Sketch of Lecture 1

Wed, 1/20/2021

Review: The calculus of congruences

Example 1. Today is Wednesday. What day of the week will it be a year (365 days) from now?

Solution. Since $365 \equiv 1 \pmod 7$, it will be Thursday on 1/20/2022.

$$a \equiv b \pmod n \qquad \text{means} \qquad a = b + mn \quad (\text{for some } m \in \mathbb{Z})$$

In that case, we say that "a is congruent to b modulo n".

In other words: $a \equiv b \pmod n$ if and only if $a - b$ is divisible by n.

Example 2. $17 \equiv 5 \pmod{12}$ as well as $17 \equiv 29 \equiv -7 \pmod{12}$

We say that $5, 17, 29, -7$ all represent the same **residue** modulo 12.

There are exactly 12 different residues modulo 12.

Example 3. Every integer x is congruent to one of $0, 1, 2, 3, 4, \ldots, 11$ modulo 12.

We therefore say that $0, 1, 2, 3, 4, \ldots, 11$ form a **complete set of residues** modulo 12.

Another natural complete set of residues modulo 12 is: $0, \pm 1, \pm 2, \ldots, \pm 5, 6$

[-6 is not included because $-6 \equiv 6$ modulo 12.]

Online homework. When entering solutions modulo n for online homework, your answer needs to be from one of the two natural sets of residues above.

Example 4. Modulo 7, we have the complete sets of residues $0, 1, 2, 3, 4, 5, 6$ and $0, \pm 1, \pm 2, \pm 3$.
A less obvious set is $0, 3, 3^2, 3^3, 3^4, 3^5, 3^6$.

Review. Note that $3^6 \equiv 1 \pmod 7$ by **Fermat's little theorem**. Because 6 is the smallest positive exponent such that $3^k \equiv 1 \pmod 7$, we say that the **multiplicative order** of $3 \pmod 7$ is 6. This makes $3 \pmod 7$ a **primitive root**.

On the other hand, the **multiplicative order** of $2 \pmod 7$ is 3. (Why?!)

Example 5. $67 \cdot 24 \equiv 4 \cdot 3 \equiv 5 \pmod 7$

The point being that we can (and should!) reduce the factors individually first (to avoid the large number we would get when actually computing $67 \cdot 24$ first). This idea is crucial in the computations we (better, our computers) will later do for cryptography.

Example 6. (but careful!) If $a \equiv b \pmod n$, then $ac \equiv bc \pmod n$ for all integers c.
However, the converse is not true! We can have $ac \equiv bc \pmod n$ without $a \equiv b \pmod n$ (even assuming that $c \not\equiv 0$).

For instance. $2 \cdot 4 \equiv 2 \cdot 1 \pmod 6$ but $4 \not\equiv 1 \pmod 6$

However. $2 \cdot 4 \equiv 2 \cdot 1 \pmod 6$ means $2 \cdot 4 = 2 \cdot 1 + 6m$. Hence, $4 = 1 + 3m$, or, $4 \equiv 1 \pmod 3$.

The issue is that 2 is not invertible modulo 6.

a is invertible modulo $n \iff \gcd(a, n) = 1$

Similarly, $ab \equiv 0 \pmod n$ does not always imply that $a \equiv 0 \pmod n$ or $b \equiv 0 \pmod n$.

For instance. $4 \cdot 15 \equiv 0 \pmod 6$ but $4 \not\equiv 0 \pmod 6$ and $15 \not\equiv 0 \pmod 6$

Armin Straub
straub@southalabama.edu

Good news. These issues do not occur when n is a **prime** p.

- If $ab \equiv 0 \pmod{p}$, then $a \equiv 0 \pmod{p}$ or $b \equiv 0 \pmod{p}$.
- Suppose $c \not\equiv 0 \pmod{p}$. If $ac \equiv bc \pmod{p}$, then $a \equiv b \pmod{p}$.

Example 7. Determine $4^{-1} \pmod{13}$.

Recall. This is asking for the **modular inverse** of 4 modulo 13. That is, a residue x such that $4x \equiv 1 \pmod{13}$.

Brute force solution. We can try the values $0, 1, 2, 3, \ldots, 12$ and find that $x = 10$ is the only solution modulo 13 (because $4 \cdot 10 \equiv 1 \pmod{13}$).

This approach may be fine for small examples when working by hand, but is not practical for serious congruences. On the other hand, the Euclidean algorithm, reviewed below, can compute modular inverses extremely efficiently.

Glancing. In this special case, we can actually see the solution if we notice that $4 \cdot 3 = 12$, so that $4 \cdot 3 \equiv -1 \pmod{13}$ and therefore $4^{-1} \equiv -3 \pmod{13}$.

Example 8. Solve $4x \equiv 5 \pmod{13}$.

Solution. From the previous problem, we know that $4^{-1} \equiv -3 \pmod{13}$.
Hence, $x \equiv 4^{-1} \cdot 5 \equiv -3 \cdot 5 = -2 \pmod{13}$.

(Bézout's identity) Let $a, b \in \mathbb{Z}$ (not both zero). There exist $x, y \in \mathbb{Z}$ such that

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

The integers x, y can be found using the **extended Euclidean algorithm**.
In particular, if $\gcd(a, b) = 1$, then $a^{-1} \equiv x \pmod{b}$.

Here, $\mathbb{Z}$ denotes the set of all integers $0, \pm 1, \pm 2, \ldots$

Example 9. Find $d = \gcd(17, 23)$ as well as integers r, s such that $d = 17r + 23s$.

Solution. We apply the extended Euclidean algorithm:

$$\begin{array}{lll} \gcd(17, 23) & \boxed{23} = 1 \cdot \boxed{17} + 6 & \text{or: } \boxed{A} \quad 6 = 1 \cdot \boxed{23} - 1 \cdot \boxed{17} \\ = \gcd(6, 17) & \boxed{17} = 3 \cdot \boxed{6} - 1 & \boxed{B} \quad 1 = -1 \cdot \boxed{17} + 3 \cdot \boxed{6} \\ = 1 & & \end{array}$$

Backtracking through this, we find that:

$$1 \underset{\boxed{B}}{=} -1 \cdot \boxed{17} + 3 \cdot \boxed{6} \underset{\boxed{A}}{=} -4 \cdot \boxed{17} + 3 \cdot \boxed{23}$$

That is, **Bézout's identity** takes the form $1 = -4 \cdot 17 + 3 \cdot 23$.

Example 10. Determine $17^{-1} \pmod{23}$.

Solution. By the previous example, $1 = -4 \cdot 17 + 3 \cdot 23$. Reducing modulo 23, we get $-4 \cdot 17 \equiv 1 \pmod{23}$. Hence, $17^{-1} \equiv -4 \pmod{23}$. [Or, if preferred, $17^{-1} \equiv 19 \pmod{23}$.]

Armin Straub
straub@southalabama.edu

Example 11. Determine $16^{-1} \pmod{25}$.

Solution. We apply the extended Euclidean algorithm:

$$\begin{aligned} &\gcd(16,25) && \boxed{25} = 2\cdot\boxed{16} - 7 && \text{or:} && \boxed{A} \quad 7 = -1\cdot\boxed{25} + 2\cdot\boxed{16} \\ &= \gcd(7,16) && \boxed{16} = 2\cdot\boxed{7} + 2 && && \boxed{B} \quad 2 = 1\cdot\boxed{16} - 2\cdot\boxed{7} \\ &= \gcd(2,7) && \boxed{7} = 3\cdot\boxed{2} + 1 && && \boxed{C} \quad 1 = \boxed{7} - 3\cdot\boxed{2} \\ &= 1 \end{aligned}$$

Backtracking through this, we find that:

$$1 \underset{\boxed{C}}{=} \boxed{7} - 3\cdot\boxed{2} \underset{\boxed{B}}{=} 7\cdot\boxed{7} - 3\cdot\boxed{16} \underset{\boxed{A}}{=} -7\cdot\boxed{25} + 11\cdot\boxed{16}$$

That is, **Bézout's identity** takes the form $-7\cdot 25 + 11\cdot 16 = 1$.
Reducing modulo 25, we get $11\cdot 16 \equiv 1 \pmod{25}$. Hence, $16^{-1} \equiv 11 \pmod{25}$.

Application: credit card numbers

Have you ever thought about the numbers on your credit card? Usually, these are 16 digits. For instance, 4266 8342 8412 9270.

No worries (or false hopes...). While close, this is not exactly my credit card number.

- The first digit(s) of a credit card identify the issuer of the card. For instance, a leading 4 is typically Visa, 51 to 55 indicate Mastercard, and 34, 37 indicate American Express. The above credit card is indeed a Visa card.

  More information at: https://en.wikipedia.org/wiki/Payment_card_number

- The last digit is a **check digit**, and a valid credit card number must pass the **Luhn check** (patented by IBM scientist Hans Peter Luhn in 1954/60; now in public domain).

  This works as follows: every second digit, starting with the first, is doubled. If that results in a two-digit number, we take the sum of those two digits.

$$\begin{bmatrix} & 4 & 2 & 6 & 6 & 8 & 3 & 4 & 2 & 8 & 4 & 1 & 2 & 9 & 2 & 7 & 0 \\ \times 2 & 8 & & 12 & & 16 & & 8 & & 16 & & 2 & & 18 & & 14 & \\ & 8 & 2 & 3 & 6 & 7 & 3 & 8 & 2 & 7 & 4 & 2 & 2 & 9 & 2 & 5 & 0 \end{bmatrix}$$

  The other half of the digits is left unchanged. We then add all these digits and reduce modulo 10:

$$8+2+3+6+7+3+8+2+7+4+2+2+9+2+5+0 \equiv 0 \pmod{10}$$

  The result of that computation must be 0. Otherwise, the credit card number fails the Luhn check and is invalid.

Example 12. (extra exercise)

(a) Check that the number 4266 8342 8412 9280 fails the Luhn check.

(b) How do we have to change the last digit to turn this into a valid credit card number?

Armin Straub
straub@southalabama.edu

The purpose of the Luhn check is to detect accidental errors.

[A random credit card number has a 90% chance of failing the Luhn check. Why?!]

On the other hand, as the previous example shows, it provides basically no protection against malicious attacks (except against amateur criminals not aware of the Luhn check).

The Luhn check was designed before online banking (patent filed in 1954). So a special focus is on detecting accidental errors that occur frequently when writing down (things like) credit card numbers by hand.

- For instance, it is common that a single digit gets messed up. Every such error is detected by the Luhn check. (Why?!)
- Another common error is to transpose two digits. Every such error (with the exception of 09 versus 90) is detected.

  For instance. A 82 at the beginning contributes $7+2=9$ to the check sum, while a 28 contributes $4+8\equiv 2$ to the sum. Hence, replacing one with the other will result in the Luhn check failing.

  Advanced comment. An alternative checksum formula that can detect all single digit changes as well as all transpositions is the Verhoeff algorithm (1969). It is, however, much more complicated and cannot be readily performed by hand.

Example 13. The doubling and sum-of-digits procedure permutes the digits as follows:

original digit	0	1	2	3	4	5	6	7	8	9
adjusted digit	0	2	4	6	8	1	3	5	7	9
difference $(\mathrm{mod}\,10)$	0	1	2	3	4	6	7	8	9	0

Note. Looking at the differences modulo 10, we can see why the Luhn check is able to detect all transposition errors (except 09 versus 90).

Example 14. The Luhn check has the somewhat complicated feature that every second digit has to be doubled. Why do we not just add all the original digits and reduce the sum modulo 10?

Solution. One reason is that this simplified check does not catch the transposition of two digits. Why?!

[On the other hand, that simplified check does also detect if just a single digit is incorrect.]

Example 15. (extra) The International Standard Book Number ISBN-10 consists of nine digits $a_1a_2\ldots a_9$ followed by a tenth check digit a_{10} (the symbol X is used if the digit equals 10), which satisfies

$$a_{10}\equiv\sum_{k=1}^{9} ka_k \pmod{11}.$$

The ISBN 0-13-186239-? is missing the check digit (printed as "?"). Compute it!

Solution. $1\cdot 0+2\cdot 1+3\cdot 3+4\cdot 1+5\cdot 8+6\cdot 6+7\cdot 2+8\cdot 3+9\cdot 9=210\equiv 1 \pmod{11}$

Hence, the full ISBN is 0-13-186239-1.

This is another example of **error checking**, which is standard practice for all sorts of identification numbers (such as bank account numbers, VIN). With a little more effort **error correction** is also possible.

Comment. The check digit is designed so that it is always possible to detect when a single digit is messed up. It is also always possible to detect when two digits are transposed.

Euler's phi function

Definition 16. Euler's phi function $\phi(n)$ denotes the number of integers in $\{1, 2, ..., n\}$ that are relatively prime to n.

In other words, $\phi(n)$ counts how many residues are invertible modulo n.

If the prime factorization of n is $n = p_1^{k_1} \cdots p_r^{k_r}$, then $\phi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right)$.

Why is this true?

- Why is the formula "obvious" if $n = p^k$ is a prime power?
- On the other hand, for composite n, say $n = ab$, we have: $\boxed{\phi(ab) = \phi(a)\phi(b) \text{ if } \gcd(a,b) = 1}$
  This is a consequence of the Chinese remainder theorem. (Review if necessary! We'll use it later but will only review it briefly then.)

The above formula follows from combining these two observations. Can you fill in the details?

Example 17. Compute $\phi(35)$.

Solution. $\phi(35) = \phi(5 \cdot 7) = \phi(5)\phi(7) = 4 \cdot 6 = 24$

Example 18. Compute $\phi(100)$.

Solution. $\phi(100) = \phi(2^2 \cdot 5^2) = \phi(2^2)\phi(5^2) = (2^2 - 2^1) \cdot (5^2 - 5^1) = 40$

[Alternatively: $\phi(100) = \phi(2^2 \cdot 5^2) = 100\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{5}\right) = 40$]

Historical examples of symmetric encryption

Alice wants to send a secret message to Bob.

What Alice sends will be transmitted through an unsecure medium (like the internet), meaning that others can read it. However, it is important to Alice and Bob that noone else can understand it.

The original message is referred to as the **plaintext** m. What Alice actually sends is called the **ciphertext** c (the encrypted message).

Symmetric encryption algorithms rely on a secret key k (from some **key space**) shared by Alice and Bob (but unknown to anyone else).

Our ultimate goal will be to secure messaging against both:

- eavesdropping (goal: **confidentiality**)
- tampering (goal: **integrity** and, even stronger, **authenticity**)

  The symmetric encryption approach, by itself, cannot fully protect against tampering. For instance, an attacker can collect previously sent messages, resend them, or use them to replace new messages. (You could preface each message with something like a time stamp to address these issues. But that's getting ahead of ourselves; and there are better ways.)

