

Lecture 12: GCD, Extended Euclid, and Running Time

Lecturer and scribe: Karthik Gajulapalli

We will start with a simple observation about common divisors, turn it into an efficient recursive algorithm, and then recover more information from the same recursion. The final algorithm finds both the gcd and integers that express it as a linear combination of the inputs.

1 The Greatest Common Divisor

Recall that $d \mid a$ means $a = dk$ for some integer k . For integers a, b that are not both zero, the **greatest common divisor**, written $\gcd(a, b)$, is the largest positive integer dividing both. For example, the positive common divisors of 18 and 30 are 1, 2, 3, 6, so $\gcd(18, 30) = 6$. Other examples are

$$\gcd(12, 20) = 4, \quad \gcd(8, 15) = 1, \quad \gcd(7, 21) = 7.$$

Two integers are **coprime**, or **relatively prime**, if their gcd is 1. The numbers themselves need not be prime: 8 and 15 are both composite, but they are coprime.

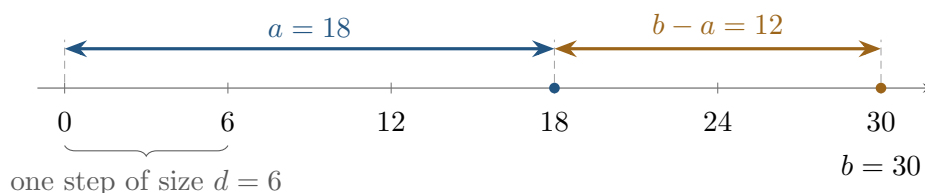
Signs do not affect the gcd: $\gcd(a, b) = \gcd(|a|, |b|)$. Also, $\gcd(a, 0) = |a|$ for $a \neq 0$, because every integer divides zero. We leave $\gcd(0, 0)$ undefined. Our algorithms use nonnegative inputs, not both zero; signed inputs can first be replaced by their absolute values.

2 Euclid's Observation

Suppose $0 < a \leq b$. Then

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

Think of a and b as points on a number line. Their distance is $b - a$. If a step size d divides both positions, it also divides the distance between them.



Proof. The pairs (a, b) and $(a, b - a)$ have exactly the same common divisors. If $d \mid a$ and $d \mid b$, then $d \mid (b - a)$. Conversely, if $d \mid a$ and $d \mid (b - a)$, then $d \mid (a + (b - a)) = b$. Therefore their greatest common divisors agree. $\square$

2.1 A First Recursive Algorithm

Subtract the smaller positive number from the larger. Stop when an input becomes zero.

```
def gcd_subtract(a, b):      # a, b >= 0; not both zero
    if a == 0:
        return b
    if b == 0:
        return a
    if a <= b:
        return gcd_subtract(a, b - a)
    return gcd_subtract(a - b, b)
```

The recursive promise is simple: `gcd_subtract(a, b)` returns $\gcd(a, b)$. Euclid's observation justifies the call in either branch. Each subtraction strictly decreases the nonnegative integer $a + b$, so the recursion eventually reaches a base case.

2.2 Example: (18,30)

Follow the recursive calls:

$$(18, 30) \longrightarrow (18, 12) \longrightarrow (6, 12) \longrightarrow (6, 6) \longrightarrow (6, 0).$$

The base case returns 6. Each earlier call returns that same value to its caller, so the original call returns $\gcd(18, 30) = 6$. There are four subtraction steps in this example.

2.3 Why Repeated Subtraction Can Be Slow

If a is small compared with b , we subtract it over and over. For instance,

$$(1, 1000) \longrightarrow (1, 999) \longrightarrow (1, 998) \longrightarrow \cdots \longrightarrow (1, 0)$$

requires 1000 subtractions. More generally, $(1, N)$ takes N subtraction steps, even though the answer is immediately 1.

One subtraction may make very little progress. We should combine many identical subtractions into one division with remainder.

