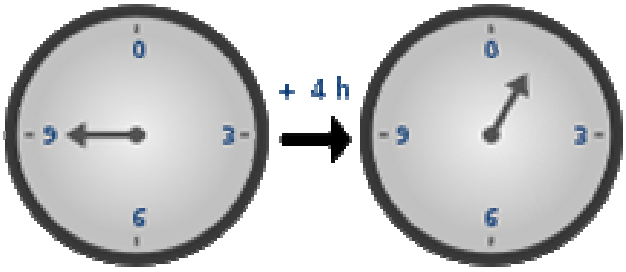
(1)

Modular arithmetic

In mathematics, **modular arithmetic** (sometimes called **clock arithmetic**) is a system of arithmetic for integers, where numbers "wrap aroundحول يلتف " after they reach a certain value the **modulus**.

[The Swiss mathematician Leonhard Euler pioneered the modern approach to congruence in about 1750, when he explicitly introduced the idea of congruence modulo a number *N*.]

[Modular arithmetic was further advanced by Carl Friedrich Gauss in his book *Disquisitiones Arithmeticae*, published in 1801.]



Time-keeping on this clock uses arithmetic modulo 12.

Simple Example:

A familiar use of modular arithmetic is in the 12-hour clock, in which the day is divided **into two 12-hour** periods. If the time is 7:00 now, then 8 hours later it will be 3:00. Usual addition would suggest that the later time should be 7 + 8 = 15, but this is not the answer because clock time "wraps around" every 12 hours; in 12-hour time, there is no "15 o'clock". Likewise, if the clock starts at 12:00 (noon) and 21 hours elapse, then the time will be 9:00 the next day, rather than 33:00. Since the hour number starts over after it reaches 12, this is **arithmetic *modulo* 12**. 12 is congruent not only to 12 itself, but also to 0, so the time called "12:00" could also be called "0:00", since 0 ≡ 12 mod 12.

Congruence relation

Modular arithmetic can be handled mathematically by introducing a congruence relation on the integers that is compatible with the operations of the **ring of integers**: addition, subtraction, and multiplication.

For a positive integer *n*, two integers *a* and *b* are said to be congruent modulo *n*, written:

$$a \equiv b \pmod{n},$$

if their difference *a* − *b* is an integer multiple of *n*. The number *n* is called the modulus of the congruence.

Example,

$$38 \equiv 2 \pmod{12}$$

because 38 − 2 = 36, which is a multiple of 12.

The **same rule holds** for negative values:

$$-8 \equiv 7 \pmod{5}.$$
$$2 \equiv -3 \pmod{5}.$$
$$-3 \equiv -8 \pmod{5}.$$

When *a* and *b* are either both positive or both negative, then $a \equiv b \pmod{n}$ can also be thought of as asserting that both *a* / *n* and *b* / *n* have the same remainder. For instance:

$$38 \equiv 14 \pmod{12}$$

because both 38 / 12 and 14 / 12 **have the same remainder, 2**. It is also the case that 38 − 14 = 24 is an **integer multiple of 12**, which agrees with the prior definition of the congruence relation.

This would have been clearer if the notation **($a \equiv_n b$)** had been used, instead of the common traditional notation.

The properties that make this relation a congruence relation (respecting **addition, subtraction, and multiplication)** are the following:

If

$$a_1 \equiv b_1 \pmod{n}$$

and

$$a_2 \equiv b_2 \pmod{n},$$

then:

- $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$
- $a_1 - a_2 \equiv b_1 - b_2 \pmod{n}$

It should be noted that the above two properties would still hold if the theory were expanded to include all real numbers, that is if a_1, a_2, b_1, b_2, n were not necessarily all integers.

The next property, however, would fail if these variables were not all integers:

- $a_1 a_2 \equiv b_1 b_2 \pmod{n}$.

Ring of congruence classes

Like any congruence relation, **congruence modulo *n*** is an equivalence relation, and the equivalence class of **the integer *a***, denoted by $\overline{a}_n$, is the set:

$$\{\ldots, a-2n, a-n, a, a+n, a+2n, \ldots\}.$$

This set, consisting of the integers congruent to ***a* modulo *n***, is called the "**congruence class**" or "**residue class**" or simply "**residue**" of the integer *a*, modulo *n*.