Armin Straub
straub@southalabama.edu

Shift cipher

The alphabet for our messages will be $A, B, ..., Z$, which we will identify with $0, 1, ..., 25$.

So, for instance, C is identified with the number 2.

Example 19. (shift cipher) A key is an integer $k \in \{0, 1, ..., 25\}$. Encryption works character by character using

$$E_k: \quad x \mapsto x + k \pmod{26}.$$

Obviously, the decryption D_k works as $x \mapsto x - k \pmod{26}$.

The **key space** is $\{0, 1, ..., 25\}$. It has size 26. [Well, $k = 0$ is a terrible key. Maybe we should exclude it.]

For instance. If $k = 1$, then the message $HELLO$ is encrypted as $IFMMP$.

If $k = 2$, then the message $HELLO$ is encrypted as $JGNNQ$.

Historic comment. Caesar encrypted some private messages with a shift cipher (typically using $k = 3$). The shift cipher is therefore also often called Caesar's cipher.

While completely insecure today, it was fairly secure at the time (with many of his enemies being illiterate).

Modern comment. Many message boards on the internet "encrypt" things like spoilers or solutions using a shift cipher with $k = 13$. This is called ROT13. What's special about the choice $k = 13$?

Solution. Since $-13 \equiv 13 \pmod{26}$, for ROT13, encryption and decryption are the same!

Example 20. (affine cipher) A slight upgrade to the shift cipher, we encrypt each character as

$$E_{(a,b)}: \quad x \mapsto ax + b \pmod{26}.$$

How does the decryption work? How large is the key space?

Solution. Each character x is decrypted via $x \mapsto a^{-1}(x - b) \pmod{26}$.

The key is $k = (a, b)$. Since a has to be invertible modulo 26, there are $\phi(26) = \phi(2) \cdot \phi(13) = 12$ possibilities for a. There are 26 possibilities for b. Hence, the key space has size $12 \cdot 26 = 312$.

Vigenere cipher (vector shift cipher)

See Section 2.3 of our book for a full description of the Vigenere cipher.

This cipher was long believed by many (until early 20th) to be secure against ciphertext only attacks (more on the classification of attacks shortly).

Example 21. Let us encrypt $HOLIDAY$ using a Vigenere cipher with key BAD (i.e. $1, 0, 3$).

	H	O	L	I	D	A	Y
$+$	B	A	D	B	A	D	B
$=$	I	O	O	J	D	D	Z

Hence, the ciphertext is $IOOJDDZ$.

An encrypted message

Example 22. (bonus challenge!) You find a post-it with the following message:

$$TERRGVATF \quad FGENATRE$$

Can you make any sense of it?

(To collect a bonus point, send me an email before next week with the plaintext and how you found it.)

Example 23. (bonus challenge!) Eve, can you crack the following message?

$$KOBOSNOXSGYOM$$

Word on the street is that Alice was using the Vigenere cipher with a key of size 2.

(To collect a bonus point, send me an email before next week with the plaintext and how you found it.)

Example 24. The challenge from Example 22 was encrypted using .... The key space has size ..., so a brute-force attack results in immediate success: we find that the plaintext is ...

This is the worst kind of vulnerability: we successfully mounted a **ciphertext only attack**.

That is, just knowing the encrypted message, we were able to decrypt it (and discover the key that was used).

Attacks

So far, we considered the weakest kind of attack only: namely, a **ciphertext only attack**. And, even then, the historical ciphers prove to be terribly insecure.

However, we need to also worry about attacks where our enemy has additional insight.

- In a **known plaintext attack**, the enemy somehow has knowledge of a plaintext-ciphertext pair (m, c).
- In a **chosen plaintext attack**, the enemy can, herself, compute $c = E(m)$ for a chosen plaintext m ("gained some sort of access to our encryption device").
- In a **chosen ciphertext attack**, the enemy can, herself, compute $m = D(c)$ for a chosen ciphertext c ("gained some sort of access to our decryption device").

There exist many variations of these. Sometimes, the attacker can make several choices (maybe even adaptively), sometimes she only has partial information.

Example 25. Alice sends the ciphertext $BKNDKGBQ$ to Bob. Somehow, Eve has learned that Alice is using the Vigenere cipher and that the plaintext is $ALLCLEAR$. Next day, Alice sends the message $DNFFQGE$. Crack it and figure out the key that Alice used! (What kind of attack is this?)

Solution. This is a known plaintext attack.

Since $m + k = c$ (to be interpreted characterwise, modulo 26, and with k repeated as necessary), we can find k simply as $k = c - m$.

For instance, since A (value 0!) got encrypted to B, the first letter of the key is B.

c			B	K	N	D	K	G	B	Q
m		$-$	A	L	L	C	L	E	A	R
k		$=$	B	Z	C	B	Z	C	B	Z

We conclude that the key is $k = BZC$. Now, we can decrypt any future message that Alice sends using this key. For instance, we easily decrypt $DNFFQGE$ to $CODERED$ (using $m = c - k$).

All of the historical ciphers we have seen, including the substitution cipher below, fall apart completely under a known plaintext attack.

Example 26. (substitution cipher) In a substitution cipher, the key k is some permutation of the letters $A, B, ..., Z$. For instance, $k = FRA...$. Then we encrypt $A \to F$, $B \to R$, $C \to A$ and so on. How large is the key space?

Solution. Key space has size $26! \approx 10^{26.6} \approx 2^{88.4}$, so a key can be stored using 89 bits. That's actually a fairly large key space (for instance, DES has a key size of 56 bits only). Too large to go through by brute force.

However, still easy to break. Since each letter is always replaced with the same letter, this cipher is susceptible to a **frequency attack**, exploiting that certain letters (and, more generally, letter combinations!) occur much more frequently in, say, English text than others. For instance, Lewand's book on Cryptology lists the following frequencies:

E: 12.7%, T: 9.1%, A: 8.2%, O: 7.5%, I: 7%, N: 6.7%, S: 6.3%, H: 6.1%, R: 6%, D: 4.3%, L: 4%, C: 2.8%, ...

The rarest letters are Q and Z with a frequency of about 0.1% only. (The exact frequencies and precise ordering various between different sources and the body of text that the frequencies were obtained from.)

The most common letter pairs (digrams) are TH HE AN RE ER IN ON AT ND ST ES EN OF TE ED OR TI HI AS TO.

More information at: https://en.wikipedia.org/wiki/Letter_frequency

Comment. Note that the frequencies and even the ranking depend considerably on the source of text. For instance, using government telegrams, a military resource lists EN followed by RE, ER as the most frequent digrams. That same manual suggests SENORITA as a mnemonic to remember the most frequent letters.

http://www.umich.edu/~umich/fm-34-40-2/ (Field Manual 34-40-2, Department of the Army, 1990)

Example 27. It seems convenient to add the space as a 27th letter in the historic encryption schemes. Can you think of a reason against doing that?

Solution. In most texts, the space occurs more frequently and more regularly than any other letter. Adding it to the encryption schemes would make them even more susceptible to attacks.

Fermat's little theorem

Example 28. (warmup) What a terrible blunder... Explain what is wrong!

$$\text{(incorrect!)} \quad 10^9 \equiv 3^2 = 9 \equiv 2 \pmod 7$$

Solution. $10^9 = 10 \cdot 10 \cdot \ldots \cdot 10 \equiv 3 \cdot 3 \cdot \ldots \cdot 3 = 3^9$. Hence, $10^9 \equiv 3^9 \pmod 7$.

However, there is no reason, why we should be allowed to reduce the exponent by 7 (and it is incorrect).

Corrected calculation. $3^2 \equiv 2$, $3^4 \equiv 4$, $3^8 \equiv 16 \equiv 2$. Hence, $3^9 = 3^8 \cdot 3^1 \equiv 2 \cdot 3 \equiv -1 \pmod 7$.

By the way, this approach of computing powers via exponents that are $2,\ 4,\ 8,\ 16,\ 32,\ \ldots$ is called **binary exponentiation**. It is crucial for efficiently computing large powers.

Corrected calculation (using Fermat). $3^6 \equiv 1$ just like $3^0 = 1$. Hence, we are allowed to reduce exponents modulo 6. Hence, $3^9 \equiv 3^3 \equiv -1 \pmod 7$.

Theorem 29. (Fermat's little theorem) Let p be a prime, and suppose that $p \nmid a$. Then

$$a^{p-1} \equiv 1 \pmod p.$$

Proof. (beautiful!) Since a is invertible modulo p, the first $p-1$ multiples of a,

$$a, 2a, 3a, \ldots, (p-1)a$$

are all different modulo p. Clearly, none of them is divisible by p.

Consequently, these values must be congruent (in some order) to the values $1, 2, \ldots, p-1$ modulo p. Thus,

$$a \cdot 2a \cdot 3a \cdot \ldots \cdot (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (p-1) \pmod p.$$

Cancelling the common factors (allowed because p is prime!), we get $a^{p-1} \equiv 1 \pmod p$. □

Remark. The "little" in this theorem's name is to distinguish this result from Fermat's last theorem that $x^n + y^n = z^n$ has no integer solutions if $n > 2$ (only recently proved by Wiles).

Example 30. (bonus challenge!) You intercept the following message from Alice:

WHCUHFWXOWHUQXOMOMQVSQWAMWHCUHFXOLNWXQMQVSQWAWMQLN

Your experience tells you that Alice is using a substitution cipher. You also know that this message contains the word "secret". Can you crack it?

Note. In modern practice, it is not uncommon to know (or suspect) what a certain part of the message should be. For instance, PDF files start with "%PDF" (0x25504446).

See https://en.wikipedia.org/wiki/Magic_number_(programming) for more such instances.

(To collect a bonus point, send me an email before next week with the plaintext and how you found it.)

Example 31. Compute $3^{1003} \pmod{101}$.

Solution. Since 101 is a prime, $3^{100} \equiv 1 \pmod{101}$ by Fermat's little theorem.

Because $3^{100} \equiv 3^0 \pmod{101}$, this enables us to reduce exponents modulo 100.

In particular, since $1003 \equiv 3 \pmod{100}$, we have $3^{1003} \equiv 3^3 = 27 \pmod{101}$.

Euler's theorem

Recall that Fermat's little theorem is just the special case of Euler's theorem :

Theorem 32. (Euler's theorem) If $n \geqslant 1$ and $\gcd(a,n) = 1$, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Proof. Euler's theorem can be proved along the lines of our earlier proof of Fermat's little theorem. The only adjustment is to only start with multiples ka where k is invertible modulo n. There are $\phi(n)$ such residues k, and so that's where Euler's phi function comes in. Can you complete the proof? □

Example 33. What are the last two (decimal) digits of 3^{7082}?

Solution. We need to determine $3^{7082} \pmod{100}$. $\phi(100) = \phi(2^2 5^2) = \phi(2^2)\phi(5^2) = (2^2 - 2^1)(5^2 - 5^1) = 40$.

Since $\gcd(3, 100) = 1$ and $7082 \equiv 2 \pmod{40}$, Euler's theorem shows that $3^{7082} \equiv 3^2 = 9 \pmod{100}$.

Binary exponentiation

Example 34. Compute $3^{25} \pmod{101}$.

Solution. Fermat's little theorem is not helpful here.

Instead, we do **binary exponentiation**:

$3^2 = 9$, $3^4 = 81 \equiv -20$, $3^8 \equiv (-20)^2 = 400 \equiv -4$, $3^{16} \equiv (-4)^2 \equiv 16$, all modulo 101

$25 = 16 + 8 + 1$ [Every integer $n \geqslant 0$ can be written as a sum of distinct powers of 2 (in a unique way).]

Hence, $3^{25} = 3^{16} \cdot 3^8 \cdot 3^1 \equiv 16 \cdot (-4) \cdot 3 = -192 \equiv 10 \pmod{101}$.

Example 35. (extra practice) Compute $2^{20} \pmod{41}$.

Solution. $2^2 = 4$, $2^4 = 16$, $2^8 = 256 \equiv 10$, $2^{16} \equiv 100 \equiv 18$. Hence, $2^{20} = 2^{16} \cdot 2^4 \equiv 18 \cdot 16 = 288 \equiv 1 \pmod{41}$.

Or: $2^5 = 32 \equiv -9 \pmod{41}$. Hence, $2^{20} = (2^5)^4 \equiv (-9)^4 = 81^2 \equiv (-1)^2 = 1 \pmod{41}$.

Comment. Write $a = 2^{20} \pmod{41}$. It follows from Fermat's little theorem that $a^2 = 2^{40} \equiv 1 \pmod{41}$. The argument below shows that $a \equiv \pm 1 \pmod{41}$ [but we don't know which until we do the calculation].

The equation $x^2 \equiv 1 \pmod{p}$ is equivalent to $(x-1)(x+1) \equiv 0 \pmod{p}$ [b/c $(x-1)(x+1) = x^2 - 1$]. Since p is a prime and $p \mid (x-1)(x+1)$, we must have $p \mid (x-1)$ or $p \mid (x+1)$. In other words, $x \equiv \pm 1 \pmod{p}$.

Armin Straub
straub@southalabama.edu

Representations of integers in different bases

We are commonly using the **decimal system** of writing numbers:

$$1234 = 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3.$$

10 is called the base, and $1, 2, 3, 4$ are the digits in base 10. To emphasize that we are using base 10, we will write $1234 = (1234)_{10}$. Likewise, we write

$$(1234)_b = 4 \cdot b^0 + 3 \cdot b^1 + 2 \cdot b^2 + 1 \cdot b^3.$$

In this example, $b > 4$, because, if b is the base, then the digits have to be in $\{0, 1, ..., b-1\}$.

Example 36. $25 = \boxed{1} \cdot 2^4 + \boxed{1} \cdot 2^3 + \boxed{0} \cdot 2^2 + \boxed{0} \cdot 2^1 + \boxed{1} \cdot 2^0$. We write $25 = (11001)_2$.

Example 37. Express 49 in base 2.

Solution.

- $49 = 24 \cdot 2 + \boxed{1}$. Hence, $49 = (...1)_2$ where ... are the digits for 24.
- $24 = 12 \cdot 2 + \boxed{0}$. Hence, $49 = (...01)_2$ where ... are the digits for 12.
- $12 = 6 \cdot 2 + \boxed{0}$. Hence, $49 = (...001)_2$ where ... are the digits for 6.
- $6 = 3 \cdot 2 + \boxed{0}$. Hence, $49 = (...0001)_2$ where ... are the digits for 3.
- $3 = 1 \cdot 2 + \boxed{1}$, with $\boxed{1}$ left over. Hence, $49 = (110001)_2$.

Other bases.

