香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Mathematics in OI (I)

Daniel Hsieh {QwertyPi}

2024-03-09

Why Mathematics?

- We are dealing with numbers in our programs every day
  - Even characters are ASCII numbers!
- In OI, mathematics are everywhere
  - Many OI problems require mathematical knowledge to solve
  - The time complexity may be reduced by using some formulas
- Maths in OI (I): mainly cover number theory
- Maths in OI (II): mainly cover combinatorics

Table of Contents

GREEK ALPHABET

By Ben Crowder • bencrowder.net • Last modified 3 May 2012

Αα	Ββ	Γγ	Δδ	Εε	Ζζ
ALPHA [a] *ἄλφα*	**BETA** [b] *βῆτα*	**GAMMA** [g] *γάμμα*	**DELTA** [d] *δέλτα*	**EPSILON** [e] *ἒ ψιλόν*	**ZETA** [dz] *ζῆτα*
Ηη	Θθ	Ιι	Κκ	Λλ	Μμ
ETA [ɛ:] *ἦτα*	**THETA** [tʰ] *θῆτα*	**IOTA** [i] *ἰῶτα*	**KAPPA** [k] *κάππα*	**LAMBDA** [l] *λάμβδα*	**MU** [m] *μῦ*
Νν	Ξξ	Οο	Ππ	Ρρ	Σσς
NU [n] *νῦ*	**XI** [ks] *ξεῖ*	**OMICRON** [o] *ὂ μικρόν*	**PI** [p] *πεῖ*	**RHO** [r] *ῥῶ*	**SIGMA** [s] *σῖγμα*
Ττ	Υυ	Φφ	Χχ	Ψψ	Ωω
TAU [t] *ταῦ*	**UPSILON** [u] *ὐ ψιλόν*	**PHI** [pʰ] *φεῖ*	**CHI** [kʰ] *χεῖ*	**PSI** [ps] *ψεῖ*	**OMEGA** [ɔ:] *ὦ μέγα*

Some Notations and Abbreviations

- $\forall$: for all
- $\exists$: there exists
- $s.t.$: such that

- $\mathbb{N}$: set of natural numbers
- $\mathbb{Z}$: set of integers
- $\mathbb{Q}$: set of rational numbers
- $\mathbb{R}$: set of real numbers

- $\in$: is an element of
- $\cap$: set intersection
- $\cup$: set union

- $\wedge$: logical AND
- $\vee$: logical OR
- $\Rightarrow$: implies
- $\iff$: if and only if (iff)

Summation Sign Σ

Sigma: σ (lower case), Σ (upper case)
For any integers $l \leq r$,

$$\sum_{k=l}^{r} a_k = a_l + a_{l+1} + \cdots + a_r$$

Example:

$$\sum_{k=3}^{7} k^3 = 3^3 + 4^3 + 5^3 + 6^3 + 7^3 = 775$$

Summation Sign Σ

Another Example. Compute the following:

$$\sum_{k=20}^{30} (k^2 + 4k - 7)$$

```
int sum = 0;
for (int k = 20; k <= 30; k++) {
    sum += k * k + 4 * k - 7;
}
return sum;
```

Product Sign Π

Pi: π (lower case), Π (upper case)
For any integers $l \leq r$,

$$\prod_{k=l}^{r} a_k = a_l \times a_{l+1} \times \cdots \times a_r$$

Example:

$$\prod_{k=3}^{7} (k^2 - 3) = (3^2 - 3)(4^2 - 3)(5^2 - 3)(6^2 - 3)(7^2 - 3) = 2604888$$

Product Sign Π

Probably most well-known example: Factorial, defined as

$$n! = \prod_{k=1}^{n} k = 1 \times 2 \times \cdots \times n$$

Note that $0! = 1$ by definition (in accordance with its combinatorical meaning)

In short, the summation sign (Σ) and product sign (Π) help you to express terms easier when lots of numbers are added / multiplied together.

