Codestudio > Library > Competitive Programm... > Advanced Level >

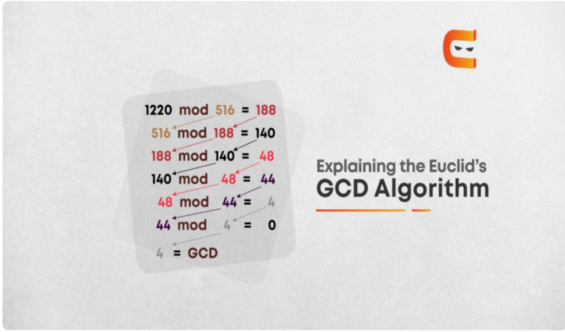
GCD Euclidean Algorit...

Related Articles

- Catalan Numbers
- Count Number of Valid Parenthesis
- Minimum Number of Arrows to Burst Balloons
- Chinese remainder theorem

Algorithm

K Kushleen Waraich
Last Updated: Oct 31, 2022



One of the earliest known numerical algorithms is that developed by Euclid (the father of geometry) in about 300 B.C. for computing the Greatest Common Divisor (GCD) of two positive integers. Euclid's algorithm is an efficient method for calculating the GCD of two numbers, the largest number that divides both of them without any remainder.

Algorithm:

- The first way of doing it if we try to subtract the smallest number from the greatest, GCD remain the same and in this way, if we keep repeating this step we'll finally get GCD.
- But as above process subtraction could be time-consuming then instead of subtracting what we do, we divide the smaller number, the algorithm stops when we find remainder 0.

Let GCD(x,y) be the GCD of positive integers x and y.
If x = y, then obviously GCD(x,y) = GCD(x,x) = x
Euclid's insight was to observe that, if x > y, then GCD(x,y) = GCD(x-y,y).

Actually, this is easy to prove. Suppose that d is a divisor of both x and y. Then there exist integers q_1 and q_2 such that $x = q_1 d$ and $y = q_2 d$. But then $x - y = q_1 d - q_2 d = (q_1 - q_2)d$. Therefore d is also a divisor of x-y.

Using similar reasoning, one can show the converse, i.e., that any divisor of x-y and y is also a divisor of x. Hence, the set of common divisors of x and y is the same as the set of common divisors of x-y and y. In particular, the largest values in these two sets are the same, which is to say that GCD(x,y) = GCD(x-y,y).

For improving the other we can also use, (For less no. of iterations) GCD(x,y) = GCD(x%y,y).

For illustration, the Euclidean algorithm can be used to find the greatest common
divisor of a = 1071 and b = 462.
1071 mod 462= 147
462 mod 147= 21
147 mod 21 =0
Since the last remainder is zero, the algorithm ends

Implementation: If we compare this algorithm with a naïve approach (brute force), what we generally try to do. Let's see How?

Naïve Approach:

```
//We'll start from 1 to smallest of the two number until we find a number
that divides into both of them.

Int gcd (int a, int b) {
    Int ans=1;
    For(int i;i<=min(a,b);i++) {
        If (a%i ==0 && b%i ==0){
            ans=i;
        }
    }
}
```

But Euclid's method is faster than this naive method, So lets we follow the Euclidean method to find out the GCD of 4598 and 3211. We represent the two number to the following way, Dividend should be the large number and Divisor is another number.we will repeat the process until the remainder is equal to zero.

Dividend = Divisor * Quotient + Remainder
4598 = 3211 * 1 + 1387
3211= 1387 * 2 + 437
1387 = 437 * 3 + 76
437 = 76 * 5 + 57
76= 57 * 1 + 19
57 = 19 * 3 + 0

At the point, the remainder is 0 and we get the GCD 19. Notice that every step dividend and devisor are exchanged and remainder and divisor exchanged. When we remainder is 0 that means the remaining divisor our GCD.

Euclid's GCD Algorithm:

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

Time Complexity: O(Log min(a, b))

Binary Euclidean Algorithm: This algorithm finds the gcd using only subtraction, binary representation, shifting and parity testing. We will use a divide and conquer technique. The following function calculate gcd(a, b, res) = gcd(a, b, 1) · res. So to calculate gcd(a, b) it suffices to call gcd(a, b, 1) = gcd(a, b).

12.3: Greatest common divisor using binary Euclidean algorithm.

```
def gcd(a, b, res):
    if a == b:
        return res * a
    elif (a % 2 == 0) and (b % 2 == 0):
        return gcd(a // 2, b // 2, 2 * res)
    elif (a % 2 == 0):
        return gcd(a // 2, b, res)
    elif (b % 2 == 0):
        return gcd(a, b // 2, res)
    elif a > b:
        return gcd(a - b, b, res)
    else:
        return gcd(a, b - a, res)
```

This algorithm is superior to the previous one for very large integers when it cannot be assumed that all the arithmetic operations used here can be done in constant time. Due to the binary representation, operations are performed in linear time based on the length of the binary representation, even for very big integers. On the other hand, modulo applied in algorithm 10.2 has worse time complexity. It exceeds O(log n · log log n), where n = a + b. Thus the time complexity is O(log(a · b)) = O(log a + b) = O(log n).

Least Common Multiple:

The least common multiple (lcm) of two integers a and b is the smallest positive integer that is divisible by both a and b. There is the following relation:

lcm(a, b) = a·b/gcd(a,b)

Knowing how to compute the gcd(a, b) in O(log(a+b)) time, we can also compute the lcm(a, b) in the same time complexity.

Extended Euclidean Algorithm:

Although Euclid GCD algorithm works for almost all cases we can further improve it and this algorithm is known as the Extended Euclidean Algorithm. This algorithm not only finds GCD of two numbers but also integer coefficients x and y such that:

ax + by = gcd(a, b)

Input: a = 35, b = 15

Output: gcd = 5

x = 1, y = -2

(Note that 351 + 15(-2) = 5)

Basically, what this algorithm does, it updates the results of gcd(a,b) using the results calculated by recursive call gcd(b%a, a). Let values of x and y obtained by the recursive calls are x1 and y1. Then x and y are as follows:

x = y1 – (b/a)*x1

y= x1

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

This implementation of the extended Euclidean algorithm produces correct results for negative integers as well.

Approach	Time Complexity	Space Complexity
Brute Force Solution	O(min(number1,number2))	O(1)
Euclidean Algorithm	O(max(number1, number2))	O(1)
Optimized Euclidean Algorithm	O(log(number1+number2))	O(log(number1+number2))

Some Problems Based on Euclid's Algorithm:

- Program to find the LCM of two numbers using GCD.
- Program to find GCD of floating-point numbers.
- Program to find the common ratio of three numbers.
- Program to find GCD of an array of integers.
- Program to find the sum of squares of N natural numbers.
- For more such problems or enhance your skills, you can also practice on our coding platform Codezen.

To read more, click here.

Previous Article   Next Article