What is 49 in base 3? $49 = 16 \cdot 3 + \boxed{1}$, $16 = 5 \cdot 3 + \boxed{1}$, $5 = 1 \cdot 3 + \boxed{2}$, $\boxed{1}$. Hence, $49 = (1211)_3$.

What is 49 in base 5? $49 = (144)_5$.

What is 49 in base 7? $49 = (100)_7$.

Example 38. Bases 2, 8 and 16 (binary, octal and hexadecimal) are commonly used in computer applications.

For instance, in JavaScript or Python, 0b... means $(...)_2$, 0o... means $(...)_8$, and 0x... means $(...)_{16}$.

The digits $0, 1, ..., 15$ in hexadecimal are typically written as $0, 1, ..., 9, A, B, C, D, E, F$.

Example. FACE value in decimal? $(FACE)_{16} = 15 \cdot 16^3 + 10 \cdot 16^2 + 12 \cdot 16 + 14 = 64206$

Practical example. `chmod 664 file.tex` (change file permission)

664 are octal digits, consisting of three bits: $1 = (001)_2$ execute (x), $2 = (010)_2$ write (w), $4 = (100)_2$ read (r)

Hence, 664 means rw,rw,r. What is rwx,rx,-? 750

By the way, a fourth (leading) digit can be specified (setting the flags: setuid, setgid, and sticky).

Example 39. (terrible jokes, parental guidance advised)

There are I0 types of people... those who understand binary, and those who don't.

Of course, you knew that. How about:

There are II types of people... those who understand Roman numerals, and those who don't.

It's not getting any better:

There are I0 types of people... those who understand hexadecimal, F the rest...

Example 40. (yet another joke) Why do mathematicians confuse Halloween and Christmas?

Because 31 Oct = 25 Dec.

Get it? $(31)_8 = 1 + 3 \cdot 8 = 25$ equals $(25)_{10} = 25$.

Fun borrowed from: https://en.wikipedia.org/wiki/Mathematical_joke

Modern ciphers

Example 41. For modern ciphers, we will change the alphabet from $A, B, ..., Z$ to $0, 1$. One of the most common ways of encoding text is **ASCII**.

In ASCII (American Standard Code for Information Interchange), each letter is represented using 8 bits (1 byte). Among the $2^8 = 256$ many characters are the usual letters, as well as common symbols.

For instance: space $= (20)_{16}$, "0"$=(30)_{16}$, $A = (41)_{16} = (0100, 0001)_2 = 65$, $a = (61)_{16} = (0110, 0001)_2 = 97$

See, for instance, http://www.asciitable.com for the full table.

Example 42. The new (8/2018) insignia of **FinCEN** features binary digits. What do they mean?

01000110 01101001 01101110 01000011 01000101 01001110 https://www.fincen.gov

By the way. If you ever have more than $\$10,000$ in foreign accounts, you must file a report to FinCEN.

One-time pad

Definition 43. The "exclusive or" (XOR), often written $\oplus$, is defined bitwise:

	0	0	1	1
$\oplus$	0	1	0	1
$=$	0	1	1	0

Note. On the level of individual bits, this is just addition modulo 2.

By the way. Best thing about a boolean: even if you are wrong, you are only off by a bit.

Example 44. $1011 \oplus 1111 = 0100$

Example 45. Observe that $a \oplus b \oplus b = a$.

One way to see that is think bitwise in terms of addition modulo 2. Then, $a + b + b = a + 2b \equiv a \pmod 2$.

A **one-time pad** works as follows. We use a key k of the same length as the message m. Then the ciphertext is

$$c = E_k(m) = m \oplus k.$$

To decipher, we use $m = D_k(c) = c \oplus k$.

As the name indicates, we must never use this key again!

Note. Observe that encryption and decryption are the same routine.

Comment. If that is helpful, a one-time pad is doing exactly the same as the Vigenere cipher if we use a key of the same length as the message (also, we use $0, 1$ as our letters instead of the classical $A, B, ..., Z$).

Example 46. Using a one-time pad with key $k = 1100, 0011$, what is the message $m = 1010, 1010$ encrypted to?

Solution. $c = m \oplus k = 0110, 1001$

If a one-time pad (with perfectly random key) is used exactly once to encrypt a message, then **perfect confidentiality** is achieved (eavesdropping is hopeless).

Meaning that Eve intercepting the ciphertext can draw absolutely no conclusions about the plaintext (because, without information on the key, every text of the right length is actually possible and equally likely), see next example.

Armin Straub
straub@southalabama.edu

Example 47. A ciphertext only attack on the one-time pad is entirely hopeless. Explain why!

Solution. The attacker only knows $c = m \oplus k$. The attacker is unable to get any information on m, because every other message m' (of the right length) could have resulted in the same ciphertext c.

Indeed, the key $k' = m' \oplus c$ encrypts m' to c as well (because $m' \oplus k' = m' \oplus (m' \oplus c) = c$). Moreover, every plaintext m' is equally likely because it corresponds to a unique key.

The next example highlights the importance of only using the key once.

Example 48. (attack on the two-time pad) Alice made a mistake and encrypted the two plaintexts m_1, m_2 using the same key k. How can Eve exploit that?

Solution. Eve knows the two ciphertexts $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$.

Hence, she can compute $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$.

This means that Eve knows $m_1 \oplus m_2$, which is information about the original plaintexts (no key involved!). That's a cryptographic disaster: Eve should never be able to learn *anything* about the plaintexts.

In fact. If the plaintexts are, say, English text encoded using ASCII then Eve very possibly can (almost) reconstruct both m_1 and m_2 from $m_1 \oplus m_2$. The reason for that is that the messages are expressed in ASCII, which means 8 bits per character of text. However, the **entropy** (a measure for the amount of information in a message) of (longer) typical English text is frequently below 2 bits per character.

Some details and beautiful graphical illustration are given at:

http://crypto.stackexchange.com/questions/59/taking-advantage-of-one-time-pad-key-reuse

We saw in Example 47 that ciphertext only attacks on the one-time pad are entirely hopeless. What about other attacks?

Attacks like known plaintext or chosen plaintext don't apply if the key is only to be used once.

Yet, the one-time pad by itself provides **little protection of integrity**. The next example shows how tampering is possible without knowledge about the key.

Example 49. Alice sends an email to Bob using a one-time pad. Eve knows that and concludes that, per email standard, the plaintext must begin with `To: Bob`. Eve wants to tamper with the message and change it to `To: Boo`, for a light scare.

- Eve wants to change the 7th letter of the plain text m from b to o.
- Since b is $0x62$ and o is $0x6F$, we have $b \oplus o = 0x0D$. Hence, $b \oplus 0x0D = o$.
- Therefore, if $e = 0x\underbrace{000000000000}_{6 \text{ characters}}D00\ldots$, then $\underbrace{\text{“TO: Bob...”}}_{m} \oplus e = \underbrace{\text{“TO: Boo...”}}_{m'}$.
- Alice sends $c = m \oplus k$. If Eve changes the ciphertext c to $c' = c \oplus e$, then Bob receives c' and decrypts it to $c' \oplus k = \overbrace{\underbrace{m \oplus k}_{=c} \oplus e}^{c'} \oplus k = m \oplus e = m'$, which is what Eve intended.

Using the one-time pad presents several challenges, including:

- keys must not be reused (see Example 48)
- while perfectly protecting against eavesdropping (if done correctly), the one-time pad is not secure against tampering (see Example 49)
- key distribution and management

  Alice and Bob have to somehow exchange huge amounts of keys, so that, at a later time, they are able to communicate securely.
- for perfect confidentiality, the key must be perfectly random

  But how can we produce huge amounts of random bits?

  Especially, how to teach a deterministic machine like a computer to do that? Think about it! This is much more challenging that it may seem at first...

These issues make one-time pads difficult to use in practice.

Historic comment. During the Cold War, the "hot line" between Washington and Moscow apparently used one-time pads for secure communication.

Example 50. (bonus!) $p = 29137$ is an example of a left-truncatable prime: the number itself as well as all truncations 9137, 137, 37, 7 are prime. By simply exhausting all possibilities (start with a single digit and keep adding (nonzero) digits on the left until no choice results in a prime), we find that there is a largest left-truncatable prime, namely, 357686312646216567629137.

https://www.youtube.com/watch?v=azL5ehbw_24

Challenge. Find the largest left-truncatable prime which does not have 1 as a digit.

Send me the prime, and an explanation how you found it, by next week for a bonus point!

Comment. You can play the same game in bases different from 10. We expect that (based on the prime number theorem), for every base, there always are just a finite number of truncatable primes (an extra bonus if you can point me to a proof of that claim!), though the number tends to increase with larger bases. The largest truncatable prime for base 30, for instance, is not known (it is estimated to have about 82 digits in base 30).

https://oeis.org/A103463

Example 51. One thing that makes the one-time pad difficult to use is that the key needs to be the same length as the plaintext. What if we have a shorter key and just repeat it until it has the length we need?

That's essentially the Vigenere cipher (in a different alphabet).

Solution. Assuming the attacker knows the length of our key (if she doesn't she can just try all possibilities), this is equivalent to using the one-time pad several times with the same key. That should never be done! Even using a key twice means that we become susceptible to a ciphertext only attack (see Example 48).

So, repeating the key is a terrible idea. However, the idea to create a longer (random) key out of a shorter (random) key is good (we will discuss pseudorandom generators next).

Let us emphasize that, in order to be perfectly confidential, the key for a one-time pad must be chosen completely at random (otherwise, an attacker can make assumptions on the used keys).

Indeed, the need to generate random numbers shows in every modern cipher.

Stream ciphers

Once we have a way to generate **pseudorandom numbers**, we can use the idea of the one-time pad to create a **stream cipher**.

Start with key of moderate size (say, 128 bits).

Use the key k and a PRG (**pseudorandom generator**) to generate a much longer **pseudorandom keystream** $\mathrm{PRG}(k)$. Then encrypt $E_k(m) = m \oplus \mathrm{PRG}(k)$.

We lost perfect confidentiality. Security relies on choice of PRG (must be unpredictable).

As with the one-time pad, we must never reuse the same keystream! That does not mean that we cannot reuse the key: we can do that using a **nonce**: $E_k(m) = m \oplus \mathrm{PRG}((\mathrm{nonce}, k))$, where the seed is produced by combining the nonce and k (for instance, just concatenating them).

The nonce is then passed (unencrypted) along with the message.

To make sure that we never reuse the same keystream, we must never use the same nonce with the same key.

Remark. A nonce can only be used once, as is in its name. Apparently, according to Urban Dictionary, it is also common as a British insult, roughly equivalent to wanker.

How to generate random numbers?

Natural randomness is surprisingly difficult to harness.

You can for instance play around with a Geiger counter but our department is short on these and getting lots of random numbers is again challenging.

Linear congruential generators

(linear congruential generator) Let a, b, m be chosen parameters.
From the seed x_0, we produce the sequence $x_{n+1} \equiv a x_n + b \pmod{m}$.

The choice of a, b, m is crucial for this to generate acceptable pseudorandom numbers.

For instance, glibc uses $a = 1103515245$, $b = 12345$, $m = 2^{31}$. (This is one of two implementations.) In that case, each x_i is represented by precisely 31 bits. [Note that the choice of m makes this very fast.]

https://en.wikipedia.org/wiki/Linear_congruential_generator

Linear congruential generators (LCG) are easy to predict and must not be used for cryptographic purposes. More generally, all polynomial generators are cryptographically insecure. They are still used in practice, because they are fast and easy to implement and have decent statistical properties. (For instance, our online homework is generated using random numbers, and there is no need for crypto-level security there.)

Statistical trouble. Can you see why the sequences produced by the glibc LCG alternate between even and odd numbers? (Similarly, other low bits are much less "random" than the higher bits.) Because of this defect, some programs (and other implementations of `rand()` based on LCGs) throw away the low bits entirely.

Comment. The particular choices of a and b in glibc are somewhat mysterious. See, for instance:

https://stackoverflow.com/questions/8569113/why-1103515245-is-used-in-rand

Example 52. Generate values using the linear congruential generator $x_{n+1} \equiv 5x_n + 3 \pmod{8}$, starting with the seed $x_0 = 6$.

Solution. $x_1 \equiv 1$, $x_2 \equiv 0$, $x_3 \equiv 3$, $x_4 \equiv 2$, $x_5 \equiv 5$, $x_6 \equiv 4$, $x_7 \equiv 7$, $x_8 \equiv 6$. This is the value x_0 again, so the sequence will now repeat. Note that we went through all 8 residues before repeating. Period 8.

Note. Because $8 = 2^3$ we can represent each x_i using exactly 3 bits. Then $x_1, x_2, x_3, \ldots = 1, 0, 3, \ldots$ corresponds to the bit stream $(001\ 000\ 011\ \ldots)_2$.

Example 53. (extra) Observe that the sequence produced by the linear congruential generator $x_{n+1} \equiv a x_n + b \pmod{m}$ must repeat, at the latest, after m terms. (Why?!)

One can give precise conditions on a, b, m to achieve a full period m. Namely, this happens if and only if $\gcd(b, m) = 1$ and $a - 1$ is divisible by all primes (as well as 4) dividing m.

(a) Generate values using a linear congruential generator $x_{n+1} \equiv 2x_n + 1 \pmod{10}$, starting with the seed $x_0 = 5$. When do they repeat? Is that consistent with the mentioned condition?

(b) What are possible values for a so that the LCG $x_{n+1} \equiv a x_n + 11 \pmod{100}$ has period 100?

(c) glibc uses $a = 1103515245$, $b = 12345$, $m = 2^{31}$. After how many terms will the sequence repeat?

Solution.

(a) $x_1 \equiv 1$, $x_2 \equiv 3$, $x_3 \equiv 7$, $x_4 \equiv 5$. This is the value x_0 again, so the sequence will repeat. Period 4.
[The period is less than 10. This is as predicted by the mentioned condition, because $a - 1$ is not divisible by 2 and 5.]

(b) We need that $a - 1$ is divisible by 4 and 5. Equivalently, $a \equiv 1 \pmod{20}$. Hence, possible values are $a = 1, 21, 41, 61, 81$.

(c) Clearly, $\gcd(b, m) = 1$. Also, $a - 1$ is divisible by 4 (and no primes other than 2 divide m). Hence, for every seed, values repeat only after going through all 2^{31} residues.

Example 54. Let's use the PRG $x_{n+1} \equiv 5x_n + 3 \pmod 8$ as a stream cipher with the key $k = 4 = (100)_2$. The key is used as the seed x_0 and the keystream is $\mathrm{PRG}(k) = x_1 x_2 \ldots$ (where each x_i is 3 bits). Encrypt the message $m = (101\ 111\ 001)_2$.

