## Euclid's Algorithm

Andreas Klappenecker CPSC 629 Analysis of Algorithms

We begin this course by revisiting an algorithm that is more than 2000 years old. The algorithm was described in Book VII of Euclid's *Elements*, but it is certainly much older. You might recall that Euclid was one of the first faculty members of the 'Museum' of Alexandria, an institution of higher education comparable to our universities today, which was founded around 300 B.C. Euclid's *Elements* is a beautiful work that logically developed geometry and other branches of mathematics; it is without a doubt one of the most influential works in the history of science.

**Integer Divisibility.** Let us first recall some terminology, so that we have a common language. An integer d is called a **divisor** of an integer a if there exists an integer a' such that a = a'd. An integer d is a **common divisor** of integers a and b if there exist integers a' and b' such that a = a'd and b = b'd.

The greatest common divisor of two integers a and b that are not both 0 is a common divisor d > 0 of a and b such that all other common divisors of a and b divide d. We denote the greatest common divisor of aand b by gcd(a, b). It is sometimes useful to define gcd(0, 0) = 0.

Notations. We write  $d \mid a$  for the fact that d is a divisor of a. We follow Knuth and write  $a \perp b$  if the integers a and b are coprime, i.e., when gcd(a, b) = 1.

**Euclid's Algorithm.** Euclid's algorithm calculates the greatest common divisor of two positive integers a and b. The algorithm rests on the observation that a common divisor d of the integers a and b has to divide the difference a - b. Indeed, if a = a'd and b = b'd for some integers a' and b', then a - b = (a' - b')d; hence, d divides a - b.

Algorithm E (Subtractive Algorithm for Greatest Common Divisors) Input: a, b positive integers Output: d = gcd(a, b)

while  $(a \neq b)$  do  $(a,b) = (\min(a,b), \max(a,b) - \min(a,b));$ return a; We have used the compound assignment (a, b) = (c, d) that assigns the values of c and d respectively to a and b at the same time. An invariant of the algorithm is the relation

$$gcd(a,b) = gcd(\min(a,b), \max(a,b) - \min(a,b)).$$

The algorithm terminates because the values of a and b decrease in each iteration but remain positive, so a and b must coincide after finitely many steps. When the algorithm terminates, then the values of the variables a and b coincide. We already noted that the gcd of the two variables remains the same throughout the iterations. Since gcd(d, d) = d for any nonzero integer d, we can conclude that the algorithm produces the correct result.

*Example.* Suppose that a = 34 and b = 19. Then the value of the variables a and b in each iteration is

Iteration 0: 
$$(a, b) = (34, 19)$$
  
Iteration 1:  $(a, b) = (19, 15)$   
Iteration 2:  $(a, b) = (15, 4)$   
Iteration 3:  $(a, b) = (4, 11)$   
Iteration 4:  $(a, b) = (4, 7)$   
Iteration 5:  $(a, b) = (4, 3)$   
Iteration 6:  $(a, b) = (3, 1)$   
Iteration 7:  $(a, b) = (1, 2)$   
Iteration 8:  $(a, b) = (1, 1)$ 

Therefore, gcd(34, 19) = gcd(1, 1) = 1.

*Remark.* You might have seen the Euclidean algorithm in another form that involves division with remainder. We can easily derive that version from Algorithm E. Indeed, if we calculate, for instance, gcd(15, 4) with the subtraction algorithm, then we get after three subtractive steps

$$gcd(15,4) = gcd(4,11) = gcd(4,7) = gcd(4,3).$$

Note that these steps repeatedly subtract 4 until we obtain a number that is less than 4. We can comprise these three steps in a single division with remainder step,  $gcd(15, 4) = gcd(4, 15 \mod 4)$ , and arrive at the same result. We will detail the algorithm based on this idea in the next paragraph.

