. To compute (x) and (y), we start with 1 and 0.
(next)

The next few steps of the formula we compute the factors (x) and (y) such that (a x + b y = 3).

(next next next)

Our final result is (x = -5) and (y = 11).

An example, ctd.

Now lets apply this to our example. Since 585 is 3 times 195, we multiply both sides of our equation by 195.

This gives us (x = -975) and (y = 2145). So apparently we sold 975 apples and bought 2145 oranges.

What, you don't think that sounds right?

Well, it turns out that there are an infinite number of solutions to this equation.

(next)

We can add multiples of 72 times 33 over 3 to the x side and subtract those multiples from the y side and still preserve the properties of the formula. We just have to find the right value for n now. If we say that x must be positive, then we solve (-975 + 11n > 0) for (n), and get (n > 88.6). Rounding up gives us (n=89).

Adding those terms to both sides gives us this final formula, with a lot more reasonable interpretation that we bought 4 apples and 9 oranges.

That is the basic algorithm. Since the number of solutions to such an equation is infinite, problem setters will usually add some other constraint to the problem to make it finite, as we did here.

Next
Transcript — Fenwick Trees →