1 Die hard, once and for all

You are standing next to a water source. You have two empty jugs: A 3 litre jug and a 5 litre jug with no marks on them. You must fill the larger jug with precisely 4 litres of water. Can you do it?

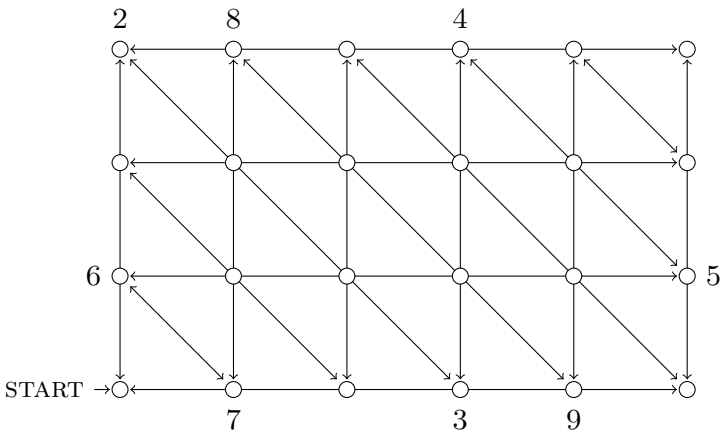
This scenario is straight out of the 1995 classic "Die Hard 3: With a Vengeance" featuring Bruce Willis and Samuel L. Jackson. Should they fail to complete this task within 5 minutes, a bomb goes off and New York City is obliterated. In the nick of time, Bruce and Samuel come up with a solution:

large jug	small jug	action
—	3ℓ	fill up small jug from source
3ℓ	—	transfer water into large jug
3ℓ	3ℓ	fill up small jug
5ℓ	1ℓ	top up large jug from small jug
—	1ℓ	spill large jug
1ℓ	—	transfer water into large jug
1ℓ	3ℓ	fill up small jug
4ℓ	—	done!

The rumour is that in Die Hard 4, Bruce and Samuel will be given a 21 litre and a 26 litre jug; in Die Hard 5, a 899 litre and 1,147 litre jug; and in Die Hard 6: "Die hard, once and for all", a 6 litre and 9 litre jug. What should they do?

The Die Hard state machine The water jug game can be described by a state machine. The states are pairs of numbers (x, y) describing the amount of water in the large and smaller jug, respectively. The transitions represent the allowed pourings. The initial state is $(0, 0)$, and we are interested in reaching a state of the form $(4, y)$.

The Die Hard 3 state machine is small enough that we can represent it with a diagram. The sequence of transitions START $\to 2 \to 3 \to \cdots \to 9$ achieves the desired objective.



In general, the jugs can be large and we need to describe the state machine mathematically. If a and b are the jug capacities the possible states are the pairs

$$(x, y) \text{ such that } 0 \leq x \leq a \text{ and } 0 \leq y \leq b.$$

There are at most six transitions out of each state (represented by the horizontal, vertical, and diagonal arrows in the picture) given by the pouring rules:

1. **Spill a jug:** The spilled out jug now contains 0 litres, while the contents of the other jug stay the same. Symbolically,

$$(x, y) \to (0, y) \tag{T1}$$
$$(x, y) \to (x, 0) \tag{T2}$$

2. **Fill up a jug from the source:** The filled up jug now contains as much water as its capacity. The contents of the other jug stay the same. Symbolically,

$$(x, y) \to (a, y) \tag{T3}$$
$$(x, y) \to (x, b) \tag{T4}$$

3. **Transfer water between jugs:** Water can be poured from one jug into the other until one of them becomes full or empty. These transitions are a bit trickier to write down, but the effort will pay off later on.

   If water is poured from the first into the second jug then two things can happen. If the total amount of water does not exceed the capacity b of the second jug, in which case the second one will contain $x + y$ litres. Otherwise, the second jug fills up to its capacity b, and $x + y - b$ litres of water remain in the first jug:

$$(x, y) \to \begin{cases} (0, x + y), & \text{if } x + y \leq b, \\ (x + y - b, b), & \text{if } x + y > b. \end{cases} \tag{T5}$$

   We have analogous transitions for pouring water from the second jug into the first one:

$$(x, y) \to \begin{cases} (x + y, 0), & \text{if } x + y \leq a, \\ (a, x + y - a), & \text{if } x + y > a. \end{cases} \tag{T6}$$

We can now formulate the water jug game with a v litre target question mathematically: Is there a sequence of transitions that reaches a state of the type (x, v) or (v, y)? If you play a bit more with the Die Hard 3 state machine, you will notice that this is possible for any target v (as long as it is an integer between 0 and 5). But what happens in the other movies?

If you repeat the experiment with the Die Hard 6 state machine you notice something different happens: No matter what you do, the amount of water in either jug is always 3, 6, or 9 litres. To describe this behaviour we introduce the concept of integer combination.

Integer combinations

A number c is an *integer combination* of a and b if there exist integers s and t such that $c = s \cdot a + t \cdot b$. For example,

- 2 is an integer combination of 4 and 6 because $2 = (-1) \cdot 4 + 1 \cdot 6$.

- 1 is an integer combination of 3 and 5 because $1 = 2 \cdot 3 + (-1) \cdot 5$.
- 2 is an integer combination of 2 and 6 because $2 = 1 \cdot 2 + 0 \cdot 6$.