**Division with Remainder.** Recall that the division with remainder of an integer *a* by a nonzero integer *b* yields a **quotient**  $q = \lfloor a/b \rfloor$  and a **remainder**  $a \mod b = a - b \lfloor a/b \rfloor$ .

Algorithm G (GCD with Division and Remainder) Input: a, b positive integers Output: d = gcd(a, b)while  $(b \neq 0)$  do  $(a, b) = (min(a, b), max(a, b) \mod min(a, b));$ return a;

The algorithm terminates after a finite number of iterations, since b is replaced in each iteration by a smaller non-negative integer. We will estimate in the next paragraph how many steps are needed in the worst case.

*Example.* Suppose that a = 34 and b = 19. Then Algorithm G takes the following steps:

| Iteration 0: | (a,b) = (34,19) |
|--------------|-----------------|
| Iteration 1: | (a,b) = (19,15) |
| Iteration 2: | (a,b) = (15,4)  |
| Iteration 3: | (a,b) = (4,3)   |
| Iteration 4: | (a,b) = (3,1)   |
| Iteration 5: | (a,b) = (1,0)   |

**Exercise 1.** If we assume that the input satisfies a > b > 0, then in each iteration of Algorithm G the values of the variables a and b satisfy a > b. Why? How does this allow you to simplify the algorithm?

We show now that the algorithm is correct. We denote by  $\langle a, b \rangle$  the set  $\{ax + by | x, y \in \mathbb{Z}\}$ ; this set is called the **ideal** generated by a and b in the ring of integers. Notice that if  $\langle a, b \rangle$  contains the integers c and d, then  $\langle c, d \rangle$  is a subset of  $\langle a, b \rangle$ .

**Lemma 1.** If  $b \neq 0$ , then  $\langle a, b \rangle = \langle b, a \mod b \rangle$ .

*Proof.* The ideal  $\langle a, b \rangle$  contains the remainder  $r = a \mod b$ , since r = a - qb with  $q = \lfloor a/b \rfloor$ . Thus, if  $b \neq 0$  then the ideal  $\langle b, a \mod b \rangle$  is a subset of  $\langle a, b \rangle$ . On the other hand, a is an element of the ideal  $\langle b, a \mod b \rangle$ , since a = qb + r. Therefore,  $\langle a, b \rangle$  is contained in the ideal  $\langle b, a \mod b \rangle$ . It follows that the ideals  $\langle a, b \rangle$  and  $\langle b, a \mod b \rangle$  are the same.

The lemma shows that the ideal generated by the values of the variables a and b in Algorithm G is an invariant of the loop. Indeed, since  $\langle a, b \rangle = \langle b, a \rangle$ , the previous lemma implies  $\langle a, b \rangle = \langle \min(a, b), \max(a, b) - \min(a, b) \rangle$ . Suppose that the variable a in Algorithm G contains the value g when the

algorithm terminates, then  $\langle a, b \rangle = \langle g, 0 \rangle$ . It follows that a and b are multiples of g, hence g is a common divisor of a and b. On the other hand, g can be expressed in the form g = ax + by. Therefore, each common divisor of a and b divides g, hence g = gcd(a, b). This proves that Algorithm G is correct.

**Analysis.** We analyze the number of iterations that Algorithm G takes in the worst case to compute the greatest common divisor. Recall that the Fibonacci sequence  $(F_k)_{k\geq 0}$  is defined by  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_k = F_{k-1} + F_{k-2}$  for k > 1. We will see that the worst case behavior is attained in the input consists of two consecutive Fibonacci numbers.

**Lemma 2.** If a > b > 0 and Algorithm G performs  $k \ge 1$  iterations before it terminates, then  $a \ge F_{k+2}$  and  $b \ge F_{k+1}$ .

