4b. Number theory: modular arithmetic

Grzegorz Kosiorowski

Krakow University of Economics

1. Modular arithmetic
2. Systems of linear congruences
3. Chinese remainder theorem
4. Euler's totient function

Modular arithmetic - motivation

Standard arithmetics of integers is sometimes not sufficient for modelling some processes (and presenting them in the form of algorithms). In particular, this is the case for cyclical phenomena such as time calculations. For example, let us consider a 24-hour clock. If we go to sleep at 21 for 9 hours, we are going to wake up at 6 o'clock, not 30 $(21 + 9)$ o'clock. Analogously, if it is 3 o'clock (at night) and someone asks what time was 5 hours ago, we do not answer (-2) $(3 - 5)$, but 22 o'clock. This is because we are not interested in the result of integer addition/subtraction but in its remainder from integer division by 24.
The system of calculations based on remainders describing cyclical phenomena is called the *modular arithmetic*.

Congruence modulo

Congruence

Two integers a and b are *congruent* to each other *modulo* n, if their difference $a - b$ is a multiple of n (or, equivalently, if the result of integer divisions of a and b by n have the same remainder). We denote $a \equiv b(mod\ n)$ or $a \equiv_n b$. n is called a *modulus* for this relation.

This notation is intentionally similar to the notation for the remainder. If $a = b \mod n$, then $a \equiv b(mod\ n)$. The inverse implication is not necessarily true, because in the first case $a < n$, but in the second a is any: for example $8 \equiv 13(mod\ 5)$, although $13 \mod 5 = 3$).

Congruence as equivalence

Congruence as equivalence

For all a, b, c and $n > 0$ it holds that:
a) $a \equiv_n a$,
b) $a \equiv_n b \Leftrightarrow b \equiv_n a$,
c) if $a \equiv_n b$ and $b \equiv_n c$, then $a \equiv_n c$.

The relation of congruence modulo n divides $\mathbb{Z}$ into disjoint subsets (so-called residue classes) in such a way that any two numbers belong to the same subset if and only if they are congruent to each other modulo n. At the same time, any two numbers from different residue classes are not congruent to each other modulo n. So, the residue classes generated by the congruence modulo n relation are sets of numbers which give the same remainder from division by n. Thus, we can represent an entire residue class as one number: a remainder from division by n.

Properties of congruence

Properties of congruence

For any a, b, c, d and $n > 0$ it holds that:
a) if $a \equiv_n b$ and $c \equiv_n d$, then $a + c \equiv_n b + d$,
b) if $a \equiv_n b$ and $c \equiv_n d$, then $a - c \equiv_n b - d$,
c) if $a \equiv_n b$ and $c \equiv_n d$, then $ac \equiv_n bd$.

For example, let $a \equiv_{17} 5$ i $b \equiv_{17} 3$. Then $a + b \equiv_{17} 8$, $a - b \equiv_{17} 2$, $ab \equiv_{17} 15$.

Modular arithmetic

In line with the properties stated before, we can define *congruences*, namely operations defined for residue classes of congruence relation (namely, for remainders) in the same way as for integers. As the algebraic operations work in the same way for each number of the residue class, we may just use a representative of each class (the most convenient is a remainder of the division by n) in our calculations. So, instead of writing $a + b \equiv_6 ?$, where $a \equiv_6 3$ and $b \equiv_6 5$, we may write simply:
$3 + 5 \equiv_6 2$ (because when we add **any** number that is congruent to 3 modulo 6 and **any** number that is congruent to 5 modulo 6, we always obtain a number that is congruent to 2 modulo 6) .
Analogously, $3 - 5 \equiv_6 4$, $3 \cdot 5 \equiv_6 3$.
The set of remainders of the division by n is denoted as $\mathbb{Z}_n = \{0, 1, \ldots, n\}$.

Modular arithmetic on $\mathbb{Z}_4$

The "addition table" in $\mathbb{Z}_4$:

$+$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

The "multiplication table" in $\mathbb{Z}_4$:

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Naturally, we can define subtraction as an inverse operation to addition. However, the division on $\mathbb{Z}_n$ might be not that simple. For example, in $\mathbb{Z}_4$ the division (namely, the inverse operation to multiplication) sometimes gives no result: (e.g. $3 : 2$) and sometimes gives multiple results (e.g. $2 : 2$).