We can now state an invariant of the Die Hard state machines.

Invariant 1. The amount of water in each jug is always an integer combination of a and b.

To prove this invariant, the following property of integer combinations will be useful.

Lemma 2. *If x and y are integer combinations of a and b, then any integer combination of x and y is also an integer combination of a and b.*

For example, 6 and 9 are both integer combinations of 12 and 15:

$$6 = 3 \cdot 12 + -2 \cdot 15$$
$$9 = 2 \cdot 12 - 15$$

and 3 is an integer combination of 6 and 9:

$$3 = -6 + 9.$$

By combining the equations we can write 3 as an integer combination of 12 and 15:

$$3 = -6 + 9 = -(3 \cdot 12 - 2 \cdot 15) + (2 \cdot 12 - 15) = -12 + 15.$$

We can generalize this type of reasoning into a proof of Lemma 2.

Proof of Lemma 2. If x and y are integer combinations of a and b, we can write $x = s \cdot a + t \cdot b$ and $y = s' \cdot a + t' \cdot b$ for some integers s, t, s', t'. Then any integer combination of x and y has the form

$$p \cdot x + q \cdot y = p \cdot (sa + tb) + q \cdot (s'a + t'b) = (ps + qs') \cdot a + (pt + qt') \cdot b$$

so it is an integer combination of a and b. □

We will now apply Lemma 2 to prove the invariant.

Proof of Invariant 1. We will prove that the invariant holds in the initial state and that it is preserved by the transitions. It then follows by induction that the invariant holds after an arbitrary number of steps.

Initial state: Initially, each jug has 0 liters of water and $0 = 0 \cdot a + 0 \cdot b$.

Transitions: Assume the invariant holds in state (x, y). namely both x and y are integer combinations of a and b. We show that this remains true after any possible transition $(x, y) \to (x', y')$. In all the transitions T1-T6, both of x' and y' are some integer combination of x, y, a, b. By Lemma 2, x' and y' are therefore integer combinations of a and b. □

Greatest common divisors

We say d *divides* a if $a = kd$ for some integer k. For example, 2 divides 4 and -6 but 2 does not divide 5; 4 does not divide 2; and every number divides zero.

Lemma 3. *Every common divisor of a and b also divides all integer combinations of a and b.*

Proof. Notice that d divides a number a *if and only if* a is an integer combination of d and zero.

If d is a common divisor of a and b then a and b are integer combinations of d and zero. By Lemma 2 so is any integer combination of a and b, so d divides this integer combination. □

The *greatest common divisor (GCD)* of two integers a, b is the largest integer d so that d divides a and d divides b. We write $\gcd(a, b)$ for the GCD of a and b. For example, $\gcd(2, 6) = 2$, $\gcd(4, 6) = 2$, $\gcd(3, 5) = 1$.

Combining Invariant 1 and Lemma 3, we obtain a useful corollary:

Corollary 4. $\gcd(a, b)$ *always divides the amount of water in each jug.*

In particular, a 6 litre jug and a 9 litre jug can never measure 4 litres of water:

Corollary 5. *In Die Hard 6, Bruce dies.*

2 Euclid's algorithm

Euclid's algorithm is a procedure for calculating the GCD of two positive integers. It was invented by the Euclideans around 3,000 years ago and it still the fastest procedure for calculating GCDs. To explain Euclid's algorithm, we need to recall division with remainder.

Theorem 6. *Let n and d be integers with $d > 0$. There exists unique integers q and r such that*

$$n = q \cdot d + r \quad \textit{and} \quad 0 \leq r < d.$$

For example, if $n = 13$ and $d = 3$, we can write $13 = 4 \cdot 3 + 1$. Moreover, $q = 4$ and $r = 1$ are unique assuming r is in the range $0 \leq r < d$.

The numbers q and r can be calculated by the usual "division with remainder rule" that you learned in school.

Proof. First, we show existence: Let q be the largest integer such that $qd \leq n$. That means $(q + 1)d > n$. Then $0 \leq n - qd < d$. Set $r = n - qd$.

Now we show uniqueness: Suppose we can write n in the desired form in two different ways:

$$n = qd + r, 0 \leq r < d$$
$$n = q'd + r', 0 \leq r' < d.$$

Then $qd + r = q'd + r'$, so $(q - q')d = r' - r$. Therefore $r' - r$ divides d. Since $0 \leq r, r' < d$ we must have $-d < r' - r < d$. The only number in this range that divides d is zero, so $r' - r = 0$ and $q' - q = 0$. It follows that $q' = q$ and $r' = r$, so the two representations are the same. □

Euclid's algorithm is a *recursive* algorithm for calculating the GCD of positive integers.

Euclid's algorithm $E(n, d)$, where n, d are integers such that $n \geq d \geq 0$.
If $d = 0$, return n.
Otherwise, write $n = qd + r$ where $0 \leq r < d$. Return $E(d, r)$.