*Proof.* The statement is true for k = 1 iterations because, by assumption,  $b \ge 1 = F_2$  and  $a \ge 2 = F_3$ . Assume that the statement is true when k-1 iterations are made. We will prove that it is true when k iterations are needed. Since k > 0, we have b > 0 and after one iteration the values of (a, b)are substituted with the values  $(b, a \mod b)$ . By our induction hypothesis,  $b \ge F_{k+1}$  and  $a \mod b \ge F_k$ . Since a > b, we have  $\lfloor a/b \rfloor \ge 1$ ; therefore

$$a \ge b + (a - \lfloor a/b \rfloor b) = b + a \mod b.$$

We can conclude that  $a \ge F_{k+2} = F_{k+1} + F_k$ .

**Lemma 3.** Let  $\phi = (1 + \sqrt{5})/2$ . We have  $F_{n+1} \ge \phi^{n-1}$  for  $n \ge 1$ .

*Proof.* We prove this by induction. For n = 1, we have  $F_2 = 1 \ge \phi^0$ . Assume that the claim is true for all  $F_k$  with  $2 \le k \le n$ . We have

$$F_{n+1} = F_n + F_{n-1} \ge \phi^{n-2} + \phi^{n-3} = \phi^{n-2} \left( 1 + \frac{1}{\phi} \right) = \phi^{n-1}.$$

**Theorem 4.** If a > b > 0, then Algorithm G terminates after at most  $1 + \log_{\phi}(b/\gcd(a, b))$  iterations, where  $\phi = (1 + \sqrt{5})/2$ .

*Proof.* Note that Algorithm G takes the same number of steps for input (a, b) and input  $(a/\operatorname{gcd}(a, b), b/\operatorname{gcd}(a, b))$ . Therefore, we can assume without loss

of generality that gcd(a, b) = 1, and it suffices to show that the algorithm terminates after at most  $1 + \log_{\phi} b$  iterations.

Suppose that the algorithm terminates after n iterations. From Lemmas 1 and 2, we know that  $b \ge F_{n+1} \ge \phi^{n-1}$ . Taking logarithms, we get  $\log_{\phi} b \ge n-1$ ; hence,  $n \le 1 + \log_{\phi} b$ . This proves the claim.

**Corollary 5.** Suppose that a and b are positive integers that can be represented by n-bits, that is,  $0 < a, b < 2^n$ . The bit-complexity of Algorithm G to compute gcd(a, b) is at most  $O(n^3)$ .

*Proof.* By Theorem 4, Algorithm G needs O(b) iterations, and each remainder calculation can be done with  $O(n^2)$  bit operations. We obtain the result from  $O(b)O(n^2) = O(n^3)$ .

If Algorithm G is implemented with long-number arithmetic and the numbers are not represented by the same numbers of bits throughout the rounds, then the previous Corollary gives an overly pessimistic estimate of the running time. We can use an amortized analysis to arrive at a better bound of  $O(n^2)$  in that case.

**Diophantine Equations.** Recall that a diophantine equation is an equation in which only integer solutions are allowed. One application of the Euclidean algorithm is to find the solution to one of the simplest diophantine equations. Given integers a, b, and c, the task is to find a pair of integers x and y such that

$$ax + by = c$$

holds. Such an equation may have infinitely many solutions or none.

Let us first derive a non-constructive solution to our problem.

**Lemma 6.** For integers a and b, we have  $\langle a, b \rangle = \langle \gcd(a, b), 0 \rangle$ .

*Proof.* We note that  $\langle a, b \rangle = \langle b, a \rangle = \langle -a, b \rangle$ . Therefore, we can assume that the ideal is generated by nonnegative integers. The claim certainly holds when a = 0 or b = 0, since gcd(a, 0) = a and gcd(0, b) = b. If a and b are positive integers, then the claim follows from our correctness argument of the Euclidean algorithm, see the remarks following Lemma 1.

**Theorem 7.** Suppose that a, b and c are integers. The diophantine equation ax + by = c has an integer solution if and only if gcd(a, b) divides c.

*Proof.* Since gcd(a, b) divides a and b, it must divide ax + by for any integer x and y; thus gcd(a, b) must divide c.