Solution. We first use the PRG with seed $x_0 = k$ to produce the keystream $\mathrm{PRG}(k) = 7,\ 6,\ 1,\ \ldots = (111\ 110\ 001\ \ldots)_2$.

We then encrypt and get $c = E_k(m) = m \oplus \mathrm{PRG}(k) = (101\ 111\ 001)_2 \oplus (111\ 110\ 001)_2 = (010\ 001\ 000)_2$.

Decryption. Observe that decryption works in the exact same way:

$D_k(c) = c \oplus \mathrm{PRG}(k) = (010\ 001\ 000)_2 \oplus (111\ 110\ 001)_2 = (101\ 111\ 001)_2$.

Note. The keystream continues as $\mathrm{PRG}(k) = 7, 6, 1, 0, 3, 2, 5, 4, \ldots$ At this point it repeats itself because we obtained the value 4, which was our seed. Since the state of this PRG only depends on the value of x_n, and there are 8 possible values for x_n, the period 8 is the longest possible. The previous (extra) example gave conditions on the PRG that guarantee that the period is as long as possible.

Example 55. Can you think of a way in which the numbers produced by a linear congruential generator differ from truly random ones?

Solution. An easy observation for our small examples is the following: by construction, $x_{n+1} \equiv a x_n + b \pmod m$, individual values don't repeat unless a full period is reached and everything repeats. Truly random numbers do repeat every now and then (however, if m is large, then this observation is not exactly practical).

Of course, knowing the parameters $a,\ b,\ m$, the numbers generated by the PRG are terribly **predictable**. Knowing just one number, we can produce all the next ones (as well as the ones before). A PRG that is safe for cryptographic purposes should not be predictable like that! (See next example.)

The next example illustrates the vulnerability of stream ciphers, based on predictable PRGs.

Recall that it is common to know or guess pieces of plaintexts; for instance every PDF begins with $\%\mathrm{PDF}$.

Example 56. Eve intercepts the ciphertext $c = (111\ 111\ 111)_2$. It is known that a stream cipher with PRG $x_{n+1} \equiv 5x_n + 3 \pmod 8$ was used for encryption. Eve also knows that the plaintext begins with $m = (110\ 1\ldots)_2$. Help her crack the ciphertext!

Solution. Since $c = m \oplus \mathrm{PRG}$, we learn that the initial piece of the keystream is $\mathrm{PRG} = m \oplus c = (110\ 1\ldots)_2 \oplus (111\ 1\ldots)_2 = (001\ 0\ldots)_2$. Since each x_n is 3 bits, we conclude that $x_1 = (001)_2 = 1$.

Because the PRG is predictable, we can now recreate the entire keystream! Using $x_{n+1} \equiv 5x_n + 3 \pmod 8$, we find $x_2 \equiv 0$, $x_3 \equiv 3$, ... In other words, $\mathrm{PRG} = 1, 0, 3, \ldots = (001\ 000\ 011\ \ldots)_2$.

Hence, Eve can decrypt the ciphertext and obtain $m = c \oplus \mathrm{PRG} = (111\ 111\ 111)_2 \oplus (001\ 000\ 011)_2 = (110\ 111\ 100)_2$.

Review.

- A **pseudorandom generator** (PRG) takes a seed x_0 and produces a stream $\mathrm{PRG}(x_0) = x_1\, x_2\, x_3 \ldots$ of numbers, which should "look like" random numbers.

  For cryptographic purposes, these numbers should be indistinguishable from random numbers. Even for somebody who knows everything about the PRG except the seed. (See Example 60.)

- Once we have a PRG, we can use it as a **stream cipher**: Using the key k, we encrypt $E_k(m) = m \oplus \mathrm{PRG}(k)$. [Here, the key stream $\mathrm{PRG}(k)$ is assumed to be in bits.]

  As with the one-time pad, we must never reuse the same keystream!

- To reuse the key, we can use a **nonce**: $E_k(m) = m \oplus \mathrm{PRG}((\mathrm{nonce}, k))$, where the seed is produced by combining the nonce and k (for instance, just concatenating them).

  The nonce is then passed (unencrypted) along with the message.

  To never reuse the same keystream, we must never use the same nonce with the same key.

Linear feedback shift registers

Here is another basic idea to generate pseudorandom numbers:

(linear feedback shift register (LFSR) Let ℓ and $c_1, c_2, \ldots, c_\ell$ be chosen parameters. From the seed $(x_1, x_2, \ldots, x_\ell)$, where each x_i is one bit, we produce the sequence

$$x_{n+\ell} \equiv c_1 x_{n+\ell-1} + c_2 x_{n+\ell-2} + \ldots + c_\ell x_n \pmod 2.$$

This method is particularly easy to implement in hardware (see Example 58), and hence suited for applications that value speed over security (think, for instance, encrypted television).

Example 57. Which sequence is generated by the LFSR $x_{n+2} \equiv x_{n+1} + x_n \pmod 2$, starting with the seed $(x_1, x_2) = (0, 1)$?

Solution. $(x_1, x_2, x_3, \ldots) = (0, 1, 1, 0, 1, 1, \ldots)$ has period 3.

Note. Observe that the two previous values determine the state, so there are $2^2 = 4$ states of the LFSR. The state $(0, 0)$ is special (it generates the zero sequence $(0, 0, 0, 0, \ldots)$), so there are 3 other states. Hence, it is clear that the generated sequence has to repeat after at most 3 terms.

Comment. Of course, if we don't reduce modulo 2, then the sequence $x_{n+2} = x_{n+1} + x_n$ generates the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

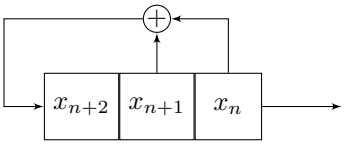
Example 58. Which sequence is generated by the LFSR $x_{n+3} \equiv x_{n+1} + x_n \pmod 2$, starting with the seed $(x_1, x_2, x_3) = (0, 0, 1)$? What is the period?

[Let us first note that the LFSR has $2^3 = 8$ states. Since the state $(0, 0, 0)$ remains zero forever, 7 states remain. This means that the generated sequence must be periodic, with period at most 7.]

Solution. $(x_1, x_2, x_3, \ldots) = (0, 0, 1, 0, 1, 1, 1, 0, 0, 1, \ldots)$ has period 7.

Again, this is not surprising: 3 previous values determine the state, so there are $2^3 = 8$ states. The state $(0, 0, 0)$ is special, so there are 7 other states.

Note that this LFSR can be implemented in hardware using three registers (labeled x_n, x_{n+1}, x_{n+2} in the sketch to the right). During each cycle, the value of x_n is read off as the next value produced by the LFSR.



Note. In the part $0, 0, 1, 0, 1, 1, 1$ that repeats, the bit 1 occurs more frequently than 0.

The reason for that is that the special state $(0, 0, 0)$ cannot appear.

For the same reason, the bit 1 will always occur slightly more frequently than 0 in LFSRs. However, this becomes negligible if the period is huge, like $2^{31} - 1$ in Example 59.

Armin Straub
straub@southalabama.edu

Example 59. The recurrence $x_{n+31} \equiv x_{n+28} + x_n \pmod 2$, with a nonzero seed, generates a sequence that has period $2^{31}-1$.

Note that this is the maximal possible period: this LFSR has 2^{31} states. Again, the state $(0,0,\ldots,0)$ is special (the entire sequence will be zero), so that there are $2^{31}-1$ other states. This means that the terms must be periodic with period at most $2^{31}-1$.

Comment. glibc (the second implementation) essentially uses this LFSR.

Advanced comment. One can show that, if the characteristic polynomial $f(T) = x^\ell + c_1 x^{\ell-1} + c_2 x^{\ell-2} + \ldots + c_\ell$ is irreducible modulo 2, then the period divides $2^\ell - 1$. Here, $f(T) = T^{31} + T^{28} + 1$ is irreducible modulo 2, so that the period divides $2^{31}-1$. However, $2^{31}-1$ is a prime, so that the period must be exactly $2^{31}-1$.

Example 60. Eve intercepts the ciphertext $c = (1111\ 1011\ 0000)_2$ from Alice to Bob. She knows that the plaintext begins with $m = (1100\ 0\ldots)_2$. Eve thinks a stream cipher using a LFSR with $x_{n+3} \equiv x_{n+2} + x_n \pmod 2$ was used. If that's the case, what is the plaintext?

Solution. The initial piece of the keystream is $\mathrm{PRG} = m \oplus c = (1100\ 0\ldots)_2 \oplus (1111\ 1\ldots)_2 = (0011\ 1\ldots)_2$.

Each x_n is a single bit, and we have $x_1 \equiv 0$, $x_2 \equiv 0$, $x_3 \equiv 1$. The given LFSR produces $x_4 \equiv x_3 + x_1 \equiv 1$, $x_5 \equiv x_4 + x_2 \equiv 1$, $x_6 \equiv 0$, $x_7 \equiv 1$, and so on. Continuing, we obtain $\mathrm{PRG} = x_1 x_2 \ldots = (0011\ 1010\ 0111)_2$.

Hence, the plaintext would be $m = c \oplus \mathrm{PRG} = (1111\ 1011\ 0000)_2 \oplus (0011\ 1010\ 0111)_2 = (1100\ 0001\ 0111)_2$.

A PRG is **predictable** if, given the stream it outputs (but not the seed), one can with some precision predict what the next bit will be (i.e. do better than just guessing the next bit).

In other words: the bits generated by the PRG must be indistinguishable from truly random bits, even in the eyes of someone who knows everything about the PRG except the seed.

The PRGs we discussed so far are entirely predictable because the state of the PRGs is part of the random stream they output.

For instance, for a given LFSR, it is enough to know any ℓ consecutive outputs $x_n, x_{n+1}, \ldots, x_{n+\ell-1}$ in order to predict all future output.

We have seen two simple examples of PRGs so far:

- linear congruential generators $x_{n+1} \equiv a x_n + b \pmod m$
- LFSRs $x_{n+\ell} \equiv c_1 x_{n+\ell-1} + c_2 x_{n+\ell-2} + \ldots + c_\ell x_n \pmod 2$

Of course, we could also combine LFSRs and linear congruential generators (i.e. look at recurrences like for LFSRs but modulo any parameter m).

However, much of the appeal of an LFSR comes from its extremely simple hardware realization, as the sketch in Example 58 indicates.

Example 61. (extra) One can also consider nonlinear recurrences (it mitigates some issues). Our book mentions $x_{n+3} \equiv x_{n+2} x_n + x_{n+1} \pmod 2$. Generate some numbers.

Solution. For instance, using the seed $0, 0, 1$, we generate $\overbrace{0, 0, 1}^{\text{seed}}, 0, 1, 1, 1, 0, 1, \ldots$ which now repeats (with period 4) because the state $1, 0, 1$ appeared before. Observe that the generated sequences is only what is called eventually periodic (it is not strictly periodic because $0, 0, 1$ never shows up again).

Example 62. (bonus!) The LFSR $x_{n+31} \equiv x_{n+28} + x_n \pmod 2$ from Example 59, which is used in glibc, is entirely predictable because observing $x_1, x_2, \ldots, x_{31}$ we know what $x_{32}, x_{33}, \ldots$ are going to be. Alice tries to reduce this predictability by using only $x_3, x_6, x_9, \ldots$ as the output of the LFSR. Demonstrate that this PRG is still perfectly predictable by showing the following:

Challenge. Find a simple LFSR which produces $x_3, x_6, x_9, \ldots$

Send me the LFSR, and an explanation how you found it, by next week for a bonus point!

Comment. There is nothing special about this LFSR. Moreover, a generalization of this argument shows that only outputting every Nth bit of an LFSR is always going to result in an entirely predictable PRG.

A popular way to reduce predictability is to combine several LFSRs (in a nonlinear fashion):

Example 63. The CSS (content scramble system) is based on 2 LFSRs and used for the encryption of DVDs. Before discussing the actual CSS let us consider a baby version of CSS. Our PRG uses the LFSR $x_{n+3} \equiv x_{n+1} + x_n \pmod 2$ as well as the LFSR $x_{n+4} \equiv x_{n+2} + x_n \pmod 2$. The output of the PRG is the output of these two LFSRs added with carry.

Adding with carry just means that we are adding bits modulo 2 but add an extra 1 to the next bits if the sum exceeded 1. This is the same as interpreting the output of each LFSR as the binary representation of a (huge) number, then adding these two numbers, and outputting the binary representation of the sum.

If we use $(0,0,1)$ as the seed for LFSR-1, and $(0,1,0,1)$ for LFSR-2, what are the first 10 bits output by our PRG?

Solution. With seed $0,0,1$ LSFR-1 produces $0,1,1,1,0,0,1,0,1,1,\ldots$

With seed $0,1,0,1$ LSFR-2 produces $0,0,0,1,0,1,0,0,0,1,\ldots$

We now add these two:

	0	1	1	1	0	0	1	0	1	1	⋯
+	0	0	0	1	0	1	0	0	0	1	⋯
carry					1						1
	0	1	1	0	1	1	1	0	1	0	⋯

Hence, the output of our PRG is $0,1,1,0,1,1,1,0,1,0,\ldots$.

Important comment. Make sure you realize in which way this CSS PRG is much less predictable than a single LFSR! A single LFSR with ℓ registers is completely predictable since knowing ℓ bits of output (determines the state of the LFSR and) allows us to predict all future output. On the other hand, it is not so simple to deduce the state of the CSS PRG from the output. For instance, the initial $(0,1,\ldots)$ output could have been generated as $(0,0,\ldots)+(0,1,\ldots)$ or $(0,1,\ldots)+(0,0,\ldots)$ or $(1,0,\ldots)+(1,0,\ldots)$ or $(1,1,\ldots)+(1,1,\ldots)$.

[In this case, we actually don't learn anything about the registers of each individual LFSR. However, we do learn how their values have to match up. That's the correlation that is exploited in **correlation attacks**, like the one described next class for the actual CSS scheme.]

Advanced comment. Is the carry important? Yes! Let $a_1, a_2, \ldots$ and $b_1, b_2, \ldots$ be the outputs of LFSR-1 and LFSR-2. Suppose we sum without carry. Then the output is $a_1+b_1, a_2+b_2, \ldots$ (with addition mod 2). If Eve assigns variables $k_1, k_2, \ldots, k_7$ to the $3+4$ seed bits (the key in the stream cipher), then the output of the combined LFSR will be linear in these seven variables (because the a_i and b_i are linear combinations of the k_i). Given just a few more than 7 output bits, a little bit of linear algebra (mod 2) is therefore enough to solve for $k_1, k_2, \ldots, k_7$.