Let's apply Euclid's algorithm to calculate the GCD of 1147 and 899:

$$\begin{aligned}
E(1147, 899) &= E(899, 248) && \text{because } 1147 = 1 \cdot 899 + 248 \\
&= E(248, 155) && \text{because } 899 = 3 \cdot 248 + 155 \\
&= E(155, 93) && \text{because } 248 = 1 \cdot 155 + 93 \\
&= E(93, 62) && \text{because } 155 = 1 \cdot 93 + 62 \\
&= E(62, 31) && \text{because } 93 = 1 \cdot 62 + 31 \\
&= E(31, 0) && \text{because } 62 = 2 \cdot 31 + 0 \\
&= 31.
\end{aligned}$$

How can we be sure that Euclid's algorithm indeed outputs the GCD of n and d? This is a theorem that we will have to prove.

To analyze Euclid's algorithm we can model it as a state machine. The states are pairs of integers. The initial state is the input to the algorithm. The transitions have the form $(n, d) \to (d, r)$ where r is the remainder of dividing n by d assuming $d \geq 0$.

Invariant 7. The gcd of the state stays the same throughout the execution of Euclid's algorithm.

Proof. Consider any transition $(n, d) \to (d, r)$. Recall that n is of the form $q \cdot d + r$. By Corollary 3, every common divisor of d and r also divides their integer combination $n = qd + r$, so it is a common divisor of n and d.

By Lemma 3 again, every common divisor of n and d also divides their integer combination $r = n - q \cdot d$, so it is a common divisor of d and r.

It follows that the common divisors of n and d are the same as those of d and r, so the gcd of both pairs must be the same. □

Invariant 7 shows that Euclid's algorithm preserves the gcd of its inputs throughout its execution. Is this a good enough reason to conclude that the algorithm is correct? Not really, because for all we know the algorithm may end up running forever! We must also argue that the algorithm eventually terminates.

Theorem 8. *For all integers $n \geq d \geq 0$, $E(n, d)$ terminates and outputs* $\gcd(n, d)$.

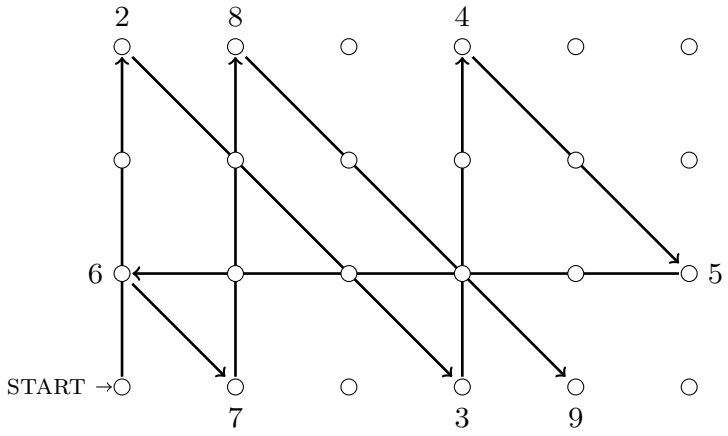
Proof. For termination, we observe that the value of d decreases in each transition but remains non-negative because d is replaced by the remainder r which is always smaller than d. Therefore the algorithm must terminate after a finite number of steps.

The algorithm terminates when it reaches a state of the form $(n, 0)$ in which case it outputs $n = \gcd(n, 0)$. By Invariant 7, its output must equal the gcd of its inputs. □

3 How to be a star in any Die Hard movie

Corollary 4 says that if the amount v is *not* a multiple of the gcd of the jug capacities a and b, then Bruce dies for sure. If it is not, can he actually survive?

To answer this question, let's look at what happened in Die Hard 3 again. Notice the curious zigzagging pattern in the transitions taken by our state machine:



In words, we keep zigzagging up-diagonal-up-diagonal until we hit the right boundary. Then we move all the way left, continue along the diagonal, and keep going until we get lucky. We can rephrase this strategy as a candidate algorithm:

ALGORITHM DieHard
INPUTS: jug capacities a, b and target v.
1 Repeat until jug A contains v litres:
2 Fill up jug B.
3 Pour as much as possible from jug B into jug A.
4 If jug A is full,
5 Spill jug A.
6 Pour out the remaining contents of jug B into jug A.

How do we know that this algorithm works? Notice how in a single iteration of the loop 2-6, jug B is filled up once and spilled out once. Since all the water from jug B ends up being poured into jug A, after the s-th iteration of the loop jug A has received $s \cdot b$ litres from jug B. Owing to step 5, some amount has been poured out of jug A. We don't know how much this is, but it is always an integer multiple of its capacity a. In summary,

After s iterations, jug A contains $s \cdot b - t \cdot a$ litres for some nonnegative integer t.

Let's see how this checks out with our Die Hard 3 scenario:

iteration	transitions	water in large jug
1	START $\to 2 \to 3$	$3 = 1 \cdot 3 - 0 \cdot 5$
2	$3 \to 4 \to 5 \to 6 \to 7$	$1 = 2 \cdot 3 - 1 \cdot 5$
3	$7 \to 8 \to 9$	$4 = 3 \cdot 3 - 1 \cdot 5$

Notice that the number t is uniquely determined by s, a, and b in every iteration. If it was any smaller the large jug would overfill, and it was any smaller the amount of water in it would be negative! We can summarize our reasoning in the following lemma.

Lemma 9. *If $0 < v < a$ and $v = s \cdot b - t \cdot a$ for some non-negative integers s, t, then algorithm* DieHard *has terminated within s iterations.*

Proof. If algorithm DieHard goes on for s iterations, $s \cdot b$ litres have been poured into jug A, and $t' \cdot a$ litres have been poured out of it for some integer t'. The amount of water in jug A after s iterations is then $s \cdot b - t' \cdot a = v + (t - t') \cdot a$.