Conversely, it follows from the previous lemma that gcd(a, b) = am + bn for some integers n and m. Hence, if gcd(a, b) divides c, there must exist an integer c' such that

$$c = c' \operatorname{gcd}(a, b) = c'(am + bn) = a(c'm) + b(c'n).$$

This shows that x = c'm and y = c'm is a solution of ax + by = c.

The proof of the previous theorem shows that if we can express gcd(a, b) as an integer linear combination of a and b, then we can solve the diophantine equation ax + by = c. We derive an extended Euclidean algorithm to solve this problem.

**Extension.** Suppose that a > b > 0. Recall that each iteration in the Euclidean algorithm replaces (a, b) by  $(b, a \mod b)$ . We can formulate this as a matrix multiplication:

$$(b, a \mod b) = (a, b) \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$$
, with  $q = \lfloor a/b \rfloor$ .

Suppose that the algorithm terminates after k iterations, then the operations performed on (a, b) can be summarized in matrix notation by

$$(\gcd(a,b),0) = (a,b) \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_2 \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix}.$$

The product of the quotient matrices  $\begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix}$  gives a 2 × 2 matrix such that

$$(\gcd(a,b),0) = (a,b) \begin{pmatrix} r & u \\ s & v \end{pmatrix},$$

and the first column of the resulting matrix gives the desired integers r and s.

The idea of the extended Euclidean algorithm is to keep track of the product of the quotient matrices along with the remainder computation. For example, the Euclidean algorithm computes the greatest common divisor of 15 and 6 by the following swap and remainder steps  $(15, 6) \rightarrow (6, 3) \rightarrow$ 

(3,0). The extended Euclidean algorithm performs these steps in matrix formulation, and records the product of the quotient matrices as follows:

| ( | 1  | 0  | $\begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix}$ | ( 0 | 1   | $\begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix}$ | ( 1 | -2 ) |   |
|---|----|----|---|-----|-----|---|-----|------|---|
|   | 0  | 1  | $\xrightarrow{(1-2)}$                           | 1   | -2  | $\xrightarrow{(1-2)}$                           | -2  | 5    | • |
|   | 15 | 6) |   | 6   | 3 ) |   | 3   | 0 )  |   |

The left column of the last matrix contains integers r = 1, s = -2, and g = 3 so that  $g = ar + bs = 15 \times 1 + 6 \times (-2) = \gcd(15, 6)$ . We use matrices in the formulation of the extended Euclidean algorithm for easier mnemonics:

Algorithm E (Extended GCD Algorithm) Input: a, b positive integers, a > b > 0Output: Integers (g, r, s) such that gcd(a, b) = g = ar + bs.  $\begin{pmatrix} r & u \\ s & v \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix};$ 

 $\begin{pmatrix} r & u \\ s & v \\ a & b \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{pmatrix};$ while  $(b \neq 0)$  do  $q := \lfloor a/b \rfloor;$   $\begin{pmatrix} r & u \\ s & v \\ a & b \end{pmatrix} := \begin{pmatrix} r & u \\ s & v \\ a & b \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 - q \end{pmatrix};$ od;
return (a, r, s)

**Exercise 2.** Derive an algorithm that finds a integer solution to ax + by = c given integers a, b, and c; if no solutions exist, then the algorithm should report that. Is it possible to derive all solutions from the solution that you have given?

**Continued Fractions.** Algorithm G with input (a, b), where a > b, is closely related to the continued fraction of the rational number a/b. If we represent a in the form a = bq + r, with  $0 \le r < b$ , then we can write the fraction a/b in the form

$$\frac{a}{b} = \frac{bq+r}{b} = q + \frac{r}{b} = q + \frac{1}{b/r}.$$

We can repeat this process for b/r since b > r. The process eventually terminates because b and r are respectively smaller than a and b.