For example, starting with $(46, 240)$, five subtractions of 46 from the larger number give

$$240 \rightarrow 194 \rightarrow 148 \rightarrow 102 \rightarrow 56 \rightarrow 10.$$

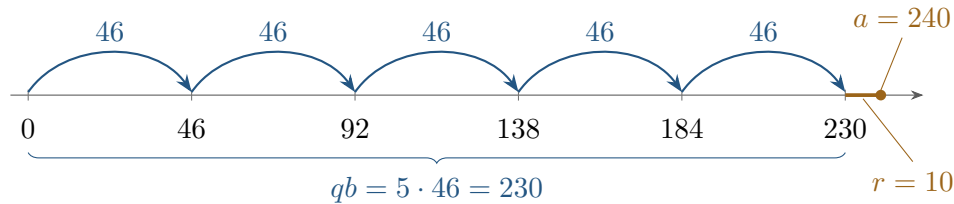
The single equation $240 = 5 \cdot 46 + 10$ records all five at once.

3 From Subtraction to Division

Now name the number being divided a and the divisor $b > 0$. Division with remainder gives

$$a = qb + r, \quad 0 \leq r < b,$$

where $q = \lfloor a/b \rfloor$ and $r = a \bmod b$. The remainder is what is left after removing q full copies of b .



Lemma 1 (The Euclidean step). *If $a = qb + r$, then $\gcd(a, b) = \gcd(b, r)$.*

Proof. If $d \mid a$ and $d \mid b$, then $d \mid (a - qb) = r$. Conversely, if $d \mid b$ and $d \mid r$, then $d \mid (qb + r) = a$. The two pairs have the same common divisors. $\square$

3.1 Euclid's Recursive Algorithm

```
def gcd(a, b):           # a, b >= 0; not both zero
    if b == 0:
        return a
    return gcd(b, a % b)
```

Here $a \% b$ is the remainder. The lemma preserves the answer, and the second input strictly decreases until it is zero. If $a < b$, the first call is simply $\gcd(b, a)$, since $a \bmod b = a$.

3.2 Example: (240,46)

$$\begin{aligned} 240 &= 5 \cdot 46 + 10, \\ 46 &= 4 \cdot 10 + 6, \\ 10 &= 1 \cdot 6 + 4, \\ 6 &= 1 \cdot 4 + 2, \\ 4 &= 2 \cdot 2 + 0. \end{aligned}$$

Thus the calls are

$$(240, 46) \rightarrow (46, 10) \rightarrow (10, 6) \rightarrow (6, 4) \rightarrow (4, 2) \rightarrow (2, 0).$$

The base case returns 2, the last nonzero remainder. There are five divisions, including the final division with remainder zero.

4 Bézout's Identity

Claim 1 (Bézout's identity). *For integers a, b that are not both zero, there exist integers x, y such that*

$$ax + by = \gcd(a, b).$$

The expression $ax + by$ is an **integer linear combination** of a and b . The integers x, y are called **Bézout coefficients**; either coefficient may be negative or zero. For example,

$$6 = 2 \cdot 18 - 1 \cdot 30, \quad 1 = 2 \cdot 8 - 1 \cdot 15.$$

In particular, coprime integers can be combined to produce 1. The coefficients need not be unique: we also have $6 = (-3) \cdot 18 + 2 \cdot 30$.

4.1 Proof by the Well-Ordering Principle

Without loss of generality, assume $a, b > 0$: the zero cases are immediate, and signs can be absorbed into the coefficients. Consider all integer linear combinations and their positive members:

$$S = \{ax + by : x, y \in \mathbb{Z}\}, \quad S^+ = \{s \in S : s > 0\}.$$

Taking $x = y = 1$ gives $a + b > 0$, so S^+ is nonempty. By the well-ordering principle, S^+ has a **smallest element**. Choose

$$\boxed{d = \min S^+}, \quad d = ax_0 + by_0 > 0.$$

The **minimality** of d will force it to divide both inputs.

First, use minimality to show that d divides both inputs. Divide a by the positive integer d :

$$a = qd + r, \quad 0 \leq r < d.$$

The remainder is another integer linear combination:

$$r = a - qd = a - q(ax_0 + by_0) = a(1 - qx_0) + b(-qy_0) \in S.$$

If $r > 0$, then $r \in S^+$ and $r < d$, contradicting that d is the **smallest element of S^+** . Thus $r = 0$, so $d \mid a$. The same argument, dividing b by d , shows $d \mid b$.