Floor and Ceiling Function $\lfloor x \rfloor, \lceil x \rceil$

Floor function $\lfloor x \rfloor$ is defined to be the largest integer not exceeding x.
For example, $\lfloor 1 \rfloor = 1, \lfloor -2.718 \rfloor = -3, \lfloor \frac{10}{3} \rfloor = 3$.

Ceiling function $\lceil x \rceil$ is defined to be the smallest integer not less than x.
For example, $\lceil 1 \rceil = 1, \lceil -2.718 \rceil = -2, \lceil \frac{10}{3} \rceil = 4$.

Table of Contents

Divisibility Sign |

Vertical bar: |

$d \mid n$: "d divides n", "n is divisible by d" or "d is a factor of n".
$d \nmid n$ simply means the opposite, i.e., "n is not divisible by d".

Examples: $2 \mid 4, 3 \mid 6, 4 \mid 0, 6 \nmid 15$

But what does the statement "n is divisible by d" actually means?

- In fact, $d \mid n$ is equivalent to that there exists an integer k such that $n = kd$.
- Formally,

$$d \mid n \iff \exists k \in \mathbb{Z} \text{ s.t. } n = kd$$

Divisibility Properties

Suppose a, b, c, x, y are all integers:

- $a \mid b$ and $b \mid c \Rightarrow a \mid c$
- $a \mid b$ and $a \mid c \Rightarrow a \mid (b \pm c)$
- $a \mid b$ and $b \mid a \Rightarrow a = \pm b$
- $a \mid b \Rightarrow a \mid bx$
- $a \mid b \Rightarrow ax \mid bx$
- $a \mid b$ and $a \mid c \Rightarrow a \mid (bx + cy)$

All these properties can be proven using the fact that

- $d \mid n$
- there exists an integer k such that $n = kd$

are equivalent.

Table of Contents

Greatest Common Divisor (GCD) - Definition

Also known as Highest Common Factor (HCF)

Consider two integers a and b.

- If d divides both a and b, then d is a common divisor of a and b.
- Symbolic form: $d \mid a \wedge d \mid b$ or simply $d \mid a, b$

Greatest Common Divisor (GCD) - Definition

Greatest common divisor (GCD) of a and b is simply the largest among the common divisors of a and b.

- We use $\gcd(a, b)$ to represent the greatest common divisor of a and b.
- You can also use (a, b) when no confusion would arise (with coordinates).

We say two integers a, b are co-prime (or relatively prime) if $\gcd(a, b) = 1$.

Greatest Common Divisor (GCD) - Computation

How can we calculate GCD of two integers?

1. Brute force! Too slow :(
2. Use C++17! But how does it work?
3. Euclidean Algorithm $\hookleftarrow$ Our target

What do we know about GCD? What observations can we make?

Euclidean Algorithm - Concept

Observation 1

$\gcd(a, b) = \gcd(b, a)$, i.e. the order does not matter.

Observation 2

For any positive integer a, $\gcd(a, 0) = a$.

Proof. Both from definition. □

Euclidean Algorithm - Concept

Observation 3

For any integers a, b and k, $\gcd(a, b) = \gcd(a, b + ka)$ holds.

Proof. Suppose $d = \gcd(a, b)$ and $e = \gcd(a, b + ka)$, our target is to show that $d \leq e$ and $e \leq d$ separately, which together will imply that d and e are equal.

- ($d \leq e$) By definition, $d \mid a$ and $d \mid b$. Therefore, $d \mid (b + ka)$ as well, and so d is a common divisor of a and $b + ka$. This means that $d \leq e$, as e is the greatest common divisor of a and $b + ka$.
- ($e \leq d$) Similarly, $e \mid a$ and $e \mid b + ka$. Therefore, $e \mid (b + ka) - (ka) = b$ as well, and so e is a common divisor of a and b. This means $e \leq d$.

