Euclid's algorithm of finding the GCD (greatest common divisor)

Given two non-negative integers $a$and b. Required to find their greatest common divisor, ie the largest number that divides both $a$and b. English "greatest common divisor" is spelled "greatest common divisor", and its symbol is common $\gcd$:

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

(Here the symbol " $|$"denotes the divisibility, ie" $k|a$"means" $k$divides a")

When it out number is zero, and the other is non-zero, their greatest common divisor, by definition, it is the second number. When the two numbers are zero, the result is undefined (infinite any suitable number), we assume in this case, the greatest common divisor of zero. So we can say about such a rule: if one of the numbers is zero, then their greatest common divisor is the second number.

Euclid's algorithm , discussed below, solves the problem of finding the greatest common divisor of two numbers $a$and $b$for $O(\log \min(a,b))$.

This algorithm was first described in the book of Euclid's "Elements" (about 300 BC), although it may, this algorithm has an earlier origin.

Algorithm

The algorithm is extremely simple and is described by the following formula:

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

Implementation

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

Using the ternary conditional operator C + +, the algorithm can be written even shorter:

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

Finally, we give shape and a non-recursive algorithm:

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

Proof of correctness

Please note that for each iteration of the Euclidean algorithm its second argument is strictly decreasing, therefore, as it is non-negative, then the Euclidean algorithm **always terminates** .

To **prove correctness** we need to show that $\gcd(a, b) = \gcd(b, a \bmod b)$for any $a \geq 0, b > 0$.

We show that the quantity in the left-hand side is divided by the present on the right and the right-hand - is divisible by standing on the left. Obviously, it would mean that the left and right sides of the same, and that proves the correctness of Euclid's algorithm.

Denoted $d = \gcd(a, b)$. Then, by definition, $d|a$and $d|b$.

Next, we expand the remainder of the division $a$on $b$through their private:

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

But then it follows:

$$d \mid (a \bmod b)$$

So, recalling the statement $d|b$, we obtain the system:

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

We now use the following simple fact: if for any three numbers p, q, rcomplete $p|q$and $p|r$then executed and: $p \mid \gcd(q, r)$. In our situation, we obtain

$d \mid \gcd(b, a \bmod b)$

Or by substituting $d$its definition as $\gcd(a, b)$we obtain:

$\gcd(a, b) \mid \gcd(b, a \bmod b)$

So, we spent half the evidence to show that the left side of the right divides. The second half of the proof is similar.

Operation time

Time of the algorithm is evaluated **Lame theorem** which establishes a surprising connection Euclid's algorithm and the Fibonacci sequence:

If $a > b \geq 1$and $b < F_n$for some n, the Euclidean algorithm performs no more $n - 2$ recursive calls.

Moreover, we can show that the upper bound of this theorem - optimal. When $a = F_n, b = F_{n-1}$it will be done $n - 2$recursive call. In other words, **successive Fibonacci numbers - the worst input** for Euclid's algorithm.

Given that the Fibonacci numbers grow exponentially (as a constant in power n), we find that the Euclidean algorithm runs in $O(\log \min(a, b))$multiplication operations.

LCM (least common multiple)

Calculation of the least common multiple (least common multiplier, lcm) reduces to the calculation $\gcd$the following simple statement:

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

Thus, the calculation of the NOC can also be done using the Euclidean algorithm, with the same asymptotic behavior:

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

(Here divided into profitable first $\gcd$, and only then is multiplied by b, as this will help to avoid overflows in some cases)

Literature

- Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. **Algorithms: Design and analysis of** [2005]