We claim that t' must equal t: If $t' > t$ then jug A would contain at most $v - a$ litres which is negative, while if $t' < t$ then jug A would contain at least $v + a$ litres, so it would overfill. Both cases give a contradiction. It follows that jug A contains exactly v litres after s iterations. □

Lemma 9 tells us that if you want to be a Die Hard star, all you need to do is make sure that the target v can be written as $s \cdot b - t \cdot a$ and the algorithm will do the rest of the work. We know that this is impossible unless v happens to a multiple of $\gcd(a, b)$. It happens that the converse is also true:

Lemma 10. *For all a and b there exist non-negative integers s and t such that $\gcd(a, b) = s \cdot b - t \cdot a$.*

We now have a complete understanding of the Die Hard state machine:

Theorem 11. *Assume $a \geq b$. Bruce lives if and only if $0 \leq v \leq a$ and $\gcd(a, b)$ divides v.*

Proof. Assume Bruce lives. By Corollary 4, $\gcd(a, b)$ must divide v. Clearly v must also be within the size of the larger bin.

Now assume $\gcd(a, b)$ divides v and $0 \leq v \leq a$. If $v = 0$ Bruce wins automatically, and if $v = a$ Bruce wins after filling in the large jug. Otherwise, $0 < v < a$. By Lemma 10, $\gcd(a, b) = s \cdot b - t \cdot a$, so $v = k \cdot \gcd(a, b) = (ks) \cdot b - (kt) \cdot a$ for some integer k. Bruce lives by Lemma 9. □

We are almost done – all that remains is to prove Lemma 10. This can be done by analyzing the following algorithm, whose purpose it is to compute the numbers s and t.

Extended Euclid's algorithm $X(n, d)$, where n, d are integers such that $n \geq d \geq 0$.
If $d = 0$, return $(1, 0)$.
Otherwise,
Write $n = qd + r$ where $0 \leq r < d$.
Calculate $(s, t) = X(d, r)$.
Return $(t, s - q \cdot t)$.

Theorem 12. *For every pair of integers n, d such that $n > d \geq 0$, $X(n, d)$ terminates and outputs a pair of integers (s, t) such that $\gcd(n, d) = s \cdot n + t \cdot d$.*

Instead of proving this theorem, let us show on an example how s and t can be calculated by extending the reasoning we used for Euclid's algorithm. Let $n = 73$ and $d = 50$. Euclid's algorithm

goes through the following steps:

$$\begin{aligned} E(73,50) &= E(50,23) && \text{because } 73 = 1\cdot 50 + 23 && \text{(E1)}\\ &= E(23,4) && \text{because } 50 = 2\cdot 23 + 4 && \text{(E2)}\\ &= E(4,3) && \text{because } 23 = 5\cdot 4 + 3 && \text{(E3)}\\ &= E(3,1) && \text{because } 4 = 1\cdot 3 + 1 && \text{(E4)}\\ &= 1. \end{aligned}$$

We want to express 1 as an integer combination of 73 and 50. We start with the equation (E4), which writes 1 as an integer combination of 4 and 3 and work backwards. We take (E4) and (E3) and cancel out the 3 which occurs in both. To do this, we scale the (E3) by -1 on both sides and add (E4) to it to get

$$-1\cdot 23 + 4 = -5\cdot 4 + 1.$$

We can rewrite this as

$$-1\cdot 23 = -6\cdot 4 + 1, \tag{1}$$

which is a representation of 1 as an integer combination of 23 and 4. We can now cancel out the 4 by combining (E2) and (1): After multiplying (E2) by 6 on both sides and subtracting (1) we get

$$6\cdot 50 - 1\cdot 23 = 12\cdot 23 + 1,$$

from where

$$6\cdot 50 = 13\cdot 23 + 1, \tag{2}$$

which is a representation of 1 as an integer combination of 50 and 23. Finally, the 23 disappears if we multiply (E1) by -13 and subtract (2) to obtain

$$-13\cdot 73 + 6\cdot 50 = -13\cdot 50 + 1$$

which gives us the desired integer combination

$$1 = -13\cdot 73 + 19\cdot 50.$$

Lemma 10 is almost the same as Theorem 12. The only difference is that Lemma 10 says that s and t are non-negative, while Theorem 12 says nothing about signs. In this respect, Lemma 10 is *more general* than Theorem 12: It provides a stronger conclusion under the same assumptions. So how can we use Theorem 12 to prove Lemma 10?

As usual, it helps to work through an example first. Suppose we want to write $4 = s\cdot 3 - t\cdot 5$ for non-negative s and t, but all we have is a representation with the wrong sign pattern, say

$$4 = -2\cdot 3 + 2\cdot 5.$$

What can we do to correct the signs? We can add $5\cdot 3$ to the first term and subtract $3\cdot 5$ from the second term. Then

$$4 = (-2+3)\cdot 3 + (2-3)\cdot 5 = 1\cdot 3 - 1\cdot 5$$

which is exactly what we wanted. In general, this may not happen after one step but we can keep repeating until we get what we want.