On the other hand, suppose we include the carry. Then the output is $a_1+b_1, a_2+b_2+a_1b_1, \ldots$ (note how a_1b_1 is $1 \pmod 2$ precisely if both a_1 and b_1 are $1 \pmod 2$, which is when we have a carry). This is not linear in the a_i and b_i (and, hence, not linear in the k_i), and we cannot solve for $k_1, k_2, \ldots, k_7$ as before.

Example 64. In each case, determine if the stream could have been produced by the LFSR $x_{n+5} \equiv x_{n+2} + x_n \pmod 2$. If yes, predict the next three terms.

(STREAM-1) $\ldots, 1, 0, 0, 1, 1, 1, 1, 0, 1, \ldots$ (STREAM-2) $\ldots, 1, 1, 0, 0, 0, 1, 1, 0, 1, \ldots$

Solution. Using the LFSR, the values $1, 0, 0, 1, 1$ are followed by $1, 1, 1, 0, \ldots$ Hence, STREAM-1 was not produced by this LFSR.

On the other hand, using the LFSR, the values $1, 1, 0, 0, 0$ are followed by $1, 1, 0, 1, 1, 1, 0, \ldots$ Hence, it is possible that STREAM-2 was produced by the LFSR (for a random stream, the chance is only $1/2^4 = 6.25\%$ that 4 bits matched up). We predict that the next values are $1, 1, 0, \ldots$

Comment. This observation is crucial for the attack on CSS described in Example 65.

Example 65. (CSS) The CSS (content scramble system) is based on 2 LFSRs and used for the encryption of DVDs. Let us indicate (in a slightly oversimplified way) how to break it.

CSS was introduced in 1996 and first compromised in 1999. One big issue is that its key size is 40 bits. Since $2^{40} \approx 1.1 \cdot 10^{12}$ is small by modern standards, even a direct brute-force attack in time 2^{40} is possible.

However, we will see below that poor design makes it possible to attack it in time 2^{16}.

Historic comment. 40 bits was the maximum allowed by US export limitations at the time.

https://en.wikipedia.org/wiki/Export_of_cryptography_from_the_United_States

CSS PRG combines one 17-bit LFSR and one 25-bit LFSR. The bits output by the CSS PRG are the sum of the bits output by the two LFSRs (this is the usual sum, including carries).

The 40 bit key is used to seed the LFSRs (the 4th bit of each seed is "1", so we need $16 + 24 = 40$ other bits).

Here's how we break CSS in time 2^{16}:

- If a movie is encrypted using MPEG then we know the first few, say x (6-20), bytes of the plaintext.
- As in Example 60, this allows us to compute the first x bytes of the CSS keystream.
- We now go through all 2^{16} possibilities for the seed of LFSR-17. For each seed:
  - We generate x bytes using LFSR-17 and subtract these from the known CSS keystream.
  - This would be the output of LFSR-25. As in Example 64, we can actually easily tell if such an output could have been produced by LFSR-25. If yes, then we found (most likely) the correct seed of LFSR-17 and now also have the correct state of LFSR-25.

This kind of attack is known as a correlation attack.

https://en.wikipedia.org/wiki/Correlation_attack

Comment. Similar combinations of LFSRs are used in GSM encryption (A5/1,2, 3 LFSRs); Bluetooth (E0, 4 LFSRs). Due to certain details, these are broken or have serious weaknesses; so, of course, they shouldn't be used. However, it is difficult to update things implemented in hardware...

Sad but important lessons

Review. CSS (content scramble system) is based on 2 LFSRs whose outputs are added with carry (the carry is important because it combines the LFSRs in a nonlinear way).

Combining LFSRs in a nonlinear fashion is a good idea for constructing PRGs for cryptographic purposes (especially because they are simple to implement in hardware). However, as the examples of CSS as well as GSM/Bluetooth encryption show, a lot of attention has to be paid to the details in order not to compromise security.

CSS (and many other examples in recent history) teach us one important lesson:

Do not implement your own ideas for serious crypto!

We will soon see that there exist cryptosystems which are believed to be secure. While none of these beliefs are proven, we do know that certain of these are in fact secure (if implemented correctly) if and only if a certain important mathematical problem cannot be easily solved.

- So, to crack such a system, one has to solve a mathematical problem that many people care about deeply. If this happens, you will most likely read about it in the (academic) news, and you will have an opportunity to update your system in time (most likely, you'll hear about progress much earlier).
- On the other hand, if you use a cryptosystem that is not well-studied, then it may well happen that an adversary breaks your system and keeps exploiting the security leak without you ever learning about it.

Not particularly related but important to keep in mind:

Frequently, security's weakest link are humans. It's very hard to protect against that.
https://en.wikipedia.org/wiki/Social_engineering_(security)

Review: Chinese remainder theorem

Example 66. (warmup)

(a) If $x \equiv 3 \pmod{10}$, what can we say about $x \pmod{5}$?

(b) If $x \equiv 3 \pmod{7}$, what can we say about $x \pmod{5}$?

Solution.

(a) If $x \equiv 3 \pmod{10}$, then $x \equiv 3 \pmod{5}$.
[Why?! Because $x \equiv 3 \pmod{10}$ if and only if $x = 3 + 10m$, which modulo 5 reduces to $x \equiv 3 \pmod{5}$.]

(b) Absolutely nothing! $x = 3 + 7m$ can be anything modulo 5 (because $7 \equiv 2$ is invertible modulo 5).

Example 67. If $x \equiv 32 \pmod{35}$, then $x \equiv 2 \pmod{5}$, $x \equiv 4 \pmod{7}$.

Why?! As in the first part of the warmup, if $x \equiv 32 \pmod{35}$, then $x \equiv 32 \pmod{5}$ and $x \equiv 32 \pmod{7}$.

The Chinese remainder theorem says that this can be reversed!

That is, if $x \equiv 2 \pmod{5}$ and $x \equiv 4 \pmod{7}$, then the value of x modulo $5 \cdot 7 = 35$ is determined.
[How to find the exact $x \equiv 32 \pmod{35}$ is discussed in the next example.]

Armin Straub
straub@southalabama.edu

Example 68. Solve $x \equiv 2 \pmod 5$, $x \equiv 4 \pmod 7$.

Solution. $x \equiv 2 \cdot 7 \cdot \underbrace{7^{-1}_{\bmod 5}}_{3} + 4 \cdot 5 \cdot \underbrace{5^{-1}_{\bmod 7}}_{3} \equiv 42 + 60 \equiv 32 \pmod{35}$

Important comment. Can you see how we need 5 and 7 to be coprime here?

Brute force solution. Note that, while in principle we can always perform a brute force search, this is not practical for larger problems. Here, if x is a solution, then so is $x + 35$. So we only look for solutions modulo 35.
Since $x \equiv 4 \pmod 7$, the only candidates for solutions are $4, 11, 18, \ldots$ Among these, we find $x = 32$.
[We can also focus on $x \equiv 2 \pmod 5$ and consider the candidates $2, 7, 12, \ldots$, but that is even more work.]

Example 69. (extra) Solve $x \equiv 1 \pmod 3$, $x \equiv 2 \pmod 5$, $x \equiv 4 \pmod 7$

Solution. $x \equiv 1 \cdot 5 \cdot 7 \cdot \underbrace{[(5 \cdot 7)^{-1}_{\bmod 3}]}_{-1} + 2 \cdot 3 \cdot 7 \cdot \underbrace{[(3 \cdot 7)^{-1}_{\bmod 5}]}_{1} + 4 \cdot 3 \cdot 5 \cdot \underbrace{[(3 \cdot 5)^{-1}_{\bmod 7}]}_{1} \equiv 67 \pmod{105}$

Note. Comparing with the previous example, note that $67 \equiv 32 \pmod{35}$.

Theorem 70. (Chinese Remainder Theorem) Let $n_1, n_2, \ldots, n_r$ be positive integers with $\gcd(n_i, n_j) = 1$ for $i \neq j$. Then the system of congruences

$$x \equiv a_1 \pmod{n_1}, \quad \ldots, \quad x \equiv a_r \pmod{n_r}$$

has a simultaneous solution, which is unique modulo $n = n_1 \cdots n_r$.

In other words. The Chinese remainder theorem provides a bijective (i.e., 1-1 and onto) correspondence

$$x \pmod{nm} \mapsto \begin{bmatrix} x \pmod n \\ x \pmod m \end{bmatrix}$$

provided that m and n are coprime.

For instance. Let's make the correspondence explicit for $n = 2$, $m = 3$:
$0 \mapsto \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $1 \mapsto \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $2 \mapsto \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $3 \mapsto \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $4 \mapsto \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $5 \mapsto \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Example 71. Solve $x \equiv 4 \pmod 5$, $x \equiv 10 \pmod{13}$.

Solution. $x \equiv 4 \cdot 13 \cdot \underbrace{13^{-1}_{\bmod 5}}_{2} + 10 \cdot 5 \cdot \underbrace{5^{-1}_{\bmod 13}}_{-5} \equiv 104 - 250 \equiv 49 \pmod{65}$

Check. Since it is easy to do so, we should quickly check our answer: $49 \equiv 4 \pmod 5$, $49 \equiv 10 \pmod{13}$

Example 72. Let $p, q > 3$ be distinct primes.

(a) Show that $x^2 \equiv 9 \pmod p$ has exactly two solutions (i.e. ± 3).

(b) Show that $x^2 \equiv 9 \pmod{pq}$ has exactly four solutions (± 3 and two more solutions $\pm a$).

Solution.

(a) If $x^2 \equiv 9 \pmod p$, then $0 \equiv x^2 - 9 = (x-3)(x+3) \pmod p$. Since p is a prime it follows that $x - 3 \equiv 0 \pmod p$ or $x + 3 \equiv 0 \pmod p$. That is, $x \equiv \pm 3 \pmod p$.

(b) By the CRT, we have $x^2 \equiv 9 \pmod{pq}$ if and only if $x^2 \equiv 9 \pmod p$ and $x^2 \equiv 9 \pmod q$. Hence, $x \equiv \pm 3 \pmod p$ and $x \equiv \pm 3 \pmod q$. These combine in four different ways.
For instance, $x \equiv 3 \pmod p$ and $x \equiv 3 \pmod q$ combine to $x \equiv 3 \pmod{pq}$. However, $x \equiv 3 \pmod p$ and $x \equiv -3 \pmod q$ combine to something modulo pq which is different from 3 or -3.

Why primes >3? Why did we exclude the primes 2 and 3 in this discussion?

Comment. There is nothing special about 9. The same is true for $x^2 \equiv a^2 \pmod{pq}$ for each integer a.

Example 73. Determine all solutions to $x^2 \equiv 9 \pmod{35}$.

Solution. By the CRT:

$$\begin{aligned} & x^2 \equiv 9 \pmod{35} \\ \iff\ & x^2 \equiv 9 \pmod 5 \text{ and } x^2 \equiv 9 \pmod 7 \\ \iff\ & x \equiv \pm 3 \pmod 5 \text{ and } x \equiv \pm 3 \pmod 7 \end{aligned}$$

The two obvious solutions modulo 35 are ± 3. To get one of the two additional solutions, we solve $x \equiv 3 \pmod 5$, $x \equiv -3 \pmod 7$. [Then the other additional solution is the negative of that.]

$x \equiv 3 \cdot 7 \cdot \underbrace{7^{-1}_{\bmod 5}}_{3} - 3 \cdot 5 \cdot \underbrace{5^{-1}_{\bmod 7}}_{3} \equiv 63 - 45 \equiv 18 \pmod{35}$

Hence, the solutions are $x \equiv \pm 3 \pmod{35}$ and $x \equiv \pm 17 \pmod{35}$. [$\pm 18 \equiv \pm 17 \pmod{35}$]

Silicon slave labor. We can let Sage (more next page!) do the work for us:

```
Sage] solve_mod(x^2 == 9, 35)
```

$[(17), (32), (3), (18)]$

Sage

Any serious cryptography involves computations that need to be done by a machine. Let us see how to use the open-source computer algebra system **Sage** to do basic computations for us.

Sage is freely available at sagemath.org. Instead of installing it locally (it's huge!) we can conveniently use it in the cloud at cocalc.com from any browser.

[For basic computations, you can also simply use the textbox on our course website.]

Sage is built as a **Python** library, so any Python code is valid. For starters, we will use it as a fancy calculator.

Example 74. Let's start with some basics.

```
Sage] 17 % 12
```

5

```
Sage] (1 + 5) % 2  # don't forget the brackets
```

0

```
Sage] inverse_mod(17, 23)
```

19

```
Sage] xgcd(17, 23)
```

$(1, -4, 3)$

```
Sage] -4*17 + 3*23
```

1

```
Sage] euler_phi(84)
```

24

Example 75. Why is the following bad?

```
Sage] 3^1003 % 101
```

27

The reason is that this computes 3^{1003} first, and then reduces that huge number modulo 101:

```
Sage] 3^1003
```

$$35695912125981779196042292013307897881066394884308000526952849942124372128361032287601\\
01447396641767302556399781555972361067577371671671062036425358196474919874574608035466\\
17047063989041820507144085408031748926871104815910218235498276622866724603402112436668\\
09387969298949770468720050187071564942882735677962417251222021721836167242754312973216\\
80102291029227131545307753863985171834477895265551139587894463150442112884933077598746\\
04125161734774642865878855686737747603770909400 27$$

We know how to efficiently avoid computing huge intermediate numbers (binary exponentiation!). Sage does the same if we instead use something like:

```
Sage] power_mod(3, 1003, 101)
```

27

Example 76. (review) The solutions to $x^2 \equiv 9 \pmod{35}$ are ± 3 and $\pm 17 \pmod{35}$.

Example 77. Determine all solutions to $x^2 \equiv 4 \pmod{105}$.

Solution. By the CRT:

$$\begin{aligned} & x^2 \equiv 4 \pmod{105} \\ \iff\ & x^2 \equiv 4 \pmod 3 \text{ and } x^2 \equiv 4 \pmod 5 \text{ and } x^2 \equiv 4 \pmod 7 \\ \iff\ & x \equiv \pm 2 \pmod 3 \text{ and } x \equiv \pm 2 \pmod 5 \text{ and } x \equiv \pm 2 \pmod 7 \end{aligned}$$

At this point, we see that there are $2^3 = 8$ solutions.