□

Euclidean Algorithm - Concept

Observation 4

For any integers $a, b (b \neq 0)$, $\gcd(a, b) = \gcd(b, a \bmod b)$ holds.

Proof. We write $a = qb + r$ where q and $0 \leq r < b$ are integers. Then $r = a \bmod b$. Hence,

$$\begin{aligned}\gcd(a, b) &= \gcd(qb + r, b) \\ &= \gcd(b, qb + r) \text{ (Observation 1)} \\ &= \gcd(b, r) \text{ (Observation 3)}\end{aligned}$$

□

Euclidean Algorithm - Procedure

We have introduced all the building blocks of Euclidean Algorithm!

Euclidean Algorithm

Given two positive integers a and b. To find their greatest common divisor, we can

- Apply the fact that $\gcd(a, b) = \gcd(b, a \bmod b)$ repeatedly until the second parameter becomes 0.
- The answer is then simply the first parameter.

Euclidean Algorithm - Procedure

The code for Euclidean Algorithm is incredibly simple:

```
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

Now, two questions for you:

1. Why is it guaranteed to terminate?
2. When does this algorithm perform the worst?

Euclidean Algorithm - Correctness

1. Why is it guaranteed to terminate?

Consider what happens in the recursive Euclidean Algorithm for $\gcd(a, b)$:

$$
\begin{array}{lll}
a = q_0 b + r_0 & 0 \le r_0 < b & \gcd(a, b) = \gcd(b, r_0) \\
b = q_1 r_0 + r_1 & 0 \le r_1 < r_0 & \gcd(b, r_0) = \gcd(r_0, r_1) \\
 & \vdots & \\
r_{k-2} = q_k r_{k-1} + r_k & 0 \le r_k < r_{k-1} & \gcd(r_{k-2}, r_{k-1}) = \gcd(r_{k-1}, r_k)
\end{array}
$$

Euclidean Algorithm - Correctness

1. Why is it guaranteed to terminate?

Notice how

$$0 \leq r_k < r_{k-1} < \cdots < r_1 < r_0 < b$$

That is, $\{r_k\}$ is strictly decreasing.

Therefore, the algorithm must eventually terminates! □

Euclidean Algorithm - Time Complexity

❷ When does this algorithm perform the worst?

Recall this recursion:

$$\begin{array}{lll} a = q_0 b + r_0 & 0 \le r_0 < b & \gcd(a,b) = \gcd(b, r_0) \\ b = q_1 r_0 + r_1 & 0 \le r_1 < r_0 & \gcd(b, r_0) = \gcd(r_0, r_1) \\ & \vdots & \\ r_{N-2} = q_N r_{N-1} + r_N & 0 \le r_N < r_{N-1} & \gcd(r_{N-2}, r_{N-1}) = \gcd(r_{N-1}, r_N) \end{array}$$

assuming the algorithm terminates after $(N+1)$ steps.
Except for the first iteration which possibly $a < b$,

$$b > r_0 > r_1 > \cdots > r_{N-1} > r_N$$

Hence, $q_0 \ge 0$ and $q_i \ge 1$ for $1 \le i \le N$.

Euclidean Algorithm - Example

a	b	r	q
2024	525		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$2024 = 525 \times 3 + 449$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$2024 = 525 \times 3 + 449$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$525 = 449 \times 1 + 76$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$525 = 449 \times 1 + 76$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$449 = 76 \times 5 + 69$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$449 = 76 \times 5 + 69$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$76 = 69 \times 1 + 7$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$76 = 69 \times 1 + 7$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$69 = 7 \times 9 + 6$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$69 = 7 \times 9 + 6$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6	1	1