When the modulus *n* is known from the context, that **residue** may also be denoted $[a]$.

The **set of all congruence classes modulo *n*** is denoted $\mathbb{Z}/n\mathbb{Z}$ or $\mathbb{Z}/n$

When $n \neq 0$, $\mathbb{Z}/n\mathbb{Z}$ has *n* elements, and can be written as:

$$\mathbb{Z}/n\mathbb{Z} = \{\overline{0}_n, \overline{1}_n, \overline{2}_n, \ldots, \overline{n-1}_n\}.$$

We can define addition, subtraction, and multiplication on $\mathbb{Z}/n\mathbb{Z}$ by the following rules:

- $\overline{a}_n + \overline{b}_n = \overline{(a+b)}_n$
- $\overline{a}_n - \overline{b}_n = \overline{(a-b)}_n$
- $\overline{a}_n \overline{b}_n = \overline{(ab)}_n$.

The verification that this is a proper definition uses the properties given before.

In this way, $\mathbb{Z}/n\mathbb{Z}$ becomes a **commutative ring.**

Example,

in the ring $\mathbb{Z}/24\mathbb{Z}$, we have

$$\overline{12}_{24} + \overline{21}_{24} = \overline{9}_{24}$$

as in the arithmetic for the 24-hour clock.

The set $\mathbb{Z}/n\mathbb{Z}$ has a number of important mathematical properties that are foundational to various branches of mathematics.

Remainders

The notion of modular arithmetic is related to that of the remainder in division. The operation of finding the remainder is sometimes referred to as the modulo operation and we may see 2 = 14 (mod 12).

The difference is in the use of **congruency**, indicated by "≡", and **equality** indicated by "=". **Equality** implies specifically the "common residue", the least non-negative member of an equivalence class. When working with modular arithmetic, each equivalence class is usually represented by its common residue,

For example

38 ≡ 2 (mod 12) which can be found using long division. It follows that, while it is correct to say 38 ≡ 14 (mod 12), and 2 ≡ 14 (mod 12), it is incorrect to say 38 = 14 (mod 12) (with "=" rather than "≡").

The difference is clearest **when dividing a negative number**, since in that case remainders are negative. Hence to express the remainder we would have to write −5 ≡ −17 (mod 12), rather than 7 = −17 (mod 12), since equivalence can only be said of common residues with the same sign.

In computer science, it is the remainder operator that is usually indicated by either "%" (e.g. in C, Java, Javascript, Perl and Python) or "mod" (e.g. in Pascal, BASIC, SQL, Haskell), with exceptions (e.g. Excel).

These operators are commonly pronounced as "mod", but it is specifically a remainder that is computed (since in C++ negative number will be returned if the first argument is negative, and in Python a negative number will be returned if the second argument is negative). The function *modulo* instead of *mod*, like 38 ≡ 14 (modulo 12) is sometimes used to indicate the common residue rather than a remainder (e.g. in Ruby).

Parentheses are sometimes dropped from the expression, e.g. 38 ≡ 14 mod 12 or 2 = 14 mod 12, or placed around the divisor e.g. 38 ≡ 14 mod (12). Notation such as 38(mod 12) has also been observed, but is ambiguous without contextual clarification.

Functional representation of the remainder operation

The **remainder operation** can be represented using the floor function.

If $b \equiv a \pmod{n}$, where $n > 0$, then if the remainder b is calculated

$$b = a - \left\lfloor \frac{a}{n} \right\rfloor \times n,$$

Where $\left\lfloor \frac{a}{n} \right\rfloor$ is the largest integer less than or equal to $\frac{a}{n}$, then

$$a \equiv b \pmod{n} \text{ and,}$$
$$0 \leq b < n.$$

If instead a remainder b in the range $\{ -n \leq b < 0 \}$ is required, then

$$b = a - \left\lfloor \frac{a}{n} \right\rfloor \times n - n.$$

Residue systems

Each residue class modulo n may be represented by any one of its members, although we usually represent each residue class by the smallest nonnegative integer which belongs to that class (since this is the proper remainder which results from division). Note

that any two members of different residue classes modulo n are incongruent modulo n. Furthermore, every integer belongs to one and only one residue class modulo n.