Proof of Lemma 10. By Theorem 12, there exist integers s and t such that $\gcd(a,b) = s\cdot a + t\cdot b$. Then for every integer k,

$$(s+kb)\cdot a + (t-ka)\cdot b = sa + tb = \gcd(a,b).$$

No matter what the values of s and t are, when k is sufficiently large, $s+kb$ is positive and $t-ka$ is negative. This gives us a representation of $\gcd(a,b)$ of the desired form. □

4 Modular arithmetic

Let's fix an integer p, which we will call the *modulus*. Theorem 6 says that for every integer n there exists a unique integer between 0 and $p-1$ such that $n = qp + r$ for some q. We call r the *remainder* of n *modulo* p — in short n modulo p — and we write

$$r = n \bmod p.$$

We can do arithmetic using the numbers $0, 1, \ldots, p-1$: additions, subtractions, multiplications, divisions. The arithmetic remains valid as long as we take the remainders of all expressions modulo p. Taking remainders all the time is tedious so instead we'll take advantage of the concept of congruence: Two integers m and n are *congruent* modulo p if p divides $m - n$. We write $m \equiv n \pmod{p}$ for "m and n are congruent modulo p." Congruences behave nicely with respect to additions and multiplications:

$$\text{If} \quad x \equiv x' \pmod{p} \quad \text{and} \quad y \equiv y' \pmod{p}, \qquad \begin{aligned} &\text{then} \quad x + y \equiv x' + y' \pmod{p} \\ &\text{and} \quad x \cdot y \equiv x' \cdot y' \pmod{p}. \end{aligned}$$

Addition, subtraction, and multiplication Addition and subtraction modulo p is fairly easy: For example, if I want to calculate $3 + 8$ modulo 9 first I add 3 and 8 as integers to get 11 then I take the remainder of 11 modulo 9 to get 2:

$$(3 + 8) \bmod 9 = 11 \bmod 9 = 2.$$

The *additive inverse* of a modulo p is the number $(-a) \mod p$. Say the modulus is 5. Then the additive inverse of 0 is 0, the additive inverse of 1 is 4, the additive inverse of 2 is 3, and vice versa. Subtraction can then be done by replacing each negative number with its additive inverse.

$$(3 - 7) \bmod 8 = (3 + 1) \bmod 8 = 4 \bmod 8 = 4.$$

To multiply two numbers modulo p, we also multiply them as integers and take the remainder modulo p. For example

$$3 \cdot 4 \bmod 7 = 12 \bmod 7 = 5.$$

Modular arithmetic gives another way to prove some theorems from Lecture 2. Take, for instance, the theorem that n^2 is even if and only if n is even. Here is a proof by cases by modular arithmetic: If n is even then $n \equiv 0 \pmod{2}$, so $n^2 \equiv 0 \pmod{2}$ and n^2 is also even. Otherwise, $n \equiv 1 \pmod{2}$, so $n^2 \equiv 1 \pmod{2}$ which is odd.

Another theorem in that lecture stated that if n is odd then n^2 is of the form $8k + 1$. Here is a proof using modular arithmetic: If n is odd then n must be congruent to 1, 3, 5 or 7 modulo 8, so n^2 must be congruent to 1, 9, 25, or 49 modulo 8. All of them are congruent to 1 modulo 8.

Division It may sometimes happen that two nonzero numbers multiply to zero in modular arithmetic, for example $6 \cdot 10 \bmod 15$ equals zero. This is a problem for division, because we cannot take an equation like $6 \cdot 10 \equiv 0 \cdot 10 \pmod{15}$, divide both sides by 10 and get a meaningful answer.

Fortunately this does not happen when the modulus p is a prime number, so we will assume that this is the case. First, let's show how to calculate *multiplicative inverses*, namely ratios of the form $1/x$, more commonly written as x^{-1}. As usual, division by zero is forbidden, but otherwise the next lemma shows that division is *well-defined*:

Lemma 13. *For every x between 1 and $p-1$ there exists a unique y between 1 and $p-1$ such that $xy \equiv 1 \pmod{p}$.*

Proof. First we show existence of y. Take any x between 1 and $p-1$. Since p is prime, $\gcd(x,p) = 1$. By Theorem 12 there exist integers s, t for which $s \cdot x + t \cdot p = 1$. Taking both sides modulo p, we get that $s \cdot x \equiv 1 \pmod{p}$. Take $y = s \bmod p$. Then $y \cdot x \equiv s \cdot x \pmod{p}$, so $y \cdot x \equiv 1 \pmod{p}$.

Now we show uniqueness. If $xy \equiv 1 \pmod{p}$ and $xy' \equiv 1 \pmod{p}$ then

$$y' \equiv y' \cdot 1 \equiv y' \cdot (xy) \equiv y \cdot (xy') \equiv y \cdot 1 \equiv y \pmod{p},$$

so p must divide $y' - y$. Since $y' - y$ is between $-(p-2)$ and $p-2$, this is impossible unless they are equal. □

The proof of Lemma 13 tells you how to calculate $y = x^{-1}$: First, use the Extended Euclid's algorithm to find s and t such that $s \cdot x + t \cdot p = 1$. Then output $s \bmod p$. For example, to get the multiplicative inverse of 11 modulo 17, I run $X(11, 17)$ to get $(-3, 2)$, namely $(-3) \cdot 11 + 2 \cdot 17 = 1$. Therefore

$$11^{-1} \bmod 17 = -3 \bmod 17 = 14.$$

To divide two numbers, we first take the multiplicative inverse of the denominator, then multiply by the numerator, for instance to divide 3 by 11 modulo 17 we calculate

$$3 \cdot 11^{-1} \bmod 17 = 3 \cdot 14 \bmod 17 = 42 \bmod 17 = 8.$$

5 Encryption*

Encryption is a type of mechanism that allows Alice and Bob to exchange private messages in a public environment, like a cellular network, or the internet.