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$7 = 6 \times 1 + 1$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6	1	1
6	1		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$7 = 6 \times 1 + 1$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6	1	1
6	1	0	6

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$6 = 1 \times 6 + 0$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6	1	1
6	1	0	6
1	0		

Team Formation Test
2024-05-25

Goal
Find $\gcd(2024, 525)$

$$6 = 1 \times 6 + 0$$

Euclidean Algorithm - Example

a	b	r	q
2024	525	449	3
525	449	76	1
449	76	69	5
76	69	7	1
69	7	6	9
7	6	1	1
6	1	0	6
1	0		

Team Formation Test
2024-05-25

$\gcd(2024, 525) = 1$

Euclidean Algorithm - Time Complexity

2. When does this algorithm perform the worst?

For the worst case scenario, the numbers should decrease as slow as possible.

This occurs exactly when $q_0 = 0$ and $q_i = 1$ for all $i \geq 1$.

Actual Worst Case: ??? (To be covered in Problem Set)
Time Complexity: $O(\log \min(a, b))$

Extended Euclidean Algorithm - Concept

As from the title, it is a simple extension to the Euclidean Algorithm!

Euclidean Algorithm: find $\gcd(a, b)$ of integers a, b
Extended Euclidean Algorithm: find integers x, y such that $ax + by = \gcd(a, b)$

This serves as a constructive proof for the Bézout's Lemma:

Bézout's Lemma

Let a and b are positive integers with greatest common divisor d. Then there exists integers x and y such that $ax + by = d$.

Extended Euclidean Algorithm - Concept

Observation 5

For positive integers a, b, the relation

$$a = \lfloor \frac{a}{b} \rfloor b + (a \bmod b)$$

holds.

Proof. It is just an alternative way of saying $a = qb + r$ with $0 \leq r < b$. □

Extended Euclidean Algorithm - Concept

Observation 6

Consider positive integers a, b and d. Suppose for some integers x', y',

$$bx' + (a \bmod b)y' = d$$

Then,

$$ay' + b(x' - \lfloor \frac{a}{b} \rfloor y') = d$$

Proof. Notice that $a = \lfloor \frac{a}{b} \rfloor b + (a \bmod b)$, which means

$$a \bmod b = a - \lfloor \frac{a}{b} \rfloor b$$

In which we can substitute into the equality to get the wanted result. □

Extended Euclidean Algorithm - Concept

For example, suppose $a = 17, b = 5$, which then $r = 2$ and $q = 3$.

If we know that

$$1 \times 5 + (-2) \times 2 = 1$$

By expanding $2 = 17 - 3 \times 5$ $(r = a - qb)$:

$$1 \times 5 + (-2) \times (17 - 3 \times 5) = 1$$

Which, by rearranging terms, we will obtain

$$(-2) \times 17 + (1 - (-2) \times (-3)) \times 5 = (-2) \times 17 + 7 \times 5 = 1$$

Extended Euclidean Algorithm - Procedure

Hence, we may simply add a backtrack to the recursive Euclidean Algorithm!