Modular cancellation rule

Modular cancellation rule

For $n > 0$, if $ad \equiv bd (mod\ n)$ and $d \perp n$, then $a \equiv b (mod\ n)$.

The rule determines when we can correctly divide both sides of a congruence in modulus n arithmetics: the correctness is guaranteed if and only if n and the divisor are coprime.
For example, from $4 \cdot 4 \equiv_8 2 \cdot 4 (\equiv_8 0)$ we cannot infer $4 \equiv_8 2$, because 4 and 8 are not coprime.
Therefore, for any future calculations on $\mathbb{Z}_n$, we should avoid dividing both sides of a congruence by the same number, unless we are absolutely sure that it works (namely, unless we checked that the divisor and the modulus are coprime).

Systems of linear congruences

Linear congruences and systems of linear congruences

A *linear congruence* is a congruence of a form $ax \equiv_n b$ such that $a, b \in \mathbb{Z}$, and $x \in \mathbb{Z}_n$ is the unknown. $y \in \mathbb{Z}_n$ is a solution of this congruence if the congruence becomes true after substituting y in place of x.

A *system of k linear congruences* is a set of k congruences of the form $Ax \equiv_n b$, where A is a matrix $k \times k$ with coordinates from $\mathbb{Z}$, $b \in \mathbb{Z}^k$ is a vector of k integers, and $x \in \mathbb{Z}_n^k$ is a vector of unknowns. A vector $y \in \mathbb{Z}_n^k$ is a solution of this system of congruences if each of the congruences of the system becomes true after substituting y in place of x.

Solving congruences and systems of congruences

We solve congruences and systems of congruences in the same way as equations and systems of equations in real numbers. However, there are a few exceptions:

- For congruences of modulus n, we use only natural numbers from the interval $[0, n-1]$. For convenience, we may use different integers in intermediate calculations but we should go back to the correct interval as soon as possible changing numbers outside of this interval into equivalent representatives.
- In line with the modular cancellation rule, we may divide both sides of congruences modulo n only by coprimes of n.

Solving congruences and systems of congruences

We solve congruences and systems of congruences in the same way as equations and systems of equations in real numbers. However, there are a few exceptions:

- Also, we should not multiply both sides of congruences by the same number which is not coprime with n. Although the solutions of the original congruence stay as solutions of the congruence after such multiplication, the opposite is not true. The multiplicated congruence may have "solutions" which were not solutions of the original congruence.

Example: $3x \equiv_4 3$. Obviously, $x \equiv_4 1$ is the only solution (by the modular cancellation rule, we can divide both sides of congruence modulo 4 by 3). If we were to multiply both sides of this congruence by 2 (not coprime with 4), we would get $6x \equiv_4 6 \Leftrightarrow 2x \equiv_4 2$, which has two solutions $x \equiv_4 1$ i $x \equiv_4 3$. The second solution of the last congruence is not a solution of the original one.

Linear congruence: example

Task

Solve

$$7x \equiv_{10} 6$$

We work in the set $\mathbb{Z}_{10}$, therefore, we aim to use only integers from 0 to 9 and we cannot use fractions! To solve this congruence, it suffices to divide both sides by 7. Are we allowed to do so? Yes, by the cancellation rule and the fact that $7 \perp 10$!

Linear congruence: example

Task

$$7x \equiv_{10} 6$$

To divide 6 by 7 in Z_{10}, we look for a number in the same residue class as 6 in arithmetic modulo 10, which is divisible by 7 in integer arithmetics. To stay in the same residue class modulo 10 i may always add 10, so I check sequentially: 16, 26, 36, 46, and while these numbers are equivalent to 6, they are not divisible by 7. Finally, 56 works, because $7 \cdot 8 \equiv_{10} 6$, thus $6 : 7 \equiv_{10} 8$. Therefore, $x \equiv_{10} 8$ is a solution of this congruence.