Second, no positive common divisor is larger. If $c \mid a$ and $c \mid b$, then $c \mid (ax_0 + by_0) = d$. For positive c , this implies $c \leq d$. Hence d is the greatest common divisor, and $ax_0 + by_0 = \gcd(a, b)$. $\square$

This proof also shows that the gcd is the *smallest positive integer linear combination* of the two inputs. It uses division with remainder, but does not need the Euclidean algorithm.

5 Recovering Bézout Coefficients by Back-Substitution

Euclid's algorithm already contains the coefficients we need. Rearrange each division as $r = a - qb$. Start with the equation producing the last nonzero remainder, then substitute earlier equations until only the original inputs remain. This is **back-substitution**.

5.1 Example: (18,30)

Euclid first swaps the inputs, since $18 = 0 \cdot 30 + 18$, and then gives

$$30 = 1 \cdot 18 + 12, \quad 18 = 1 \cdot 12 + 6, \quad 12 = 2 \cdot 6 + 0.$$

Working backward,

$$6 = 18 - 12 = 18 - (30 - 18) = 2 \cdot 18 - 30.$$

Thus $(x, y) = (2, -1)$, in the original input order $(18, 30)$.

5.2 Example: (240,46)

Use the divisions from Section 3, starting with $6 = 4 + 2$:

$$\begin{aligned} 2 &= 6 - 4 \\ &= 6 - (10 - 6) = 2 \cdot 6 - 10 \\ &= 2(46 - 4 \cdot 10) - 10 = 2 \cdot 46 - 9 \cdot 10 \\ &= 2 \cdot 46 - 9(240 - 5 \cdot 46) \\ &= -9 \cdot 240 + 47 \cdot 46. \end{aligned}$$

Thus $(x, y) = (-9, 47)$. Check: $-2160 + 2162 = 2$.

5.3 Example: (27,5)

The divisions are

$$27 = 5 \cdot 5 + 2, \quad 5 = 2 \cdot 2 + 1, \quad 2 = 2 \cdot 1 + 0.$$

Working backward,

$$1 = 5 - 2 \cdot 2 = 5 - 2(27 - 5 \cdot 5) = -2 \cdot 27 + 11 \cdot 5.$$

Thus $(x, y) = (-2, 11)$. Check: $-54 + 55 = 1$.

In each example, the divisions find the gcd on the way down. Back-substitution recovers its coefficients on the way back up. We can build this second job directly into a recursive algorithm.

6 The Extended Euclidean Algorithm

The **recursive promise** is

$$\boxed{\text{ExtendedGCD}(a, b) = (d, x, y), \quad d = \gcd(a, b), \quad ax + by = d.}$$

The inputs are a, b ; the outputs are d, x, y . In every call, x is the coefficient of *that call's first input*, and y is the coefficient of *that call's second input*.

6.1 Deriving the Recursive Return Values

If $b = 0$, then $d = a$ and $a \cdot 1 + 0 \cdot 0 = a$, so return $(a, 1, 0)$. Otherwise, write $a = qb + r$ and recursively call the algorithm on (b, r) . Suppose it returns (d, u, v) . Its promise says

$$d = \underbrace{b}_{\text{child's first input}} u + \underbrace{r}_{\text{child's second input}} v.$$

These are coefficients of b, r . To get coefficients of a, b , substitute $r = a - qb$ and collect terms:

$$\begin{aligned} d &= bu + rv = bu + (a - qb)v \\ &= a \underbrace{v}_x + b \underbrace{(u - qv)}_y. \end{aligned}$$

Therefore the current call must return

$$\boxed{(d, x, y) = (d, v, u - qv).}$$

```
def extended_gcd(a, b):    # a, b >= 0; not both zero
    if b == 0:
        return (a, 1, 0)

    q = a // b             # integer quotient
    r = a % b              # remainder
    d, u, v = extended_gcd(b, r)
    x = v
    y = u - q * v
    return (d, x, y)
```

Each call keeps its own a, b, q . The child returns coefficients for (b, r) , and the parent translates them into coefficients for (a, b) . The quotient in $u - qv$ is the *parent's* quotient.

Why this proves correctness. The base case satisfies the promise. Assuming the child satisfies its promise, the algebra above establishes the parent's identity, and $\gcd(a, b) = \gcd(b, r)$ establishes its gcd. Since $r < b$, strong induction on the second input proves the promise for every call.