```
int ex_gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int x2, y2;
    int d = ex_gcd(b, a % b, x2, y2);
    x = y2, y = x2 - (a / b) * y2;
    return d;
}
```

Extended Euclidean Algorithm - Procedure

When $b = 0$:

```
if (b == 0) { // terminating state
    x = 1, y = 0; // 1 * a + 0 * b = gcd
    return a;
}
```

Extended Euclidean Algorithm - Procedure

When $b \neq 0$:

```
int x2, y2;
// known: x2 * b + y2 * (a % b) = a = gcd
int d = ex_gcd(b, a % b, x2, y2);

// hence: y2 * a + (x2 - (a / b) * y2) * b = gcd
x = y2, y = x2 - (a / b) * y2;
return d;
```

Time Complexity: $O(\log \min(a, b))$, which is same as the standard version

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1		
69	7	6	9		
7	6	1	1		
6	1	0	6		
1	0				

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1		
69	7	6	9		
7	6	1	1		
6	1	0	6		
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$1 \times 1 + 0 \times 0 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1		
69	7	6	9		
7	6	1	1		
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$6 \times 0 + 1 \times 1 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1		
69	7	6	9		
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$7 \times 1 + 6 \times -1 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1		
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$69 \times -1 + 7 \times 10 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5		
76	69	7	1	10	-11
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$76 \times 10 + 69 \times -11 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1		
449	76	69	5	-11	65
76	69	7	1	10	-11
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$$449 \times -11 + 76 \times 65 = 1$$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3		
525	449	76	1	65	-76
449	76	69	5	-11	65
76	69	7	1	10	-11
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$525 \times 65 + 449 \times -76 = 1$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3	-76	293
525	449	76	1	65	-76
449	76	69	5	-11	65
76	69	7	1	10	-11
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

Goal
Find $x, y \in \mathbb{Z}$ such that
$2024x + 525y = 1$

$$2024 \times -76 + 525 \times 293 = 1$$

Extended Euclidean Algorithm - Example

a	b	r	q	x	y
2024	525	449	3	-76	293
525	449	76	1	65	-76
449	76	69	5	-11	65
76	69	7	1	10	-11
69	7	6	9	-1	10
7	6	1	1	1	-1
6	1	0	6	0	1
1	0			1	0

Team Formation Test
2024-05-25

$$2024 \times -76 + 525 \times 293 = 1$$

Extended Euclidean Algorithm - Application

Let us state our result:

Extended Euclidean Algorithm

Given positive integers a, b. Then, we can find integers x_0, y_0 such that

$$ax_0 + by_0 = \gcd(a, b)$$

in $O(\log \min(a, b))$.

What is its significance in terms of number theory?
Let's introduce two lemmas based on it: Bézout's Lemma and Euclid's Lemma.

Extended Euclidean Algorithm - Application

Bézout's Lemma

Let a and b are positive integers with greatest common divisor d. Then there exists integers x and y such that $ax + by = d$.

Proof. Construction by Extended Euclidean Algorithm. □

Extended Euclidean Algorithm - Application

Euclid's Lemma

Let a, b and c are positive integers. Suppose $a \mid bc$ and $\gcd(a, b) = 1$, then $a \mid c$.

Proof. Since $\gcd(a, b) = 1$, by Bézout's Lemma, there exists integers x, y such that

$$ax + by = 1$$

As $a \mid a$ and $a \mid bc$, we have

$$a \mid (a)cx + (bc)y = (ax + by)c = c$$

□

Extended Euclidean Algorithm - Generalisation

Now, let's generalise the problem - Given arbitrary integers a, b, c, can we find ALL the solutions to the linear Diophantine equation $ax + by = c$?

Let's say the greatest common divisor of a and b is d.

Two-step approach:

1. Find any specific solution.
2. Generalise that specific solution.

Extended Euclidean Algorithm - Generalisation

1. Find any specific solution.
   - Solution exists if and only if $d \mid c$ (Why?)
   - By Extended Euclidean Algorithm, we can find integers x'_0, y'_0 such that

$$ax'_0 + by'_0 = d$$

   - Therefore, multiplying the whole equation by $\frac{c}{d}$, we obtain

