Euclidean algorithm for computing the greatest common divisor

Table of Contents

Given two non-negative integers a and b, we have to find their **GCD** (greatest common divisor), i.e. the largest number which is a divisor of both a and b. It's commonly denoted by $\gcd(a, b)$. Mathematically it is defined as:

$$\gcd(a, b) = \max_{k=1\ldots\infty\,:\,k|a\,\wedge k\,|b} k.$$

(here the symbol "$|$" denotes divisibility, i.e. "$k \mid a$" means "k divides a")

When one of the numbers is zero, while the other is non-zero, their greatest common divisor, by definition, is the second number. When both numbers are zero, their greatest common divisor is undefined (it can be any

arbitrarily large number), but we can define it to be zero. Which gives us a simple rule: if one of the numbers is zero, the greatest common divisor is the other number.

The Euclidean algorithm, discussed below, allows to find the greatest common divisor of two numbers a and b in $O(\log \min(a, b))$.

The algorithm was first described in Euclid's "Elements" (circa 300 BC), but it is possible that the algorithm has even earlier origins.

Algorithm

The algorithm is extremely simple:

$$\gcd(a, b) = \begin{cases} a, & \text{if } b = 0 \\ \gcd(b, a \bmod b), & \text{otherwise.} \end{cases}$$

Implementation

```
int gcd (int a, int b) {
    if (b == 0)
        return a;
    else
        return gcd (b, a % b);
}
```

Using the ternary operator in C++, we can write it as a one-liner.

```
int gcd (int a, int b) {
    return b ? gcd (b, a % b) : a;
}
```

And finally, here is a non-recursive implementation:

```
int gcd (int a, int b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}
```

Correctness Proof

First, notice that in each iteration of the Euclidean algorithm the second argument strictly decreases, therefore (since the arguments are always non-negative) the algorithm will always terminate.

For the proof of correctness, we need to show that $\gcd(a, b) = \gcd(b, a \bmod b)$ for all $a \geq 0$, $b > 0$.

We will show that the value on the left side of the equation divides the value on the right side and vice versa. Obviously, this would mean that the left and right sides are equal, which will prove Euclid's algorithm.

Let $d = \gcd(a, b)$. Then by definition $d \mid a$ and $d \mid b$.

Now let's represent the remainder of the division of a by b as follows:

$$a \bmod b = a - b \cdot \left\lfloor \frac{a}{b} \right\rfloor$$

From this it follows that $d \mid (a \bmod b)$, which means we have the system of divisibilities:

$$\begin{cases} d \mid b, \\ d \mid (a \mod b) \end{cases}$$

Now we use the fact that for any three numbers p, q, r, if $p \mid q$ and $p \mid r$ then $p \mid \gcd(q, r)$. In our case, we get:

$$d = \gcd(a, b) \mid \gcd(b, a \mod b)$$

Thus we have shown that the left side of the original equation divides the right. The second half of the proof is similar.

Time Complexity

The running time of the algorithm is estimated by Lamé's theorem, which establishes a surprising connection between the Euclidean algorithm and the Fibonacci sequence:

If $a > b \geq 1$ and $b < F_n$ for some n, the Euclidean algorithm performs at most $n - 2$ recursive calls.

Moreover, it is possible to show that the upper bound of this theorem is optimal. When $a = F_n$ and $b = F_{n-1}$,

$gcd(a, b)$ will perform exactly $n - 2$ recursive calls. In other words, consecutive Fibonacci numbers are the worst case input for Euclid's algorithm.

Given that Fibonacci numbers grow exponentially, we get that the Euclidean algorithm works in $O(\log \min(a, b))$.

Least common multiple

Calculating the least common multiple (commonly denoted **LCM**) can be reduced to calculating the GCD with the following simple formula:

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

Thus, LCM can be calculated using the Euclidean algorithm with the same time complexity:

A possible implementation, that cleverly avoids integer overflows by first dividing a with the GCD, is given here:

```
int lcm (int a, int b) {
    return a / gcd(a, b) * b;
}
```

Practice Problems

- Codechef - GCD and LCM