System of linear congruences: example

Task

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 9x - 4y \equiv_{13} 10. \end{cases}$$

We solve it in $\mathbb{Z}_{13}$ - so only integers from 0 to 12 are allowed as solutions. We can never use fractions!
Of course, there are multiple ways of solving this sytem. The solution that follows is not the simplest but easy to generalize and very educational.

System of linear congruences: example

Task

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 9x - 4y \equiv_{13} 10. \end{cases}$$

We try to get rid of one unknown, say x, by subtracting both sides of two congruences (this operation is always allowed). Therefore, I transform the second congruence in such a way that on the right side there was $3x$ instead of $9x$. It should be simple: we need only to divide both sides by 3. Is it allowed? Yes, because 3 and 13 are coprime! However, numbers $\frac{4}{3}$ and $\frac{10}{3}$ do not exist in $\mathbb{Z}_{13}$.

System of linear congruences: example

Task

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 9x - 4y \equiv_{13} 10. \end{cases}$$

To divide 4 by 3 modulo 13, we look for a number equivalent to 4 in arithmetic modulo 13, divisible by 3. For example, it can be 30 (because $30 = 2 \cdot 13 + 4$) thus $3 \cdot 10 \equiv_{13} 4$. Therefore $4 : 3 \equiv_{13} 10$ ($3 \perp 13$, so we can divide).

To divide 10 by 3 modulo 13, we look for a number equivalent to 10 in arithmetic modulo 13, divisible by 3. For example, it can be 36 (because $36 = 2 \cdot 13 + 10$), thus $3 \cdot 12 \equiv_{13} 10$, and $10 : 3 \equiv_{13} 12$ ($3 \perp 13$, so we can divide).

System of linear congruences: example

Task

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 9x - 4y \equiv_{13} 10. \end{cases}$$

Thus the above system of congruences can be transformed into:

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 3x - 10y \equiv_{13} 12. \end{cases}$$

System of linear congruences: example

Task

$$\begin{cases} 3x - 5y \equiv_{13} 1 \\ 3x - 10y \equiv_{13} 12. \end{cases}$$

Now we subtract both sides of congruences from the system $(1 - 12 \equiv_{13} 2)$ to obtain

$$5y \equiv_{13} 2.$$

Thus, $y \equiv_{13} 3$ (because $5 \cdot 3 = 15 \equiv_{13} 2$), and $x \equiv_{13} 1$ (easy to calculate after substituting 3 in place of y to any of the congruences of the system). It is always wise to check if the obtained pair of numbers is a solution by substituting them into the original system!

Congruences: existence of solutions

As it happened with equations and systems of equations, the natural question arises: do linear congruences and systems of linear congruences always have unique solutions? In both cases, the answer is negative: it is possible that a congruence or a system of congruences has no solutions, one solution or multiple (but not infinitely many! - this is different from the case of linear equations) solutions.

For example, a congruence $2x \equiv_4 3$ has no solutions and a congruence $2x \equiv_4 2$ has two solutions.

Congruences: existence of solutions

The system of congruences:

$$\begin{cases} 3x + y \equiv_{17} 1 \\ x + 6y \equiv_{17} 2 \end{cases}$$

has no solutions, and the system of congruences:

$$\begin{cases} 3x + y \equiv_{17} 1 \\ x + 6y \equiv_{17} 6 \end{cases}$$

has 17 solutions.

Congruences: existence of solutions

The Rouche-Capelli theorem provides a full answer to questions about existence and uniqueness of solutions for systems of linear equation. The analogue for systems of congruences is much more complex, thus we mention only the following partial results:

Existence of a solution for a linear congruence

A linear congruence $ax \equiv_n b$ admits at least one solution if and only if $GCD(a, n)|b$.

A solution for a system of linear congruences

If $\det A \perp n$, then a system of linear congruences $Ax \equiv_n b$ has a unique solution.