Here is a simple protocol for doing this called the one-time pad. Say Alice wants to send Bob the message "`Charlie is a spy`". This message is represented as a large number, say by replacing each character with its ASCII representation:

$$m = \texttt{067104097114108105101032105115032097032115112121}$$

Alice and Bob have agreed ahead of time on a number p, which is at least as large as the message m to be encoded (in this example, $p = 10^{50}$ would suffice) and exchanged in secret a random number k between 0 and $p-1$ called the *secret key*.

If Alice wants to send Bob the secret message m, she *encrypts* m as $c = m + k \bmod p$ and posts the *ciphertext* c in public. To recover Alice's message, Bob calculates $c - k \bmod p$.

I will now argue that the content of Alice's message remains secret with respect to any observer Eve who sees the ciphertext but does not know the secret key. This requires a bit of elementary probability. Let's assume that all the numbers in the range $0, 1, \ldots, p-1$ were equally likely to be chosen as the secret key k. Then, regardless of what the message m is, the ciphertext $m + k \bmod n$ observed by Eve is also equally likely to take any of the values $0, 1, \ldots, p-1$. (For example, if $m = 1$ and $p = 4$, then k is equally likely to take any of the values 0, 1, 2, and 3, so $m + k \bmod p$ is equally likely to evaluate to 1, 2, 3, and 0.) Therefore the value observed by Eve is completely independent of the message sent by Alice, and Eve obtains no information about the message.

This type of encryption works as long as Alice and Bob have previously met to exchange the secret key k (and kept it secure in the meantime). Such an assumption is unreasonable if, say, Alice is a

customer in Bob's online store and she wants to send him her credit card number in secret. Alice and Bob could be living on opposite sides of the world and might have never talked to one another before. How can they agree on a secret key in such circumstance?

Before we describe one proposed solution to this problem we need to discuss a bit more modular arithmetic, specifically the operation of powering and its inverse, the *discrete logarithm*

Powering One way to calculate $b^n \bmod p$ is by performing repeated modular multiplication n times:

$$b^n \bmod p = \underbrace{((b \cdot b \bmod p) \cdot b \bmod p) \cdots b \bmod p}_{n \text{ times}}.$$

This works well in practice when the exponent n is not too large, but is very slow when n is on the order of, say, 10^{50}. A better way to power is to take advantage of the following recursive formula for modular exponentiation:

$$b^n \bmod p = \begin{cases} (b^2 \bmod p)^{n/2} \bmod p, & \text{if } n \text{ is even} \\ b \cdot (b^2 \bmod p)^{(n-1)/2} \bmod p, & \text{if } n \text{ is odd} \end{cases}$$

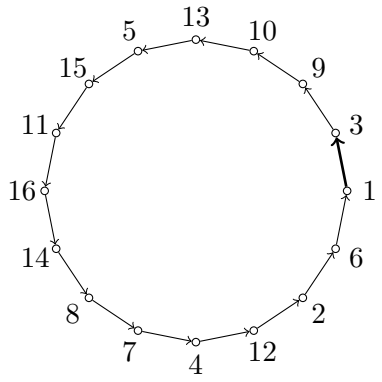
which is valid for all $n \geq 1$. This reduces the task of calculating the n-th power of a number to the task of calculating the $n/2$-th or $(n-1)/2$-th power of some other number (depending on the parity of n), and at most three modular multiplication (one to calculate $b^2 \bmod p$ and possibly another one by b). If we implement this calculation by a computer program, the exponent n will be at worst halved in every step of the recursion, so the total number of recursive calls will not exceed $\log_2 n$. It is therefore possible to calculate $b^n \bmod p$ using at most $3 \log_2 n$ modular multiplications. If n is on the order of 10^{50} then the number of multiplications is about 500, which is a huge improvement over the 10^{50} required by the naive powering procedure.

Discrete logarithm When we work with real numbers, the operation inverse to powering is taking logarithms. By analogy, we call an integer x a *discrete logarithm* of y in base b modulo p if $b^x \bmod p = y$. Unlike ordinary logarithms, discrete logarithms do not behave well at all: Some numbers may have multiple logarithms, and some may have none at all. In these notes we will focus on the situation when p is a prime number, which is already complicated enough.

Let's take the modulus $p = 17$ as an example. Then 10 has no discrete logarithm in base 9. On the other hand, 13 has (at least) two base 9 logarithms as both 9^2 and 9^{10} equal 13 modulo 17. The key to understanding the discrete logarithm is the following theorem which we won't prove:

Theorem 14. *For every prime number p there exists a number g between 1 and $p-1$ such that the sequence $g^0, g^1, \ldots, g^{p-2}$ covers every nonzero number modulo p exactly once.*

Such a number g is called a *primitive root* modulo p. For example, $g = 3$ is a primitive root modulo 17. In the following diagram, arrows indicate multiplication by 3. You can see that the sequence $1, 3, 3^2, \ldots, 3^{15}$ covers every nonzero number modulo 17 exactly once.



