

0.1 GCD and Bezout's identity

- *Algorithm*: Euclidean (algo. 1), Extended Euclidean (algo. 2)
- *Input*: Two integers a and b
- *Complexity*: $\mathcal{O}(\log(\min(a, b)))$
- *Data structure compatibility*: N/A
- *Common applications*: Modular arithmetic, such as RSA encryption

Problem. GCD and Bezout's identity

Given two integers a and b , find out the greatest common divisor d , and the Bezout's identity x and y such that $ax + by = d$.

Description

GCD is the abbreviation of the greatest common divisor, which is important in cryptography because it can decide whether two integers are coprime or not [1]. Assume that $a > b$, the trivial way to calculate the GCD is to use a loop, and perform a modular calculation at each step from 1 to b to find it. However, when the integer is very large, the running time of this method can be very low, since it has a time complexity of $\mathcal{O}(b)$. Therefore, the Euclidean algorithm is designed to solve GCD in a faster way, which has a time complexity of $\mathcal{O}(\log(\min(a, b)))$. Also assume that $a > b$, first calculate $r = a \bmod b$, then repeat the process for b and r and so on, until the remainder reaches 0, and the previous divisor b is the result.

Euclidean algorithm

Suppose that it takes N steps to use Euclidean algorithm to calculate the GCD. Denote f_N as the N_{th} number of Fibonacci series, and we can prove that $a \geq f_{N+2}$ and $b \geq f_{N+1}$ using mathematical induction. Since The N_{th} Fibonacci number has the expression

$$f_N = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^N - \left(\frac{1 - \sqrt{5}}{2} \right)^N \right] \approx \phi^N$$

where $\phi \approx 1.618$, then golden ratio. So we can get $N \approx \log_{\phi}(f_N)$. Assume that $a > b$, then we can deduce that $f_{N+1} \approx b$, and get $N + 1 \approx \log_{\phi}(b)$, and we finally get to the point that the time complexity of Euclidean algorithm is $\mathcal{O}(\log(\min(a, b)))$.

Algorithm 1: Euclidean

Input : Two integers a, b

Output: The greatest common divisor d

```
1 Function GCD( $a, b$ ):  
2   if  $b = 0$  then  
3     return  $a$ ;  
4   end if  
5   return GCD( $b, a \bmod b$ )  
6 end
```

Bezout's identity and Extended Euclidean algorithm

Bezout's identity claims that given two integers a, b and their greatest common divisor d , we can find a unique pair of coefficients x, y such that $ax + by = d$. Extended Euclidean algorithm is the extension of the traditional Euclidean Algorithm, which is created to get the Bezout's coefficient of a and b . It shares the same time complexity with the traditional Euclidean algorithm, which is $\mathcal{O}(\log(\min(a, b)))$. Moreover, it calculates the two coefficients x and y at the same time of the GCD d during recursions.

Algorithm 2: Extended Euclidean

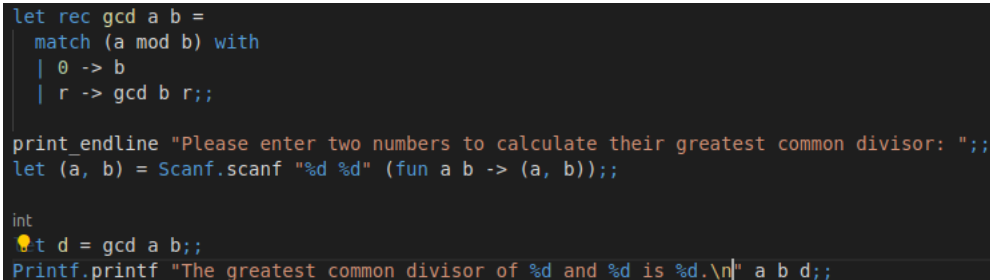
Input : Two integers a, b

Output: a tuple d, x, y , The greatest common divisor d , and the Bezout's identity x, y

```
1 Function extendedGCD( $a, b$ ):
2   if  $b=0$  then
3     | return ( $a, 1, 0$ );
4   end if
5   ( $d1, x1, y1$ )  $\leftarrow$  extendedGCD( $b, a \bmod b$ );
6    $d \leftarrow d1$ ;
7    $x \leftarrow y1$ ;
8    $y \leftarrow x1 - a/b * y1$ ;
9   return ( $d, x, y$ )
10 end
```

OCaml implementation

Below is the OCaml code screenshot, and the code will be attached in the compressed file.