The set of integers {0, 1, 2, ..., n - 1} is called the **least residue system modulo *n***. Any set of n integers, no two of which are congruent modulo n, is called a **complete residue system modulo *n***.

It is clear that the least residue system is a complete residue system, and that a complete residue system is simply a set containing precisely one representative of each residue class modulo n.[4] The least residue system modulo 4 is {0, 1, 2, 3}. Some other complete residue systems modulo 4 are:

- {1,2,3,4}
- {13,14,15,16}
- {-2,-1,0,1}
- {-13,4,17,18}
- {-5,0,6,21}
- {27,32,37,42}

Some sets which are *not* complete residue systems modulo 4 are:

- {-5,0,6,22} since 6 is congruent to 22 modulo 4.
- {5,15} since a complete residue system modulo 4 must have exactly 4 incongruent residue classes.

Reduced residue systems

Any set of $\varphi(n)$ integers that are **relatively prime to *n*** and that are mutually incongruent modulo n, where $\varphi(n)$ denotes **Euler's totient function**, is called a **reduced residue system modulo *n***.

The example above, {5,15} is an example of a **reduced residue system modulo 4**.

Applications

- Modular arithmetic is referenced in number theory, group theory, ring theory, knot theory, abstract algebra, cryptography, computer science, chemistry and the visual and musical arts.

- It is one of the foundations of number theory, touching on almost every aspect of its study, and provides key examples for group theory, ring theory and abstract algebra.

- Modular arithmetic is often used to calculate checksums that are used within identifiers - International Bank Account Numbers (IBANs) for example make use of modulo 97 arithmetic to trap user input errors in bank account numbers.

- In cryptography, modular arithmetic directly underpins public key systems such as RSA and Diffie-Hellman, as well as providing finite fields which underlie elliptic curves, and is used in a variety of symmetric key algorithms including AES, IDEA, and RC4.

- In computer science, modular arithmetic is often applied in bitwise operations and other operations involving fixed-width, cyclic data structures. The modulo operation, as implemented in many programming languages and calculators, is an application of modular arithmetic that is often used in this context. XOR is the sum of 2 bits, modulo 2.

- In chemistry, the last digit of the CAS registry number (a number which is unique for each chemical compound) is a check digit, which is calculated by taking the last digit of the first two parts of the CAS registry number times 1, the next digit times 2, the next digit times 3 etc., adding all these up and computing the sum modulo 10.

- In music, arithmetic modulo 12 is used in the consideration of the system of twelve-tone equal temperament, where octave and enharmonic equivalency occurs (that is, pitches in a 1□2 or 2□1 ratio are equivalent, and C-sharp is considered the same as D-flat).

- The method of casting out nines offers a quick check of decimal arithmetic computations performed by hand. It is based on modular arithmetic modulo 9, and specifically on the crucial property that $10 \equiv 1 \pmod 9$.

- Arithmetic modulo 7 is especially important in determining the day of the week in the Gregorian calendar. In particular, Zeller's congruence and the doomsday algorithm make heavy use of modulo-7 arithmetic.

- More generally, modular arithmetic also has application in disciplines such as law (see e.g., apportionment), economics, (see e.g., game theory) and other areas of the social sciences, where proportional division and allocation of resources plays a central part of the analysis.

Computational complexity

Since modular arithmetic has such a wide range of applications, it is important to know how hard it is to solve a system of congruences. A **linear system of congruences** can be solved in polynomial time with a form of Gaussian elimination, for details see linear congruence theorem. Algorithms, such as **Montgomery reduction**, also exist to allow simple arithmetic operations, such as multiplication and exponentiation modulo *n*, to be

performed efficiently on large numbers. Solving a system of non-linear modular arithmetic equations is NP-complete.

********** **********