6.2 Watching the Recursive Calls Return: (240,46)

The calls go down the same chain as ordinary Euclid:

$$(240, 46) \rightarrow (46, 10) \rightarrow (10, 6) \rightarrow (6, 4) \rightarrow (4, 2) \rightarrow (2, 0).$$

The returns happen in the opposite order. Read this table from top to bottom; each identity uses the inputs of the call in that row.

Call (a, b)	Returns (d, x, y)	Meaning of the return value
$(2, 0)$	$(2, 1, 0)$	$2 = 2 \cdot 1 + 0 \cdot 0$
$(4, 2)$	$(2, 0, 1)$	$2 = 4 \cdot 0 + 2 \cdot 1$
$(6, 4)$	$(2, 1, -1)$	$2 = 6 \cdot 1 + 4 \cdot (-1)$
$(10, 6)$	$(2, -1, 2)$	$2 = 10 \cdot (-1) + 6 \cdot 2$
$(46, 10)$	$(2, 2, -9)$	$2 = 46 \cdot 2 + 10 \cdot (-9)$
$(240, 46)$	$(2, -9, 47)$	$2 = 240 \cdot (-9) + 46 \cdot 47$

For the final row, the child returns $(d, u, v) = (2, 2, -9)$ and the parent's quotient is $q = 5$. Therefore

$$x = v = -9, \quad y = u - qv = 2 - 5(-9) = 47.$$

The gcd stays 2 throughout the return chain. The coefficients change because each call asks for a combination of a different pair of inputs.

7 Running Time: Halving Every Two Steps

We count one division with remainder as one Euclidean step, including the final division whose remainder is zero. Let $n = \max(a, b)$ for the original inputs. Arrange the positive inputs so that $a \geq b > 0$; otherwise, allow one initial swap step. Zero inputs are base cases.

Write $a = qb + r$, where $0 \leq r < b$. Since $q \geq 1$ and $b > r$,

$$a = qb + r \geq b + r > 2r, \quad \text{so } r < a/2.$$

Now look at two successive calls:

$$(a, b) \longrightarrow (b, r) \longrightarrow (r, b \bmod r).$$

After two steps the maximum is $r < a/2$. If $r = 0$, we stop earlier. Thus every two steps more than halve the current maximum.

After k pairs of steps, if the algorithm is still running, its maximum is less than $n/2^k$. A positive integer cannot be less than 1, so there can be only about $\log_2 n$ such pairs. An upper bound on the number of steps is therefore

$$2 \log_2 n + O(1) = O(\log n).$$

8 A Fibonacci Sketch of the Running Time

There is another way to see the logarithmic bound. The slowest progress happens when the quotients are as small as possible, so successive remainders stay large.

Let $F_1 = F_2 = 1$ and $F_{j+1} = F_j + F_{j-1}$. For consecutive Fibonacci numbers,

$$F_{j+1} = 1 \cdot F_j + F_{j-1}.$$

Euclid therefore moves down the Fibonacci sequence, with quotient 1 until the final division $2 = 2 \cdot 1 + 0$. For example,

$$(13, 8) \rightarrow (8, 5) \rightarrow (5, 3) \rightarrow (3, 2) \rightarrow (2, 1) \rightarrow (1, 0).$$

These are five divisions, and each step moves only one position down the sequence.

Why this also gives an upper bound: a sketch. For strictly decreasing positive remainders, each division has the form

$$r_{i-1} = q_i r_i + r_{i+1} \geq r_i + r_{i+1}.$$

Read this backward. The final positive remainder is $d \geq 1$, and the preceding remainder is at least $2d$. Every earlier remainder is at least the sum of the next two. Consequently, the smallest possible positive values, read backward, grow like

$$1, 2, 3, 5, 8, 13, \dots$$

A long run therefore requires large initial inputs: a run of t divisions, starting with $a > b > 0$, forces $a \geq F_{t+2}$. Since $F_j = \Theta(\varphi^j)$, where $\varphi = (1 + \sqrt{5})/2 > 1$, we get $t = O(\log n)$.

Conversely, consecutive Fibonacci inputs attain this slow growth, so some inputs require $\Theta(\log n)$ divisions. The logarithmic bound is tight in the worst case, although many inputs finish much faster: if $b \mid a$, one division suffices.