*Example.* We develop the continued fraction for 34/19 to illustrate this idea. We have

$$\frac{34}{19} = 1 + \frac{15}{19} = 1 + \frac{1}{19/15} = 1 + \frac{1}{1 + \frac{4}{15}} = \dots = 1 + \frac{1}{1 + \frac{1}{3 + \frac{1}{1 + \frac{1}{1}}}}$$

The relation to the Euclidean algorithm is immediately apparent. Indeed, compare the continued fraction with the quotient and remainder sequence  $34 = \mathbf{1} \times 19 + 15$ ,  $19 = \mathbf{1} \times 15 + 4$ ,  $15 = \mathbf{3} \times 4 + 3$ , and  $4 = \mathbf{1} \times 3 + \mathbf{1}$ .

The example clearly illustrates that the continued fraction notation is somewhat cumbersome and eventually hard to read. We define

$$//x_1, x_2, \dots, x_n // = 1/(x_1 + 1/(x_2 + 1/(\dots (x_{n-1} + 1/x_n) \dots))).$$

Every real number x in the range  $0 \le x < 1$  has a regular continued fraction that is defined the following way: Define  $x_0 = x$ , and for all  $n \ge 0$  such that  $x_n \ne 0$ , let

$$a_{n+1} = \lfloor 1/x_n \rfloor, \qquad x_{n+1} = 1/x_n - a_{n+1}.$$
 (1)

If  $x_n = 0$ , then  $a_{n+1}$  and  $x_{n+1}$  are not defined, and the regular continued fraction for x is  $//a_1, \ldots, a_n//$ . By definition,

$$x = x_0 = \frac{1}{a_1 + x_1} = \frac{1}{a_1 + 1/(a_2 + x_2)} = \cdots$$

Therefore,  $x = //a_1, \ldots, a_n + x_n //$ . It is not difficult to show that  $//a_1, \ldots, a_n //$  is an extremely good approximation to x unless n is small. For instance,

$$\pi = 3 + //7, 15, 1, 292, 1, 1, 1, 2, 1, 3, \dots //$$

When x is a rational number, then the regular continued fraction corresponds to Euclid's algorithm. In fact, Gauss preferred the continued fraction representation over the classical Euclidean algorithm and he even ignored the Euclidean algorithm in his *Disquisitiones*.

Suppose that x = u/v, where u and v are integers such that u > v > 0. Let us see how we can recover the Euclidean algorithm from the regular continued fraction algorithm. Define  $u_0 = u$  and  $v_0 = v$ . We have  $x_0 = v_0/u_0$ . Suppose that  $x_n = v_n/u_n$ , then equation (1) yields

$$a_{n+1} = \lfloor u_n / v_n \rfloor$$
  $x_{n+1} = u_n / v_n - a_{n+1} = (u_n \mod v_n) / v_n.$ 

Let us define

$$u_{n+1} = v_n, \qquad v_{n+1} = u_n \bmod v_n.$$

Then  $x_n = v_n/u_n$  holds for all steps in the regular continued fraction algorithm. The main point is that the two algorithms are basically two sides of the same coin. In fact, the continued fraction algorithm is often used in more detailed analysis of the Euclidean algorithm.

**Chinese Remainder Theorem.** We conclude with another classical application of the Euclidean algorithm, the classical Chinese Remainder Theorem.

Suppose we are to calculate modulo a number n, where n is the product of two coprime integers, n = ab with  $a \perp b$ . Then is is often advantageous to perform the calculations in parallel (modulo a) and (modulo b). In other words, we represent the numbers from  $0, \ldots, n-1$  by their remainders. The Chinese Remainder Theorem allows us to reconstruct the result as if the computations were done modulo n.