Notice also that $3^{16} \equiv 1 \pmod{17}$, so calculating $3^x \bmod 17$ for an arbitrary x reduces to calculating $3^{x \bmod 16} \pmod{17}$ for every x. This is not an accident:

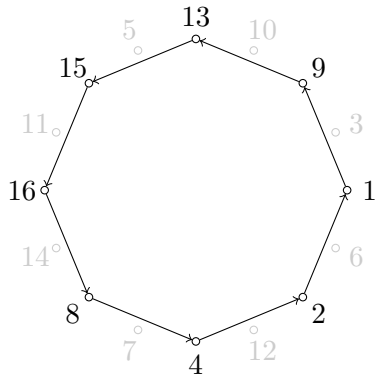
Corollary 15. *For every prime p and every nonzero a modulo p, $a^{p-1} \equiv 1 \pmod{p}$.*

Proof. First assume a is a primitive root g. Since the sequence $1, g, g^2, \ldots, g^{p-2}$ covers every nonzero number modulo p, g^{p-1} must equal g^i for some i between 0 and $p-2$. Dividing both sides by g^i we obtain that $g^{p-1-i} \equiv 1$. The only possibility is that $i = 0$ and so $g^i \equiv 1$; otherwise, $p-1-i$ indexes some entry in the sequence other than the first one, so g^{p-1-i} cannot equal 1 modulo p.

If a is not a primitive root, then it must be a power of a primitive root, namely $a = g^j$ for some j. But then $a^{p-1} \equiv (g^j)^{p-1} \equiv (g^{p-1})^j \equiv 1^j \equiv 1 \pmod{p}$, as desired. □

Corollary 15 tells us in particular that a discrete logarithm can always be represented as a nonzero number modulo $p-1$. Theorem 14 tells us that if the base is a primitive root g then every nonzero number modulo $p-1$ has a unique discrete logarithm in base g. In the above diagram, the discrete logarithm of x base 3 (modulo 17) is the number of steps it takes to reach x starting at 1.

If the base b is not a primitive root, then the powers of b will not cover all the nonzero numbers modulo p, so not every number will have a discrete logarithm in base b. For example, if $b = 9$ then the numbers that have discrete logarithms modulo 17 are the ones reachable from 1 in the following diagram (where the arrows indicate multiplication by 9):



Notice also that $9^8 \bmod 17 = 1$, so the value of the base 9 discrete logarithm (if it exists) reduces to a unique value between 0 and 7, and not between 0 and 15 as was the case in base 3. This is no accident as $9 = 3^2$ so the powers of 9 are the even powers of 3 modulo 17.

Calculating discrete logarithms Let us now assume that the base is a primitive root g modulo p and see how we could go about calculating the discrete logarithm of some input y. One way to do so is to look for the first occurrence of y in the sequence $1, g, g^2, \ldots, g^{p-2}$ modulo p and output the index of this occurrence. This works fine when p is of moderate size, but would be unimaginably slow when p is, say, on the order of 10^{50}. Is there a better way?

One possible strategy would be to begin by looking for some partial information about the discrete logarithm of y — for example, is it even or is it odd? The following theorem tells us that this information can be extracted by calculating a suitable power of g:

Theorem 16. *Assume $p > 2$ is a prime number, g is a primitive root modulo p, and y is a nonzero number modulo p. Then the discrete logarithm of y in base g modulo p is even if and only if $y^{(p-1)/2} \equiv 1 \bmod p$.*

Proof. We prove the theorem by cases depending on the parity of the discrete logarithm of y. If it is even, then $y \equiv g^{2n} \pmod p$ for some k, and $y^{(p-1)/2} \equiv (g^{p-1})^n \equiv 1 \pmod p$ by Corollary 15. If it is odd, then $y \equiv g^{2n+1} \pmod p$ for some n, and $y^{(p-1)/2} \equiv (g^{p-1})^n \cdot g^{(p-1)/2} \equiv g^{(p-1)/2} \pmod p$ by Corollary 15. By Theorem 14, $g^{(p-1)/2}$ cannot be equal to 1 modulo p. □

Let's illustrate this on our example $p = 17, g = 3$. Theorem 16 says that the discrete logarithm of y in base g is even if and only if $y^8 \equiv 1$. The numbers that satisfy this condition are exactly those that are at an even distance from 1 in the diagram: Moving forward by the same even amount 8 times always brings us back to 1, but if the amount is odd we must end up in a different place.