Modular arithmetic: applications

The idea of checksums is a simple but powerful application of modular arithmetic. A checksum is a relatively small number (or other block of data) derived from a larger set of data (e.g. identification numbers such as bank account numbers) for the purpose of detecting errors.
For example, assume that we need to assign account numbers in our bank to 50000 customers. Theoretically, 5-digit numbers should be sufficient. However, trying to use such a system, it would be easy to make a mistake in one digit and transfer money to an incorrect account. Fixing such mistakes might be costly and time-consuming. Thus, it would be better to prevent them.

Modular arithmetic: checksums

Thus, the checksums or (in this case) "check digits", which are commonly used in such identification numbers, appear. A part of such a number does not work as a identifier but has a function confirming the validity of the entire number. For example, we may add to our account number a 6-th digit defined as follows: we compute the sum of the first five digits and the 6-th digit will be equal to the remainder of division of this sum by 10. For example, for a customer numbered 39605 we would calculate a sum of digits equal to 23, so, accounting for the remainder of division of 23 by 10 the number of his account should be:

$$396053.$$

Modular arithmetic: checksums

We compute the sum of the first five digits and the 6-th digit will be equal to the remainder of division of this sum by 10. For example, for a customer numbered 39605 we would calculate a sum of digits equal to 23, so, accounting for the remainder of division of 23 by 10 the number of his account should be:

$$396053.$$

Now, miswriting one digit of the number would be impossible. Assume that someone trying to transfer money to this customer incorrectly sends a transfer to an account of 396653 (changing the 4-th digit from 0 to 6). Our system would immediately detect that there is a mistake, because the sum of digits $3, 9, 6, 6, 5$ gives a remainder of 9, not 3, from division by 10.

Modular arithmetic: checksums

The procedure from the previous slide was obviously simplified and thus relatively ineffective (for example it would not detect swapping two digits in the number). The real-life checksums based on the rules of modular arithmetic are more complex:

- International Standard Book Number (ISBN) uses arithmetics modulo 11 (for 10-digit numbers) or modulo 10 (for 13-digits).
- International Bank Account Number (IBAN) uses arithmetics modulo 97.
- Chemical Abstract Service (CAS)uses arithmetics modulo 10.
- Cryptographic hashing functions - a basis for e.g. cryptocurrencies.

Modular arithmetic: integer overflow

In computer memory, there are no "true" integers: variables that seem to be integers, in fact belong to $\mathbb{Z}_n$. This is a source of errors known as *integer overflows*.
It is possible that a value of such a variable changes and when it is set to a maximum $(n-1)$ or minimum (0) value, its next increment (or reduction) may lead to an unexpected and drastic change: from large to low values or the other way around.
Examples of problems: first flight of a rocket of Ariane 5 type (1996), a legend of "nuclear Gandhi".

Chinese remainder theorem

The following theorem is used for solving systems of congruences of different moduli:

Chinese Remainder Theorem, Sun Zi, 2nd century

Let $n_1, \ldots n_k \in \mathbb{N}$ be such that $n_i \perp n_j$ for $i \neq j$. Then, for any $a_1, \ldots, a_k$ there exists a unique number $x < n_1 \cdot \ldots \cdot n_k$ such that

$$\begin{cases} x \equiv_{n_1} a_1 \\ x \equiv_{n_2} a_2 \\ \quad \ldots \\ x \equiv_{n_k} a_k \end{cases}.$$

This time this theorem is "constructive". Namely, there exists an algorithm finding the solution x.

Chinese remainder algorithm

We are to solve a system of congruences:

$$\begin{cases} x \equiv_{n_1} a_1 \\ x \equiv_{n_2} a_2 \\ \quad \ldots \\ x \equiv_{n_k} a_k \end{cases}.$$

Chinese remainder algorithm

Input: Positive integers: $n_1, \ldots, n_k$ (coprime to each other), $a_1, \ldots, a_k$.
Variables: N - integer, $(N_i)_{i=1}^k$ - a finite sequence of integers, x - output (integer).

Chinese remainder algorithm

Chinese remainder algorithm

Input: Positive integers: $n_1, \ldots, n_k$ (coprime to each other), $a_1, \ldots, a_k$.

Variables: N - integer, $(N_i)_{i=1}^k$ - a finite sequence of integers, x - output (integer).