*Example.* Let  $n = 3 \times 5 = 15$ . The numbers  $0 \le x < 15$  are respresented by their remainder pairs  $(x \mod 3, x \mod 5)$  as follows:

| x           | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|-------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| $x \mod 3$  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1  | 2  | 0  | 1  | 2  |
| $x \bmod 5$ | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 | 0  | 1  | 2  | 3  | 4  |

You might notice that the pairs  $(x \mod 3, x \mod 5)$  do not repeat until x reaches lcm(3,5) = 15. Therefore, the map  $x \mapsto (x \mod 3, x \mod 5)$  from  $\{0, \ldots, 14\}$  onto  $\{0, 1, 2\} \times \{0, 1, 2, 3, 4\}$  is a bijection (one-on-one and onto).

Let us look at the problem more generally. A direct generalization of the above property of remainders yields the following lemma:

**Lemma 8.** Suppose that a and b are integers such that gcd(a,b) = 1. We have  $y \equiv x \mod a$  and  $y \equiv x \mod b$  if and only if  $y \equiv x \mod ab$ .

The question is how we can reconstruct x from its remainder pair representation ( $x \mod a, x \mod b$ ). Since a and b are coprime, we can determine with the help of the extended Euclidean algorithm some integers k and  $\ell$ such that  $ak + b\ell = 1$ . It follows that

 $ak + b\ell \equiv b\ell \equiv 1 \mod a,$ <br/> $ak + b\ell \equiv ak \equiv 1 \mod b.$ 

Suppose now that we are given the residues  $x_a$  and  $x_b$  of x modulo a and b:

$$x \equiv x_a \mod a$$
$$x \equiv x_b \mod b$$

We claim that the map

$$(x_a, x_b) \mapsto x_a b\ell + x_b ak \mod n$$

reconstructs x. Indeed, by Lemma 8 it suffices to verify that  $y = x_a b\ell + x_b ak$  is mapped to the same remainders modulo a and b as x:

$$y = x_a b\ell + x_b ak \equiv x_a b\ell \equiv x_a \mod a,$$
  
$$y = x_a b\ell + x_b ak \equiv x_b ak \equiv x_b \mod b,$$

so  $y \equiv x \mod ab$ .

The main application of the Chinese Remainder Theorem is that arithmetic operations such as addition and multiplication modulo n = ab can be performed modulo a and modulo b. The advantage is that the calculations are performed with numbers of a smaller range. The moduli are often chosen to be two numbers close to the wordlength of the machine, so that multiprecision arithmetic is avoided.

*Example.* Let a = 34, b = 19; thus,  $n = 34 \times 19 = 646$ . The extended Euclidean algorithm shows that  $34 \times (-5) + 19 \times 9 = 1$ . If we want to multiply x = 40 and y = 20 modulo n, then we can do this by performing the calculations modulo 34 and 19 and reconstruct the result.

| x = 40                  | $\equiv$ | $6 \mod 34$  | x = 40 | $\equiv$ | $2 \mod 19$  |
|-------------------------|----------|--------------|--------|----------|--------------|
| y = 20                  | ≡        | $20 \mod 34$ | y = 20 | $\equiv$ | $1 \bmod 19$ |
| $xy \equiv 6 \times 20$ | Ξ        | $18 \mod 34$ | xy     | $\equiv$ | $2 \mod 19$  |

We obtain the result modulo n = ab = 646 by the calculation

 $18 \times 19 \times 9 + 2 \times 34 \times (-5) \equiv 3078 - 340 \equiv 154 \mod 646$ 

This coincides with  $20 \times 40 = 800 \equiv 154 \mod 646$ , as it should be.

**Further Reading.** It is interesting to see the manifold applications of Euclid's algorithm in number theory. John Stillwell's book ["Elements of Number Theory", Springer-Verlag, 2003] gives a splendid exposition. We have merely scratched the surface in our analysis of Euclid's algorithm. Knuth ["The Art of Computer Programming", Volume 2, 3rd edition, Addison Wesley, 1998, Section 4.5.3] shows the relation to continued fractions and discusses average case analysis.