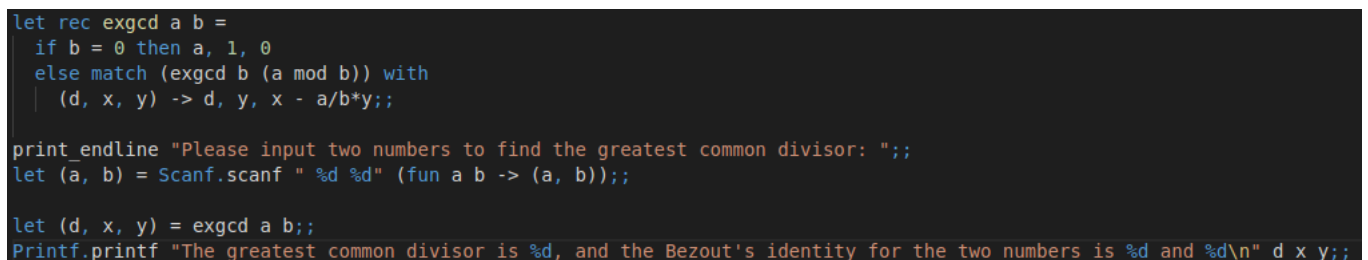


```
let rec gcd a b =
  match (a mod b) with
  | 0 -> b
  | r -> gcd b r;;

print_endline "Please enter two numbers to calculate their greatest common divisor: ";
let (a, b) = Scanf.scanf "%d %d" (fun a b -> (a, b));;

int
let d = gcd a b;;
Printf.printf "The greatest common divisor of %d and %d is %d.\n" a b d;;
```

Figure 1: Implementation of Euclidean algorithm



```
let rec exgcd a b =
  if b = 0 then a, 1, 0
  else match (exgcd b (a mod b)) with
  | (d, x, y) -> d, y, x - a/b*y;;

print_endline "Please input two numbers to find the greatest common divisor: ";
let (a, b) = Scanf.scanf "%d %d" (fun a b -> (a, b));;

let (d, x, y) = exgcd a b;;
Printf.printf "The greatest common divisor is %d, and the Bezout's identity for the two numbers is %d and %d\n" d x y;;
```

Figure 2: Implementation of Extended Euclidean algorithm

References.

- [1] Manuel. VE477 – Introduction to Algorithms (lecture slides). 2019 (cit. on p. 1).

0.2 Factorization (Multi Precision Quadratic Sieve)

- *Algorithm*: Fermat's method (algo. 3), Quadratic sieve (algo. 4)
- *Input*: A positive number needs to be factorized.
- *Complexity*: $e^{(1+\mathcal{O}(1))\sqrt{\ln n \ln \ln n}}$
- *Data structure compatibility*: N/A
- *Common applications*: Large number factorization, like RSA.

Problem. Factorization (Multi Precision Quadratic Sieve)

Given a positive number n , find its factorization using quadratic sieve.

Description

To clearly understand how the Quadratic sieve works, we should first look at Fermat's factorization method.

Fermat's factorization method

It is obvious that if an odd integer n can be represented by $n = a^2 - b^2$, then it can be factorized as $n = (a+b)(a-b)$. In this sense, if $n = cd$, where c and d are two odd numbers, n can be expressed as $n = (\frac{c+d}{2})^2 - (\frac{c-d}{2})^2$, which is the general form [1]. However, it is similar to trivial division to some extent, so it takes a long time to finally find the square collision.

Algorithm 3: Fermat's method

Input : A number n which needs factorization

Output: a tuple (c, d) , which satisfies $n = cd$

```
1 Function FermatFact( $n$ ):  
2    $a \leftarrow \lceil \sqrt{n} \rceil$ ;  
3    $b \leftarrow a^2 - n$ ;  
4   while  $b$  is not a square number do  
5      $a \leftarrow a + 1$ ;  
6      $b \leftarrow a^2 - n$ ;  
7   end while  
8    $c \leftarrow a + \sqrt{b}$ ;  
9    $d \leftarrow a - \sqrt{b}$ ;  
10  return  $(c, d)$   
11 end
```

Next, we need to know some basic knowledge about smooth numbers.

Smooth numbers

It is known that every positive integer can be represented as the multiplication of the power of prime numbers, in the form $n = \prod p_i^{e_i}$, where p_i and e_i denote for the prime number and its corresponding power. Then it comes to the definition of smooth numbers. A positive integer is called B -smooth if none of its prime factors exceeds B [2].