For instance, let us find the solution corresponding to $x \equiv 2 \pmod 3$, $x \equiv 2 \pmod 5$, $x \equiv -2 \pmod 7$:

$$x \equiv 2 \cdot 5 \cdot 7 \cdot \underbrace{[(5 \cdot 7)^{-1}_{\bmod 3}]}_{-1} + 2 \cdot 3 \cdot 7 \cdot \underbrace{[(3 \cdot 7)^{-1}_{\bmod 5}]}_{1} - 2 \cdot 3 \cdot 5 \cdot \underbrace{[(3 \cdot 5)^{-1}_{\bmod 7}]}_{1} \equiv -70 + 42 - 30 = -58 \equiv 47$$

Similarly, we find all eight solutions (note how the solutions pair up):

$\pmod 3$	$\pmod 5$	$\pmod 7$	$\pmod{105}$
2	2	2	2
−2	−2	−2	−2
2	2	−2	47
−2	−2	2	−47
2	−2	2	23
−2	2	−2	−23
−2	2	2	37
2	−2	−2	−37

The complete list of solutions is: $\pm 2, \pm 23, \pm 37, \pm 47$

Silicon slave labor. Once we are comfortable doing it by hand, we can easily let Sage do the work for us:

```
Sage] crt([2,2,-2], [3,5,7])
```

47

```
Sage] solve_mod(x^2 == 4, 105)
```

$[(37), (82), (58), (103), (2), (47), (23), (68)]$

Review: quadratic residues

Definition 78. An integer a is a **quadratic residue** modulo n if $a \equiv x^2 \pmod n$ for some x.

Important note. Products of quadratic residues are quadratic residues.

Example 79. List all quadratic residues modulo 11.

Solution. We compute all squares: $0^2 = 0$, $(\pm 1)^2 = 1$, $(\pm 2)^2 = 4$, $(\pm 3)^2 = 9$, $(\pm 4)^2 \equiv 5$, $(\pm 5)^2 \equiv 3$. Hence, the quadratic residues modulo 11 are $0, 1, 3, 4, 5, 9$.

Important comment. Exactly half of the 10 nonzero residues are quadratic. Can you explain why?

[*Hint.* $x^2 \equiv y^2 \pmod p \iff (x-y)(x+y) \equiv 0 \pmod p \iff x \equiv y$ or $x \equiv -y \pmod p$]

Example 80. List all quadratic residues modulo 15.

Solution. We compute all squares modulo 15: $0^2 = 0$, $(\pm 1)^2 = 1$, $(\pm 2)^2 = 4$, $(\pm 3)^2 = 9$, $(\pm 4)^2 \equiv 1$, $(\pm 5)^2 \equiv 10$, $(\pm 6)^2 \equiv 6$, $(\pm 7)^2 \equiv 4$. Hence, the quadratic residues modulo 15 are $0, 1, 4, 6, 9, 10$.

Important comment. Among the $\phi(15) = 8$ invertible residues, the quadratic ones are $1, 4$ (exactly a quarter). Note that 15 is of the form $n = pq$ with p, q distinct primes.

Armin Straub
straub@southalabama.edu

Theorem 81. Let p, q, r be distinct odd primes.

- The number of invertible residues modulo n is $\phi(n)$.
- The number of invertible quadratic residues modulo p is $\frac{\phi(p)}{2} = \frac{p-1}{2}$.
- The number of invertible quadratic residues modulo pq is $\frac{\phi(pq)}{4} = \frac{p-1}{2}\frac{q-1}{2}$.
- The number of invertible quadratic residues modulo pqr is $\frac{\phi(pqr)}{8} = \frac{p-1}{2}\frac{q-1}{2}\frac{r-1}{2}$.
- ...

Proof.

- We already knew that the number of invertible residues modulo n is $\phi(n)$.
- Think about squaring all residues modulo p to make a complete list of all quadratic residues. Let a^2 be one of the nonzero quadratic residues. As we observed earlier, $x^2 \equiv a^2 \pmod{p}$ has exactly 2 solutions, meaning that exactly two residues (namely $\pm a$) square to a^2. Hence, the number of invertible quadratic residues modulo p is half the number of invertible residues modulo p.
- Again, think about squaring all residues modulo pq to make a complete list of all quadratic residues. Let a^2 be one of the invertible quadratic residues. By the CRT, $x^2 \equiv a^2 \pmod{pq}$ has exactly 4 solutions (why is it important that a is invertible here?!), meaning that exactly four residues square to a^2. Hence, the number of invertible quadratic residues modulo pq is a quarter of the number of invertible residues modulo pq.
- Spell out the situation modulo pqr! □

Comment. Make similar statements when one of the primes is equal to 2.

Example 82. (bonus!) What is the total number of quadratic residues modulo pqr if p, q, r are distinct odd primes? (To collect a bonus point, send me the answer and a short explanation by next week.)

The Blum-Blum-Shub PRG

(Blum-Blum-Shub PRG) Let $M = pq$ where p, q are large primes $\equiv 3 \pmod{4}$.
From the seed y_0, we generate $y_{n+1} \equiv y_n^2 \pmod{M}$.
The random bits x_n we produce are $y_n \pmod{2}$ (i.e. $x_n = \text{least bit of}(y_n)$).

Comments next class.

Example 83. Generate random bits using the B-B-S PRG with $M = 77$ and seed 3.

Solution. With $y_0 = 3$, we have $y_1 \equiv y_0^2 = 9$, followed by $y_2 \equiv y_1^2 \equiv 4 \pmod{77}$, $y_3 \equiv 16$, $y_4 \equiv 25$, $y_5 \equiv 9$, so that the values y_n now start repeating.

These numbers are, however, not the output of the PRG. We only output the least bit of the numbers y_n, i.e. the value of $y_n \pmod{2}$. For $y_1 \equiv 9$ we output 1, for $y_2 \equiv 4$ we output 0, for $y_3 \equiv 16$ we output 0, for $y_4 \equiv 25$ we output 1, and so on.

In other words, the seed 3 produces the sequence $1, 0, 0, 1, 1, 0, 0, 1, 1, 0, \ldots$ of period 4.

Comment. Note that it was completely to be expected that the numbers repeat. In fact, we immediately see that the number of possible y_n is at most the number of invertible quadratic residues, of which there are only $\phi(77)/4 = 15$.

The Blum-Blum-Shub PRG is an example of a PRG, which is believed to be unpredictable.

More precisely, it has been shown that the ability to predict its values is equivalent to being able to efficiently solve the quadratic residuosity problem (which is believed to be hard). Currently, the best way to "solve" the quadratic residuosity problem mod M relies on factoring M. However, factoring large numbers is considered to be hard (and lots of crypto relies on that).

Quadratic residuosity problem. Given big $M=pq$ and a residue x modulo M, decide whether x is a quadratic residue. (About $M/4$ are quadratic residues (the exact number is $\phi(M)/4=(p-1)(q-1)/4$); $M/2$ are easily determined to be nonsquare using the Jacobi symbol [don't worry if you haven't heard about that].)

(Blum-Blum-Shub PRG) Let $M=pq$ where p, q are large primes $\equiv 3 \pmod 4$.
From the seed y_0, we generate $y_{n+1} \equiv y_n^2 \pmod M$.
The random bits x_n we produce are $y_n \pmod 2$ (i.e. $x_n = \text{least bit of}(y_n)$).

B-B-S is very slow, and mostly of theoretical value. However, it is interesting because it is indeed unpredictable (to anyone not knowing the factorization of M) if an important number theory problem is "hard" (this can be made precise), as is believed to be the case.

Why the conditions on p and q? Recall from the CRT that an invertible quadratic residue x^2 modulo $M=pq$ has exactly four squareroots $\pm x$, $\pm y$. The condition $3 \pmod 4$ guarantees that, of these four, exactly one is itself a quadratic residue. As a consequence, the mapping $y \mapsto y^2 \pmod M$ is 1-1 when restricted to invertible quadratic residues (see below).

Comment. For obvious reasons, the seed $y_0 \equiv \pm 1 \pmod M$ should be excluded. Also, for the above considerations, the seed needs to be coprime to M. However, we don't need to worry about that: running into a factor of M by accident is close to impossible (recall that nobody should be able to factor M even on purpose and with lots of time and resources).

Comment. To increase speed, at the expense of some security, we can also take several, say k, bits of y_n (as long as k is small, say, $k \leqslant \log_2 \log_2 M$).

Example 84.

(a) List all invertible quadratic residues modulo 21. Compute the square of all these residues.

(b) Repeat the first part modulo 33 and modulo 35. When computing the squares of these, do you notice a difference modulo 35?

[Note that $35=5\cdot 7$ with $5 \equiv 1 \pmod 4$. This case is excluded in the B-B-S PRG.]

Solution. (final answers only)

(a) Among the $\phi(21)=12$ many invertible elements, the squares are $1, 4, 16$ (exactly a quarter).
Computing the squares: $1^2 \equiv 1$, $4^2 \equiv 16$, $16^2 \equiv 4 \pmod{21}$. Note that the squares are all different!

(b) Modulo 33: among the $\phi(33)=20$ many invertible elements, the squares are $1, 4, 16, 25, 31 \equiv 8^2$ (exactly a quarter). Computing the squares: $1^2 \equiv 1$, $4^2 \equiv 16$, $16^2 \equiv 25$, $25^2 \equiv 31$, $31^2 \equiv 4 \pmod{33}$. Again, all the squares are different!

Modulo 35: among the $\phi(35) = 24$ many invertible elements, the squares are 1, 4, 9, $11 \equiv 9^2$, 16, $29 \equiv 8^2$ (exactly a quarter). Computing the squares: $1^2 \equiv 1$, $4^2 \equiv 16$, $9^2 \equiv 11$, $11^2 \equiv 16$, $16^2 \equiv 11$, $29^2 \equiv 1 \pmod{35}$. Observe that these are not all different: for instance, $9^2 \equiv 16^2 \pmod{35}$.

Advanced comment. The map $x \mapsto x^2 \pmod p$ restricted to invertible quadratic residues is 1-1 if and only if -1 is not a quadratic residue (which, by the next result, is equivalent to $p \equiv 3 \pmod 4$).

[*Sketch of proof.* The map is 1-1 if and only if, for each invertible quadratic residue x^2, exactly one of the two square roots $\pm x$ is itself a quadratic residue. This is equivalent to -1 not being a quadratic residue.

Indeed, if -1 is a quadratic residue, then x and $-x$ are either both quadratic residues or both not.

On the other hand, if not exactly one of $\pm x$ is a quadratic residue then, because exactly half of the invertible residues are quadratic, there would be some pair of residues $\pm z$ which are both quadratic. But then $-zz^{-1} \equiv -1$ would be a quadratic residue.]

Armin Straub
straub@southalabama.edu

Theorem 85. -1 is a quadratic residue modulo (an odd prime) p if and only if $p \equiv 1 \pmod 4$.

In other words, the quadratic congruence $x^2 \equiv -1 \pmod p$ has a solution if and only if $p \equiv 1 \pmod 4$.

Solution. Let us first see that $p \equiv 1 \pmod 4$ is necessary. Assume $x^2 \equiv -1 \pmod p$. Then, by Fermat's little theorem, $x^{p-1} \equiv 1 \pmod p$. On the other hand, $x^{p-1} = (x^2)^{(p-1)/2} \equiv (-1)^{(p-1)/2} \pmod p$. We therefore need $(-1)^{(p-1)/2} = 1$, which is equivalent to $(p-1)/2$ being even. Which is equivalent to $p \equiv 1 \pmod 4$. (Make sure that's absolutely clear!)

On the other hand, assume that $p \equiv 1 \pmod 4$. We will show that $x = \left(\frac{p-1}{2}\right)!$ has the property that $x^2 \equiv -1 \pmod p$. Indeed,

$$\left[\left(\frac{p-1}{2}\right)!\right]^2 = (-1)^{(p-1)/2}\left(1 \cdot 2 \cdot \ldots \cdot \frac{p-1}{2}\right)^2 = (\pm 1) \cdot (\pm 2) \cdot \ldots \cdot \left(\pm \frac{p-1}{2}\right) \equiv -1 \pmod p.$$

[Here, $(\pm 1) \cdot (\pm 2) \cdots$ is short for $1 \cdot (-1) \cdot 2 \cdot (-2) \cdots$.] For the final congruence, observe that $\pm 1, \pm 2, \ldots, \pm \frac{p-1}{2}$ is a complete set of all nonzero residues. When multiplying all residues, each will cancel with its (modular) inverse, except the ones that are their own inverse. But $a \cdot a \equiv 1 \pmod p$ has only the solution $a \equiv \pm 1$, so that ± 1 are the only residues not canceling.

Comment. The final step of our argument is known as Wilson's congruence: $(p-1)! \equiv -1 \pmod p$.

Theorem 86. (advanced) Let $M = pq$ where p, q are primes $\equiv 3 \pmod 4$. Then the sequence generated by $y_{n+1} \equiv y_n^2 \pmod M$ repeats with period dividing $\operatorname{lcm}(\phi(p-1), \phi(q-1))$.

In particular, the period of the corresponding B-B-S PRG divides $\operatorname{lcm}(\phi(p-1), \phi(q-1))$.

Proof.

- Observe that the numbers are $y_n = y_{n-1}^2 = y_{n-2}^4 = \ldots = y_0^{2^n} \pmod M$. Hence, $y_n \equiv y_0^{2^n} \pmod M$.
- Instead of determining the period directly modulo $M = pq$, we determine the periods modulo p and q. [Why? By the CRT, $y_m \equiv y_n \pmod M$ if and only if $y_m \equiv y_n \pmod p$ and $y_m \equiv y_n \pmod q$.] The period modulo M then is the lcm of of the two periods modulo p and q.
- $y_m \equiv y_n \pmod p$
  $\iff y_0^{2^m} \equiv y_0^{2^n} \pmod p$
  $\impliedby 2^m \equiv 2^n \pmod{\phi(p)}$
  [it would be "$\iff$" with $2^m \equiv 2^n \pmod k$ where k is the order of $y_0 \pmod p$]
  $\iff 2^m \equiv 2^n \pmod{p-1}$
  [note that 2 is not invertible $\pmod{p-1}$; but 2 is invertible $\left(\bmod \frac{p-1}{2}\right)$ because $p \equiv 3 \pmod 4$]
  $\iff 2^{m-1} \equiv 2^{n-1} \left(\bmod \frac{p-1}{2}\right)$ [note that $m, n \geqslant 1$]
  $\impliedby m \equiv n \left(\bmod \phi\left(\frac{p-1}{2}\right)\right)$
  [again, it would be "$\iff$" with $m \equiv n \pmod k$ where k is the order of $2 \left(\bmod \frac{p-1}{2}\right)$]
- In other words, the period $m - n$ modulo p divides $\phi\left(\frac{p-1}{2}\right) = \phi(p-1)$.