- I. $N := \prod_{i=1}^k n_i$.
- II. For each $i \in \{1, \ldots, k\}$ we define $N_i := N/n_i$. Naturally, $\mathrm{GCD}(n_i, N_i) = 1$.
- III. For each $i \in \{1, \ldots, k\}$ we find x_i such that $N_i x_i \equiv_{n_i} 1$ (because $n_i \perp N_i$).
- IV. $x := \sum_{i=1}^k a_i x_i N_i \mod N$.
- **Output**: x is the number from the Chinese Remainder Theorem.

Remark on the Chinese remainder algorithm

III. For each $i \in \{1, \ldots, k\}$ we find x_i such that $N_i x_i \equiv_{n_i} 1$ (because $n_i \perp N_i$).
How to conduct this step? Usually, just guessing x_i is quite easy (in the same way as we usually solve linear congruences). In case when it is difficult (for large numbers), we may apply the extended Euclidian algorithm. We know that $\mathrm{GCD}(n_i, N_i) = 1$. Thus, by the extended Euclidian algorithm we can find x_i, y_i such that
$x_i N_i + y_i n_i = GCD(n_i, N_i) = 1$,thus $N_i x_i = -y_i n_i + 1$, and finally $N_i x_i \equiv_{n_i} 1$.

Chinese remainder algorithm - example

Task

The commander of the palace guard of an emperor prepares his unit for an inspection. He tried to arrange them into rows of threes, but there were 2 extra soldiers left. When he tried to arrange them into rows of sevens, there were 4 soldiers left. When he tried to arrange them into rows of eights, there were 5 soldiers left. How many soldiers were in the palace guard if there were at most 168 of them?

Mathematically, we need to solve the following system of congruences:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

Chinese remainder algorithm - example

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

$3, 7$ i 8 are coprime to each other so we can apply the Chinese remainder algorithm. First, we calculate $N = 3 \cdot 7 \cdot 8 = 168$. By the Chinese remainder theorem, there exists $x < 168$ satisfying this system.

Chinese remainder algorithm - example

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

For stage II we calculate:

$$N_1 = \frac{N}{n_1} = \frac{168}{3} = 56.$$

$$N_2 = \frac{N}{n_2} = \frac{168}{7} = 24.$$

$$N_3 = \frac{N}{n_3} = \frac{168}{8} = 21.$$

Chinese remainder algorithm - example

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

For stage III we calculate:

$$56x_1 \equiv_3 1 \rightarrow x_1 = 2.$$

$$24x_2 \equiv_7 1 \rightarrow x_2 = 5.$$

$$21x_3 \equiv_8 1 \rightarrow x_3 = 5.$$

Chinese remainder algorithm - example

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

For step IV we calculate:

$$x = a_1 x_1 N_1 + a_2 x_2 N_2 + a_3 x_3 N_3 \mod 168 =$$

$$= 2 \cdot 2 \cdot 56 + 4 \cdot 5 \cdot 24 + 5 \cdot 5 \cdot 21 \mod 168 = 1229 \mod 168 = 53.$$

Therefore, there were 53 soldiers in the palace guard.

Chinese problem - alternative approach

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

There exist another approach to solve the same problem. It is more convenient for "human" calculations, but less efficient for large numbers and many equations.
We solve cogruences one by one. For the first: $x = 3i + 2$ for some i.
Substituting this into the second, we obtain:

$$3i + 2 \equiv_7 4 \Rightarrow 3i \equiv_7 2 \Rightarrow i \equiv_7 3.$$

Chinese problem - alternative approach

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

$i \equiv_7 3$, thus $i = 7j + 3$ for some j. We substitute this into $x = 3i + 2$ obtaining $x = 3(7j + 3) + 2 = 21j + 11$ for some j. We substitute it to the last congruence to obtain:

$$21j + 11 \equiv_8 5 \Rightarrow 5j + 3 \equiv_8 5 \Rightarrow 5j \equiv_8 2 \Rightarrow j \equiv_8 2.$$

Chinese problem - alternative approach

Task

Solve:

$$\begin{cases} x \equiv_3 2 \\ x \equiv_7 4 \\ x \equiv_8 5 \end{cases}$$

$j \equiv_8 2$, thus $j = 8k + 2$ for some k. Thus, $x = 21j + 11$ can be transformed into $x = 21(8k + 2) + 11 = 168k + 53$ for some k. Adding that $x < 168$ we obtain $x = 53$.