Quadratic sieve

Since it is quite slow to factorize a large integer using trivial division or Fermat's factorization method, the quadratic sieve was invented to make factorization faster. It first chooses a *B-smooth* bound where B is chosen to meet the requirement that the Legendre symbol $(\frac{n}{p_i}) = 1$, where p_i denotes for all the prime numbers that are not greater than B . Usually, B will not be greater than 40. Then, the number of primes less than B , often known as $\pi(B)$, is used to determine the interval to choose integers to try for square congruence modular n , commonly $I = [\lfloor \sqrt{n} \rfloor - \pi(B), \lfloor \sqrt{n} \rfloor + \pi(B)]$ [3].

Then we will try the squares of integers in the interval modular n , which each gives a congruence that can be represented by multiple of prime factors. Then using linear algebra, we can find some combination such that for $a \in I, b \in I, a^2 \cdot b^2 \equiv c \pmod{n}$, where c is a square number and can be expressed as the multiplication of prime numbers each has an even power. By finding this, we can calculate the GCD (greatest common divisor) $\gcd(a+b, n)$ and $\gcd(a-b, n)$, the two numbers will be the factorization of n [3].

Take a very simple example, considering the number 1649. First take the floor of its square root, which is $\lfloor 1649 \rfloor = 40$. Then, $41^2 \equiv 32 \pmod{1649}$, $42^2 \equiv 115 \pmod{1649}$, $43^2 \equiv 200 \pmod{1649}$. We can find that $32 * 200 = 6400 = 80^2$ is a square, thus $41^2 \cdot 43^2 \equiv 114^2 \equiv 80^2 \pmod{1649}$. Hence, $(114 - 80) \cdot (114 + 80) \equiv 34 \cdot 194 \equiv 0 \pmod{1649}$, which means that there exists a positive integer k such that $34 \cdot 194 = k \cdot 1649$. Therefore, $1649 = \gcd(34, 1649) \cdot \gcd(194, 1649) = 17 \cdot 97$ [4].

Algorithm 4: Quadratic sieve

Input : A number n which needs factorization

Output: a tuple (c, d) , which satisfies $n = cd$

```
1 Function QuadraticSieve( $n$ ):
2   Choose a smooth bound  $B$ ;
3    $t \leftarrow \pi(B)$ ;
4   for  $i \leftarrow 1$  to  $t + 1$  do
5     choose an integer  $x$  from  $I = [-t, t]$ ;
6      $a_i \leftarrow x + \lfloor \sqrt{n} \rfloor$ ;
7      $b_i \leftarrow a_i^2 - n$  /*  $b_i = \prod_{j=1}^t p_j^{e_{ij}}$  */;
8      $v_i \leftarrow (e_{i,1} \bmod 2, e_{i,2} \bmod 2, \dots, e_{i,t} \bmod 2)$ ;
9     Let  $T \subseteq \{1, 2, \dots, t+1\}$  such that  $\sum_{k \in T} v_k = 0$  in  $\mathbb{F}_2$ ;
        /*  $\mathbb{F}_2$  denotes for the finite field of base 2 */;
10     $x \leftarrow \prod_{k \in T} a_k$ ;
11     $y \leftarrow \prod_{j=1}^t p_j^{\sum_{k \in T} e_{kj}}$ ;
12    if  $x \not\equiv y \pmod{n}$  and  $x \not\equiv -y \pmod{n}$  then
13      | continue;
14    else
15      |  $c \leftarrow \gcd(x + y, n)$ ;
16      |  $d \leftarrow \gcd(x - y, n)$ ;
17      | return  $(c, d)$ ;
18    end if
19  end for
20 end
```

References.

- [1] Sounak Gupta and Goutam Paul. *Revisiting Fermat's Factorization for the RSA Modulus*. 2009. arXiv: [0910.4179](#) [cs.CR] (cit. on p. 3).
- [2] Carl Pomerance. "The Role of Smooth Numbers in Number Theoretic Algorithms". In: *Proceedings of the International Congress of Mathematicians*. Ed. by S. D. Chatterji. Basel: Birkhäuser Basel, 1995, pp. 411–422. ISBN: 978-3-0348-9078-6 (cit. on p. 3).

- [3] Carl Pomerance and Paul Erdős. "A Tale of Two Sieves". In: 1998 (cit. on p. [4](#)).
- [4] Jr Samuel S. Wagstaff. *The Joy of Factoring*. American Mathematical Society, 2013, pp. 195–202. ISBN: 978-1-4704-1048-3 (cit. on p. [4](#)).