  Comment. If $p \equiv 3 \pmod 4$, then $\phi\left(\frac{p-1}{2}\right) = \phi(p-1)$. Indeed, note that $p-1$ is divisible by 2 but not by 4. Hence, 2 and $\frac{p-1}{2}$ are coprime, so that $\phi(p-1) = \phi(2)\phi\left(\frac{p-1}{2}\right) = \phi\left(\frac{p-1}{2}\right)$.
- By the CRT, the period modulo $M = pq$ divides $\operatorname{lcm}(\phi(p-1), \phi(q-1))$. □

Example. In Example 83, we had $M = 7 \cdot 11$, so that the period of the PRG must divide $\operatorname{lcm}(\phi(6), \phi(10)) = \operatorname{lcm}(2, 4) = 4$.

Comment. In practice, people therefore say that, for the cycle length of B-B-S to be large, $\gcd(\phi(p-1), \phi(q-1))$ should be small.

Example 87. We mentioned that the unpredictability of the B-B-S PRG relies on the difficulty of factoring large numbers. Here's an indication how difficult it seems to be. In 1991, RSA Laboratories challenged everyone to factor several numbers including:

```
13506641086599522334960321627880596993888147560566702752448514385152651\
06048595338339402871505719094417982072821644715513736804197039641917 4\
30464965892742562393410208643832021103729587257623585096431105640735 0\
15081875106765946292055636855294752135008528794163773285339061097505443\
34999811150056977236890927563
```

Since then, nobody has been able to factor this 1024 bit number (309 decimal digits). Until 2007, cash prizes were offered up to 200,000 USD, with 100,000 USD for the number above.

https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

Let us illustrate how to actually use this number in the B-B-S PRG.

```
Sage] rsa = Integer("13506641086599522334960321627880596993888147560566702752448514385152651\
                     06048595338339402871505719094417982072821644715513736804197039641917 4\
                     30464965892742562393410208643832021103729587257623585096431105640735 0\
                     15081875106765946292055636855294752135008528794163773285339061097505443\
                     34999811150056977236890927563")

Sage] seed = randint(2,rsa-2)

Sage] y = seed; prg = []

Sage] for i in [1..25]:
          y = power_mod(y, 2, rsa)
          prg.append(y % 2)

Sage] prg
```

$$[0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1]$$

If you are able, even after a gigabyte of pseudorandom bits, to predict the next bits with an accuracy better than 50% (which is just pure guessing), then you likely have a shot at factoring the big integer. You would be the first!

Of course, it is not impressive to see a few random bits in the example above. After all, the seed (which you don't know!) itself consists of 1024 random bits. The whole point is that we can, from these 1024 random bits, produce gigabytes of further pseudorandom bits. As of this day, no one would be able to distinguish these from truly random bits.

While all of this works nicely, B-B-S is considered to be too slow for most practical purposes.

Comment. Note that $M = 135...563 \equiv 3 \pmod 4$. Hence it cannot be a product of primes p, q which are both $3 \pmod 4$.

Example 88. (extra) Generate random bits using the B-B-S PRG with $M = 209$ and seed 10. What is the period of the generated sequence? (Then repeat with seed 25.)

Solution. (final answer only) The seed 10 produces the sequence $0, 1, 0, 1, 1, 1, \ldots$ of period 6.

The seed 25 generates the sequence $1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, \ldots$ of period 12.

[By the way, it is an excellent idea to let Sage assist you.]

Primality testing

Recall that it is extremely difficult to factor large integers (this is the starting point for many cryptosystems). Surprisingly, it is much simpler to tell if a number is prime.

Example 89. The following is the number from Example 87, for which RSA Laboratories, until 2007, offered $100,000 to the first one to factorize it. Nobody has been able to do so to this day.

Has the thought crossed your mind that the challengers might be tricking everybody by choosing M to be a huge prime that cannot be factored further? Well, we'll talk more about primality testing soon. But we can actually quickly convince ourselves that M cannot be a prime. If M was prime then, by Fermat's little theorem, $2^{M-1} \equiv 1 \pmod{M}$. Below, we compute $2^{M-1} \pmod{M}$ and find that $2^{M-1} \not\equiv 1 \pmod{M}$. This proves that M is not a prime. It doesn't bring us any closer to factoring it though.

Comment. Ponder this for a while. We can tell that a number is composite without finding its factors. Both sides to this story (first, being able to efficiently tell whether a number is prime, and second, not being able to factor large numbers) are of vital importance to modern cryptography.

```
Sage] rsa = Integer("135066410865995223349603216278805969938881475605667027524485143851\
                     525510604859533833940287150571909441798207282164471551373680419703\
                     964191743046496589274256239341020864383202110372958725762358509643\
                     110564073501508187510676594629205563685529475213500852879416377328\
                     533906109750544334999811150056977236890927563")

Sage] power_mod(2, rsa-1, rsa)

    120939094432033615867650595352956996867540098463588951238902808367556733932202059338 53\
    348534147116662841968124107288512373904071077139405352848835710498409193003137847878 95\
    226029615123284879513798127406300472693925500331497519103479951096634123177725212482 97\
    950196643140069546889855131459759160570963857373851
```

Comment. Just for giggles, let us emphasize once more the need to compute $2^{N-1} \pmod{N}$ without actually computing 2^{N-1}. Take, for instance, the 1024 bit RSA challenge number $N = 135...563$. In Example 89, we computed $2^{N-1} \pmod{N}$, observed that it was $\not\equiv 1$ and concluded that N is not prime. The number 2^{N-1} itself has $N - 1 \approx 2^{1024} \approx 10^{308.3}$ binary digits. It is often quoted that the number of particles in the visible universe is estimated to be between 10^{80} and 10^{100}. Whatever these estimates are worth, our number has WAY more digits (!) than that. Good luck writing it out! [Of course, the binary digits are a single 1 followed by all zeros. However, we need to further compute with that!]

Comment. There is nothing special about 2. You could just as well use, say, 3.

Example 90. (bonus challenge) Find the factors of the following number $M = pq$:

```
893202800574373633936083863874693604950799157730735990874355694281082 7\
076151461165069181335366401887650477533577602609343916545431925218633\
751141065095634529703730490829332440131073471416542829240327143 11
```

As indicated in Example 87, this is difficult. Through some sort of espionage, however, you have learned that $\phi(M)$ is:

```
893202800574373633936083863874693604950799157730735990874355694281082 7\
076151461165069181335366401887657264952783386626998307790668498916912 5\
759563757735725786146787680002256288669908402235207462838677975 12
```

In general, if $M = pq$ is a product of two large primes p, q, given $\phi(M)$, how can we factor M?

Send me the factorization, and an explanation how you found it, by next week for a bonus point!

Comment. Even if we don't know the number of prime factors of M (in the above case we know that M is a product of two primes), we can "efficiently" factor M if we know the value of $\phi(M)$.

Armin Straub
straub@southalabama.edu

Example 91. Fermat's little theorem can be stated in the slightly stronger form:

n is a prime $\iff$ $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \{1, 2, \ldots, n-1\}$

Why? Fermat's little theorem covers the "$\Longrightarrow$" part. The "$\Longleftarrow$" part is a direct consequence of the fact that, if n is composite with divisor d, then $d^{n-1} \not\equiv 1 \pmod{n}$. (Why?!)

Fermat primality test

Input: number n and parameter k indicating the number of tests to run

Output: "not prime" or "likely prime"

Algorithm:

Repeat k times:
  Pick a random number a from $\{2, 3, \ldots, n-2\}$.
  If $a^{n-1} \not\equiv 1 \pmod{n}$, then stop and output "not prime".
Output "likely prime".

If $a^{n-1} \equiv 1 \pmod{n}$ although n is composite, then a is called a **Fermat liar** modulo n.

On the other hand, if $a^{n-1} \not\equiv 1 \pmod{n}$, then n is composite and a is called a **Fermat witness** modulo n.

Flaw. There exist certain composite numbers n (see Definition 93) for which every a is a Fermat liar (or reveals a factor of n). For this reason, the Fermat primality test should not be used as a general test for primality. That being said, for very large random numbers, it is exceedingly unlikely to meet one of these troublesome numbers, and so the Fermat test is indeed used for the purpose of randomly generating huge primes (for instance in PGP). In fact, in that case, we can even always choose $a = 2$ and $k = 1$ with virtual certainty of not messing up.

Next class, we will discuss an extension of the Fermat primality test which solves these issues (and is just mildly slower).

Advanced comment. If n is composite but not an absolute pseudoprime (see Definition 93), then at least half of the values for a satisfy $a^{n-1} \not\equiv 1 \pmod{n}$ and so reveal that n is not a prime. This is more of a theoretical result: for most large composite n, almost every a (not just half) will be a Fermat witness.

Example 92. Suppose we want to determine whether $n = 221$ is a prime. Simulate the Fermat primality test for the choices $a = 38$ and $a = 24$.

Solution.

- First, maybe we pick $a = 38$ randomly from $\{2, 3, \ldots, 219\}$.
  We then calculate that $38^{220} \equiv 1 \pmod{221}$. So far, 221 is behaving like a prime.
- Next, we might pick $a = 24$ randomly from $\{2, 3, \ldots, 219\}$.
  We then calculate that $24^{220} \equiv 81 \not\equiv 1 \pmod{221}$. We stop and conclude that 221 is not a prime.

Important comment. We have done so without finding a factor of n. (To wit, $221 = 13 \cdot 17$.)

Comment. Since 38 was giving us a false impression regarding the primality of n, it is called a **Fermat liar** modulo 221. Similarly, we say that 221 is a **pseudoprime** to the base 38.

On the other hand, we say that 24 was a **Fermat witness** modulo 221.

Comment. In this example, we were actually unlucky that our first "random" pick was a Fermat liar: only 14 of the 218 numbers (about 6.4%) are liars. As indicated above, for most large composite numbers, the proportion of liars will be exceedingly small.

Somewhat suprisingly, there exist composite numbers n with the following disturbing property: every residue a is a Fermat liar or $\gcd(a,n)>1$.

This means that the Fermat primality test is unable to distinguish n from a prime, unless the randomly picked number a happens to reveal a factor (namely, $\gcd(a,n)$) of n (which is exceedingly unlikely for large numbers).

[Recall that, for large numbers, we do not know how to find factors even if that was our primary goal.]

Such numbers are called absolute pseudoprimes:

Definition 93. A composite positive integer n is an **absolute pseudoprime** (or Carmichael number) if $a^{n-1}\equiv 1 \pmod n$ holds for each integer a with $\gcd(a,n)=1$.

The first few are $561, 1105, 1729, 2465, \ldots$ (it was only shown in 1994 that there are infinitely many of them).

These are very rare, however: there are 43 absolute pseudoprimes less than 10^6. (Versus $78,498$ primes.)

Example 94. Show that 561 is an absolute pseudoprime.

Solution. We need to show that $a^{560}\equiv 1 \pmod{561}$ for all invertible residues a modulo 561.

Since $561=3\cdot 11\cdot 17$, $a^{560}\equiv 1 \pmod{561}$ is eqivalent to $a^{560}\equiv 1 \pmod p$ for each of $p=3,11,17$.

By Fermat's little theorem, we have $a^2\equiv 1 \pmod 3$, $a^{10}\equiv 1 \pmod{11}$, $a^{16}\equiv 1 \pmod{17}$. Since $2,10,16$ each divide 560, it follows that indeed $a^{560}\equiv 1 \pmod p$ for $p=3,11,17$.

Comment. Korselt's criterion (1899) states that what we just observed in fact characterizes absolute pseudoprimes. Namely, a composite number n is an absolute pseudoprime if and only if n is squarefree, and for all primes p dividing n, we also have $p-1|n-1$.

Comment. Our argument above shows that, in fact, $a^{80}\equiv 1 \pmod{561}$ for all invertible residues a modulo 561.

Theorem 95. (Korselt's Criterion) A composite positive integer n is an absolute pseudoprime if and only if n is squarefree and $(p-1)|(n-1)$ for each prime divisor p of n.

Proof. Here, we will only consider the "if" part (the "only if" part is also not hard to show but the typical proof requires a little more insight into primitive roots than we currently have).

To that end, assume that n is **squarefree** and that $(p-1)|(n-1)$ for each prime divisor p of n. Let a be any integer with $\gcd(a,n)=1$. We will show that $a^{n-1}\equiv 1 \pmod n$.

n being squarefree means that its prime factorization is of the form $n=p_1\cdot p_2\cdots p_d$ for distinct primes p_i (this is equivalent to saying that there is no integer $m>1$ such that $m^2|n$). By Fermat's little theorem $a^{p_i-1}\equiv 1 \pmod{p_i}$ and, since $(p_i-1)|(n-1)$, we have $a^{n-1}\equiv 1 \pmod{p_i}$ for all p_i. It therefore follows from the Chinese remainder theorem that $a^{n-1}\equiv 1 \pmod n$. □

Comment. Modulo a prime p, Fermat's little theorem implies that $a^p\equiv a \pmod p$ for each integer a. (Why?!) It therefore follows from the above argument that, for an absolute pseudoprime n, we have $a^n\equiv a \pmod n$ for each integer a (and this property characterizes absolute pseudoprimes).

Review. If N is composite, then a residue a is a Fermat liar modulo N if $a^{N-1} \equiv 1 \pmod{N}$.

Example 96. Using Sage, determine all numbers n up to 5000, for which 2 is a Fermat liar.

```
Sage] def is_fermat_liar(x, n):
          return not is_prime(n) and power_mod(x, n-1, n) == 1

Sage] [ n for n in [1..5000] if is_fermat_liar(2, n) ]
```

$$[341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821, 3277, 4033, 4369, 4371, 4681]$$

Even if you have never written any code, you can surely figure out what's going on!

Heads-up! The improved primality test discussed today will reduce this list to just $2047, 3277, 4033, 4681$.

The Miller–Rabin primality test

Review. The congruence $x^2 \equiv 1 \pmod{p}$ has only the solutions $x \equiv \pm 1$.

By contrast, if n is composite (and odd), then $x^2 \equiv 1 \pmod{n}$ has additional solutions.

The Miller–Rabin primality test exploits this difference to fix the issues of the Fermat primality test.

The Fermat primality test picks a and checks whether $a^{n-1} \equiv 1 \pmod{n}$.

- If $a^{n-1} \not\equiv 1 \pmod{n}$, then we are done because n is definitely not a prime.
- If $a^{n-1} \equiv 1 \pmod{n}$, then either n is prime or a is a Fermat liar.
  But instead of leaving off here, we can dig a little deeper:
  Note that $a^{(n-1)/2}$ satisfies $x^2 \equiv 1 \pmod{n}$. If n is prime, then $x \equiv \pm 1$ so that $a^{(n-1)/2} \equiv \pm 1 \pmod{n}$.
  - Hence, if $a^{(n-1)/2} \not\equiv \pm 1 \pmod{n}$, then we again know for sure that n is not a prime.
    Advanced comment. In fact, we can now factor n! See bonus challenge below.
  - If $a^{(n-1)/2} \equiv 1 \pmod{n}$ and $\frac{n-1}{2}$ is divisible by 2, we continue and look at $a^{(n-1)/4} \pmod{n}$.
    - If $a^{(n-1)/4} \not\equiv \pm 1 \pmod{n}$, then n is not a prime.
    - If $a^{(n-1)/4} \equiv 1 \pmod{n}$ and $\frac{n-1}{4}$ is divisible by 2, we continue...