Once we know if the discrete logarithm of y in base g is even, one natural idea is to try recursion and find out the parity of the higher-order bits of the discrete logarithm n. To develop some intuition about how this strategy might work, let's fix $p = 17, g = 3$, and look for the discrete logarithm of 14. We first calculate $14^8 \bmod 17 = 16$, so the discrete logarithm of 14 must be even. In other words, we now know that $n = 2n' + 1$ and so

$$14 = 3^{2n'+1} \quad \text{for some } n' \text{ between 0 and 7.}$$

We can rewrite this equation as

$$14 \cdot 3^{-1} = (3^2)^{n'} \quad \text{for some } n' \text{ between 0 and 7.}$$

We have now reduced our problem of finding the discrete logarithm of 14 in base 3 to the problem of finding the discrete logarithm of $14 \cdot 3^{-1} \equiv 16$ in base $3^2 = 9$! The number 9 is, unfortunately, no longer a primitive root modulo 17 so we cannot apply Theorem 16 again. We can, however, apply the following *generalization*, which you should be able to prove on your own by applying the same reasoning as in the proof of Theorem 16

Theorem 17. *Assume $p > 2$ is a prime number, b is a nonzero number modulo p, and k is the smallest positive integer such that $b^k \equiv 1 \pmod p$. Assume that k is even and y has a discrete logarithm in base b modulo p. The discrete logarithm is even if and only if $y^{k/2} \equiv 1 \pmod p$.*

Theorem 17 tells us that the base 9 discrete logarithm of 16 is even if and only if $16^4 = 1$ modulo 17. This is indeed the case, so we can conclude that n' is even and so we can write $n' = 2n''$, which tells us that

$$16 = 9^{2n''} \quad \text{for some } n'' \text{ between 0 and 3.}$$

So we are left with finding the discrete logarithm n'' of 16 in base $9^2 = 13 \bmod 16$. Continuing in the same manner, we find that the parity of n'' is determined by the value $16^2 \bmod 17$, which again equals 1. Therefore n'' itself is even, and so we must have $16 = 13^{2n'''}$ for $n''' = n''/2$, which

is either zero or one. Now $13^2 \pmod 1 7 = 16$, so it follows by inspection that $n''' = 1$, from where $n'' = 2$, $n' = 4$, and $n = 9$.

In general, this strategy for finding discrete logarithm is much faster than brute-force search through the list of powers as the range of possible discrete logarithms is halved at each iteration. But were we lucky with our example here, or can it be used in a general context? One seemingly innocuous assumption that we made in Theorem 17 is that the number k, which represents the *period* of the sequence $1, b, b^2, \ldots$ must be even. When the modulus p is of the form $2^t + 1$ (as in our example) then the period of the base will remain even in all iterations because it has initial value 2^t and it is halved in every iteration. Prime numbers of this form are called *Fermat primes*, and indeed discrete logarithms modulo a Fermat prime can be calculating reasonably efficiently using the algorithm that we informally described.

Efficient calculation of discrete logarithms can, more generally, be extended to work modulo any prime p provided that the number $p - 1$ does not contain any "large" prime factors.[1] Many prime numbers p have these property, but there are also many that don't. How large can a prime factor of $p - 1$ be when p is prime? Assuming p is greater than 2, $p - 1$ must be an even number, so its largest prime factor cannot exceed $(p - 1)/2$. A prime number p for which $(p - 1)/2$ is also a prime number are called a *safe prime*. It is not known if the number of safe primes is infinite, but very large examples are known. Wikipedia says that "Finding a 500-digit safe prime, like $2 \cdot (1,416,461,893 + 10^{500}) + 1$, is now quite practical."

Unlike in the case of Fermat primes, it is not known how to calculate discrete logarithms modulo a safe prime in an efficient manner. For a number like the Wikipedia safe prime, the best known algorithms may require 10^{160} or so steps.

Key exchange We are now ready to describe a famous proposal by which Alice and Bob can exchange a key that looks secret to any observer using only public communication channels like the internet. First, Alice and Bob agree (in public) on a safe prime p and choose a primitive roof g modulo p. For example, they could take p to equal the Wikipedia safe prime and g to equal 2 (I checked that 2 is a primitive root on my computer). They then exchange the following public messages:

A: Choose a random nonzero number x modulo p (in private). Send the value $a = g^x$ to Bob.

B: Choose a random nonzero number y modulo p (in private). Send the value $b = g^y$ to Alice.

A: Upon receiving b from Bob, calculate b^x. This is Alice's secret key.

B: Upon receiving a from Alice, calculate a^y. This is Bob's secret key.

At the end of the protocol, Alice and Bob, they will both agree on the same secret key g^{xy}.

Now consider a potential observer Eve that has seen Alice's and Bob's messages. This Eve has seen the values g^x and g^y. Can she use these to gain any knowledge about the secret key g^{xy}? One way for Eve to do this is to calculate the discrete logarithms x and y, in which case she can easily compute the secret key. But calculating discrete logarithms modulo safe primes is very time-consuming.

It is, in fact, not known how to calculate g^{xy} efficiently given only knowledge of the values g^x and g^y. This gives Alice and Bob some confidence that their secret key is indeed secret, even though all their messages were exchanged in public.

[1] More precisely, the running time of the discrete logarithm procedure has a linear dependence on the size of the largest prime factor of $p - 1$.

The actual protocols used for key exchange and secure communication used on the internet are a bit more complex than the one I described here. One of the aspects in which the protocol I described may be deficient is that even though Eve can conceivably not completely recover Alice's and Bob's secret key, she might still be able to recover some partial information about it that may allow her to intercept future messages exchanged between them. If you want to know more about this fascinating subject, you can look at the book *Introduction to Modern Cryptography* by Jonathan Katz and Yehuda Lindell.

References

This lecture is based on Chapters 6 and 9 of the text *Mathematics for Computer Science* by E. Lehman, T. Leighton, and A. Meyer.