Chinese remainder algorithm: applications

- Some "secret sharing protocols" (e.g. Mignotte's and Asmuth-Bloom's Schemes): a "secret message" is encrypted and divided into pieces. Each piece is given to one individual in a form of one congruence. Only when all (or at least, a majority) of individuals share their parts of the "secret", the entire message can be recreated.
- Fast Fourier Transform (for signal processing).
- *Range ambiguity resolution*: recovering information about the correct range of objects found by a radar.

Euler's totient function - definition

Euler's totient function

Euler's totient function or Euler's φ function $\varphi : \mathbb{N} \setminus \{0\} \longrightarrow \mathbb{N}$ is defined as follows:

$$\varphi(n) = \left|\{1 \leqslant a \leqslant n \quad : \quad \mathrm{GCD}(a, n) = 1\}\right|,$$

namely, φ maps a natural number n into the amount of positive integer that are no larger than n and coprime to n.

For example, we calculate $\varphi(6)$. Positive integers that are no larger and coprime to 6 are: 1 i 5, so $\varphi(6) = 2$.
Generally, we do not calculate $\varphi(n)$ this way.

Euler's totient function - computation

Euler's totient function for primes

For any prime p it holds that:
a) $\varphi(p) = p - 1$
b) $\varphi(p^k) = p^k(1 - \frac{1}{p})$.

Euler's totient function for products of coprimes

For any two positive coprimes m and n it holds that:
$\varphi(mn) = \varphi(m)\varphi(n)$.

Conclusion: we are able to calculate a value of the Euler's totient for any number, as long as we have a prime factorization of that number.

Euler's totient function - example

Task

Find $\varphi(600)$

We can factorize $600 = 2^3 \cdot 3 \cdot 5^2$. Naturally, 2^3, 3 and 5^2 are pairwise coprime, thus we apply both theorems of the previous slide:

$$\varphi(600) = \varphi(2^3 \cdot 3 \cdot 5^2) = \varphi(2^3) \cdot \varphi(3) \cdot \varphi(5^2) = 2^3(1-\frac{1}{2}) \cdot 2 \cdot 5^2(1-\frac{1}{5}) = 160.$$

Euler's totient function: computation

It seems that we have a simple procedure for computing the Euler's totient for any natural number. However, this procedure is practically applicable only for numbers which can be easily factorized into a product of primes. As we already know, factorization is generally difficult.

Unfortunately (or: fortunately, for many encryption algorithms), a simpler (and less computationally complex) way of finding the totient than factorizing and using theorems about the totient is not known. Therefore, one can say that calculating the totient is as complex as the prime factorization (namely, of non-polynomial complexity).

Euler's Theorem and Fermat's Little Theorem

Euler's Theorem

If $a \perp n$, then $a^{\varphi(n)} \equiv_n 1$.

As a simple conclusion one may obtain:

Fermat's Little Theorem

For any prime p and any positive integer n it holds that $n^p \equiv_p n$.

Both theorems are very useful for cryptology as they simplify modular arithmetic for large numbers.

Euler's Theorem - example

Task

Calculate $19^{74} \mod 28$

19^{74} is a gigantic number and any "brute force" calculations (even with computers' help) would take a very long time.

Euler's Theorem

If $a \perp n$, then $a^{\varphi(n)} \equiv_n 1$.

Naturally, $19 \perp 28$ (because 19 is prime), thus we can apply Euler's theorem. $\varphi(28) = \varphi(2^2 \cdot 7) = 2 \cdot 6 = 12$. Thus, $19^{12} \equiv_{28} 1$.
$19^{74} = (19^{12})^6 \cdot 19^2$, so
$19^{74} \equiv_{28} (19^{12})^6 \cdot 19^2 \equiv_{28} (1)^6 \cdot 361 \equiv_{28} 361 \equiv_{28} 25$.