Write $n - 1 = 2^s \cdot m$ with m odd. In conclusion, if n is a prime, then

$$a^m \equiv 1 \quad \text{or, for some } r = 0, 1, \ldots, s-1, \quad a^{2^r m} \equiv -1 \pmod{n}.$$

In other words, if n is a prime, then the values $a^m, a^{2m}, \ldots, a^{2^s m}$ must be of the form $1, 1, \ldots, 1$ or $\ldots, -1, 1, 1, \ldots, 1$. If the values are of this form even though n is composite, then a is a **strong liar** modulo n.

This gives rise to the following improved primality test:

Miller–Rabin primality test

Input: number n and parameter k indicating the number of tests to run

Output: "not prime" or "likely prime"

Algorithm:

Write $n - 1 = 2^s \cdot m$ with m odd.
Repeat k times:
  Pick a random number a from $\{2, 3, \ldots, n-2\}$.
  If $a^m \not\equiv 1 \pmod{n}$ and $a^{2^r m} \not\equiv -1 \pmod{n}$ for all $r = 0, 1, \ldots, s-1$, then
    stop and output "not prime".
Output "likely prime".

Comment. If n is composite, then less than a quarter of the values for a can possibly be strong liars. In other words, for all composite numbers, the odds that the Miller–Rabin test returns "likely prime" are less than 4^{-k}.

Comment. Note that, though it looks more involved, the Miller–Rabin test is essentially as fast as the Fermat primality test (recall that, to compute a^{n-1}, we proceed using binary exponentiation).

Advanced comments. This is usually implemented as a probabilistic test. However, assuming GRH (the generalized Riemann hypothesis), it becomes a deterministic algorithm if we check $a = 2, 3, \ldots, \lfloor 2(\log n)^2 \rfloor$. This is mostly of interest for theoretical applications. For instance, this then becomes a polynomial time algorithm for checking whether a number is prime.

More recently, in 2002, the AKS primality test was devised. This test is polynomial time (without relying on outstanding conjectures like GRH).

Example 97. Suppose we want to determine whether $n = 221$ is a prime. Simulate the Miller–Rabin primality test for the choices $a = 24$, $a = 38$ and $a = 47$.

Solution. $n - 1 = 4 \cdot 55 = 2^s \cdot m$ with $s = 2$ and $m = 55$.

- For $a = 24$, we compute $a^m = 24^{55} \equiv 80 \not\equiv \pm 1 \pmod{221}$. We continue with $a^{2m} \equiv 80^2 \equiv 212 \not\equiv -1$, and conclude that n is not a prime.

  Note. We do not actually need to compute that $a^{n-1} = a^{4m} \equiv 81$, which features in the Fermat test and which would also lead us to conclude that n is not prime.

- For $a = 38$, we compute $a^m = 38^{55} \equiv 64 \not\equiv \pm 1 \pmod{221}$. We continue with $a^{2m} \equiv 64^2 \equiv 118 \not\equiv -1$ and conclude that n is not a prime.

  Note. This case is somewhat different from the previous in that 38 is a Fermat liar. Indeed, $a^{4m} \equiv 118^2 \equiv 1 \pmod{221}$. This means that we have found a nontrivial sqareroot of 1. In this case, the Fermat test would have failed us while the Miller–Rabin test succeeds.

- For $a = 47$, we compute $a^m = 47^{55} \equiv 174 \not\equiv \pm 1 \pmod{221}$. We continue with $a^{2m} \equiv 174^2 \equiv -1$. We conclude that n is a prime or a is a strong liar. In other words, we are not sure but are (incorrectly) leaning towards thinking that 221 was likely a prime.

Comment. In this example, only 4 of the 218 residues $2, 3, \ldots, 219$ are strong liars (namely $21, 47, 174, 200$). For comparison, there are 14 Fermat liars (namely $18, 21, 38, 47, 64, 86, 103, 118, 135, 157, 174, 183, 200, 203$).

Example 98. In Example 94, we saw that all $\phi(561) = 320$ invertible residues a modulo 561 are Fermat liars (that is, they all satisfy $a^{560} \equiv 1 \pmod{561}$). How many of them are strong liars?

Solution. Only 8 of the 558 residues $2, 3, \ldots, 559$ are strong liars (namely $50, 101, 103, 256, 305, 458, 460, 511$). That's about 1.43% (much less than the theoretic bound of 25%).

(bonus challenge) For which $N < 1000$ is the proportion of strong liars the highest?

Here (as illustrated in the case of 561 above) we define the proportion of strong liars to be the proportion of residues among $2, 3, \ldots, N-2$, which are strong liars.

[That proportion is almost 23%, just shy of the theoretical bound of 25%.]

Send in a solution by next week for a bonus point!

Example 99. How can you check whether a huge randomly selected number N is prime?

Solution. Compute $2^{N-1} \pmod N$ using binary exponentiation. If this is $\not\equiv 1 \pmod N$, then N is not a prime. Otherwise, N is a prime or 2 is a Fermat liar modulo N (but the latter is exceedingly unlikely for a huge randomly selected number N; the bonus challenge below indicates that this is almost as unlikely as randomly running into a factor of N).

Comment. There is nothing special about 2 here (you could also choose 3 or any other generic residue).

Example 100. (bonus challenge) If $a^{n-1} \equiv 1 \pmod n$ but $a^{(n-1)/2} \not\equiv \pm 1 \pmod n$, then we can find a factor of n! How?!

For instance. $a = 38$ and $n = 221$ in Example 97.

Comment. However, note that this only happens if a is a Fermat liar modulo n, and these are typically very rare. So, unfortunately, we have not discovered an efficient factorization algorithm. [But we have run into an idea, which is used for some of the best known factorization algorithms. If time permits, more on that later...]

Send in a solution by next week for a bonus point!

How many primes are there?

Theorem 101. (Euclid) There are infinitely many primes.

Proof. Assume (for contradiction) there are only finitely many primes: $p_1, p_2, \ldots, p_n$.
Consider the number $N = p_1 \cdot p_2 \cdot \ldots \cdot p_n + 1$.
None of the p_i divide N (because division of N by any p_i leaves remainder 1).
Thus any prime dividing N is not on our list. Contradiction.

Just being silly. Similarly, there are infinitely many composite numbers.

Indeed, assume (for contradiction) there are only finitely many composites: $m_1, m_2, \ldots, m_n$.
Consider the number $N = m_1 \cdot m_2 \cdot \ldots \cdot m_n$ (don't add 1).
N is not on our list. Contradiction.

Historical note. This is not necessarily a proof by contradiction, and Euclid (300BC) himself didn't state it as such. Instead, one can think of it as a constructive machinery of producing more primes, starting from any finite collection of primes. □

The following famous and deep result quantifies the infinitude of primes.

Theorem 102. (prime number theorem) Let $\pi(x)$ be the number of primes $\leqslant x$. Then

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln(x)} = 1.$$

In other words: Up to x, there are roughly $x/\ln(x)$ many primes.

Examples.

proportion of primes up to 10^6: $\frac{78,498}{10^6} = 7.85\%$ vs the estimate $\frac{1}{\ln(10^6)} = \frac{1}{6\ln(10)} = 7.24\%$

proportion of primes up to 10^{12}: $\frac{37,607,912,018}{10^{12}} = 3.76\%$ vs the estimate $\frac{1}{\ln(10^{12})} = \frac{1}{12\ln(10)} = 3.62\%$

An example of huge relevance for crypto.

By the PNT, the proportion of primes up to 2^{2048} is about $\frac{1}{\ln(2^{2048})} = 0.0704\%$.

That means, roughly, 1 in 1500 numbers of this magnitude are prime. That means we (i.e. our computer) can efficiently generate large random primes by just repeatedly generating large random numbers and discarding those that are not prime.

Comment. Here, $\ln(x)$ is the logarithm with base e. Isn't it wonderful how Euler's number $e \approx 2.71828$ is sneaking up on the primes?

Historical comment. Despite progress by Chebyshev (who succeeded in 1852 in showing that the quotient in the above limit is bounded, for large x, by constants close to 1), the PNT was not proved until 1896 by Hadamard and, independently, de la Vallée Poussin, who both used new ideas due to Riemann.

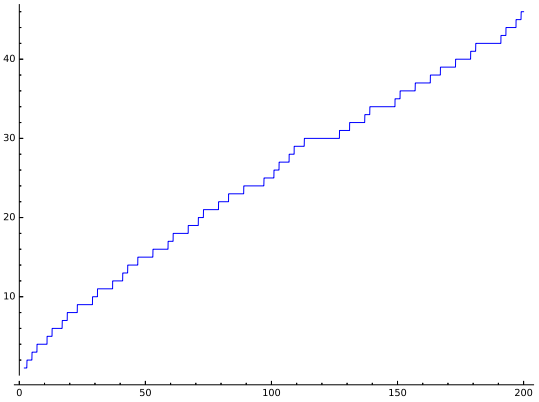
Armin Straub
straub@southalabama.edu

Example 103. Playing with the prime number theorem in Sage:

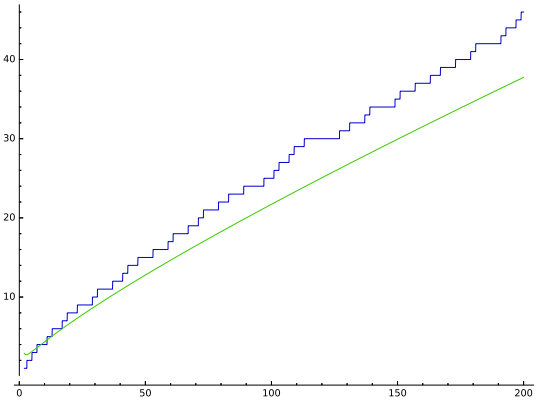
```
Sage] prime_pi(10)
```

```
4
```

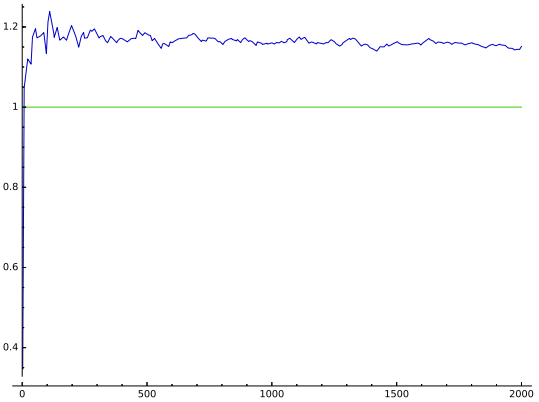
```
Sage] plot(prime_pi(x), 2, 200)
```




```
Sage] plot([prime_pi(x),x/ln(x)], 2, 200)
```




```
Sage] plot([prime_pi(x)/(x/ln(x)), 1], 2, 2000)
```



Comment. As the final plot suggests, the quotient of $\pi(x)$ and $x/\ln(x)$ indeed approaches 1 from above. This is slightly stronger than the PNT, which only claims that the quotient approaches 1.

In particular, as the previous plot suggests, for large x, $x/\ln(x)$ is always an underestimate for $\pi(x)$ (though looking at a plot like this can be very misleading).

Example 104. In 12/2018, a new largest (proven) prime was found: $2^{82,589,933}-1$.

https://www.mersenne.org/primes/?press=M82589933

This is a **Mersenne prime** (like the last 17 record primes). It has a bit over 24.8 million (decimal) digits (versus 23.2 for the previous record). The prime was found as part of GIMPS (Great Internet Mersenne Prime Search), which offers a $3,000 award for each new Mersenne prime discovered.

The EFF (Electronic Frontier Foundation) is offering $150,000 (donated anonymously for that specific purpose) for the discovery of the first prime with at least 100 million decimal digits.

https://www.eff.org/awards/coop

[Prizes of $50,000 and $100,000 for primes with 1 and 10 million digits have been claimed in 2000 and 2009.]

Definition 105. A **Mersenne prime** is a prime of the form 2^n-1.

For instance. The first few Mersenne primes have exponents $2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, \ldots$ All of these exponents are primes (but not all primes work: for instance, $2^{11}-1=23\cdot 89$). See below.

Anecdote. Euler proved in 1772 that $2^{31}-1$ is prime (then, and until 1867, the largest known prime).
"$2^{31}-1$ is probably the greatest [Mersenne prime] that ever will be discovered; for as they are merely curious, without being useful, it is not likely that any person will attempt to find one beyond it." — P. Barlow, 1811

https://en.wikipedia.org/wiki/2,147,483,647

Mersenne primes give rise precisely to all even perfect numbers (numbers whose proper divisors sum to the number itself; for instance, 6 is perfect because $6=1+2+3$). Indeed, Euclid showed that, if 2^p-1 is prime, then $2^{p-1}(2^p-1)$ is perfect [$p=2$: $2\cdot 3=6$, $p=3$: $4\cdot 7=28=1+2+4+7+14$, $p=5$: $16\cdot 33=528$, ...]. It is not known whether odd perfect numbers exist.

Example 106. (geometric sum) Evaluate $1+x+x^2+\ldots+x^n$.

Solution. $(1+x+x^2+\ldots+x^n)(x-1)=x^{n+1}-1$, so that $1+x+x^2+\ldots+x^n=\frac{x^{n+1}-1}{x-1}$.

Geometric series. In particular, $\sum_{k=1}^{\infty} x^k=\lim_{n\to\infty}\frac{x^{n+1}-1}{x-1}=\frac{1}{1-x}$, provided that $|x|<1$.

Lemma 107. If $r \mid n$, then $x^r-1 \mid x^n-1$.

Proof. Write $n=rs$. It follows from $x^s-1=(x-1)(1+x+x^2+\ldots+x^{s-1})$ that
$x^{rs}-1=(x^r-1)(1+x^r+x^{2r}+\ldots+x^{r(s-1)})$. □

Corollary 108. 2^n-1 can only be prime if n is prime.

Proof. It follows from the previous lemma that, if $n=rs$ is composite, then 2^n-1 is divisible by 2^r-1 (as well as 2^s-1). □

For instance. $2^6-1=63$ is divisible by both $2^2-1=3$ and $2^3-1=7$.

Armin Straub
straub@southalabama.edu