$$a(x'_0 \cdot \frac{c}{d}) + b(y'_0 \cdot \frac{c}{d}) = d \cdot \frac{c}{d} = c$$

Extended Euclidean Algorithm - Generalisation

2. Generalise the specific solution.

- Let's say we already have $ax_0 + by_0 = c$ for some integers x_0, y_0.
- What other solutions $ax_1 + by_1 = c$ can we find?
- Notice that if we subtract the equations, we will obtain

$$a(x_1 - x_0) + b(y_1 - y_0) = 0$$

- Which is solvable if and only if $b \mid a(x_1 - x_0)$ or $\frac{b}{d} \mid \frac{a}{d}(x_1 - x_0)$
- As $\gcd(\frac{b}{d}, \frac{a}{d}) = 1$, this means $\frac{b}{d} \mid x_1 - x_0$ and $x_1 - x_0 = \frac{b}{d}k$ for some integer k!
- (Side Note: we have implicitly applied Euclid's Lemma!)

Extended Euclidean Algorithm - Generalisation

2. Generalise the specific solution.

- Therefore, all the solutions x, y to the equation $ax + by = c$ are in form of

$$x = x_0 + \frac{b}{d}k, y = y_0 - \frac{a}{d}k$$

  for any integer k.
- Note: Sometimes there are restrictions on x and y, e.g., must be non-negative. You may deduce the range of k from the inequalities.

So that's all for Euclidean Algorithm - let's move back to GCD!

Greatest Common Divisor (GCD) - Properties

Property 1

If m is a common divisor of a and b, then $m \mid \gcd(a, b)$.

Proof. By Bézout's Lemma, there exists integers x and y such that

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

Hence, by $m \mid a$ and $m \mid b$, we know

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

□

Greatest Common Divisor (GCD) - Properties

Property 2

$\gcd(ka, kb) = k \times \gcd(a, b)$

Proof. Suppose $d = \gcd(ka, kb)$ and $e = k \times \gcd(a, b)$. We want to show that $e \leq d$ and $d \leq e$, which together implies $d = e$.

- ($d \leq e$) Notice that $k \mid d = \gcd(ka, kb)$. We write $d = kd'$. By $kd' = d \mid ka$, we know that $d' \mid a$, and similarly $d' \mid b$. Hence, d' is a common divisor of a and b, and $d = kd' \leq k \times \gcd(a, b)$.
- ($e \leq d$) Notice that $\gcd(a, b) \mid a \Rightarrow e = k \times \gcd(a, b) \mid ka$. Analogously, $e \mid kb$, which means e is a common divisor of ka and kb, and so $e \leq d$.

□

Greatest Common Divisor (GCD) - Properties

Property 3

Let d denote the greatest common divisor of a and b. Then

$$\gcd\left(\frac{a}{d}, \frac{b}{d}\right) = 1$$

Proof. By Property 2, $\gcd(a, b) = d \times \gcd\left(\frac{a}{d}, \frac{b}{d}\right)$. But $\gcd(a, b)$ is just d, and so we are done. □

Greatest Common Divisor (GCD) - Properties

Property 4

Let d denote the greatest common divisor of a and b. If k divides d, then

$$\gcd\left(\frac{a}{k}, \frac{b}{k}\right) = \frac{\gcd(a, b)}{k}$$

Proof. Similar to property 3, and leave as exercise for the readers. □

Greatest Common Divisor (GCD) - >2 Numbers

What is $\gcd(a_1, a_2, \ldots, a_N)$?

As the order doesn't matter, we can simply calculate them one by one:

$$\gcd(a_1, a_2, \ldots, a_N) = \gcd(\gcd(\ldots \gcd(\gcd(a_1, a_2), a_3) \cdots a_{N-1}), a_N)$$

Greatest Common Divisor (GCD) - Application

1. Simplifying a fraction $\frac{a}{b}$:

$$\frac{a}{b} = \frac{a \div \gcd(a,b)}{b \div \gcd(a,b)}$$

2. Solving linear Diophantine equation for integers x and y:
   - $ax + by + c$ where $\gcd(a,b) \mid c$
   - can be solved by Extended Euclidean Algorithm

3. Calculating the least common multiple (LCM):
   - $\text{lcm}(a,b)$ is defined as the *least common* multiple of integers a and b.
   - Can be calculated as $\text{lcm}(a,b) = \frac{ab}{\gcd(a,b)}$
   - To be briefly introduced in the next few slides

Least Common Multiple (LCM) - Definition

Consider two integers a and b.

- If both a and b divides m, then m is a common multiple of a and b.
- Symbolic form: $a \mid m \wedge b \mid m$ or simply $a, b \mid m$
- Least common multiple (LCM) of a and b is simply the smallest among the common multiples of a and b.
- We use $\text{lcm}(a, b)$ to represent the least common multiple of a and b.
- You can also use $[a, b]$ when no confusion would arise.

Least Common Multiple (LCM) - Properties

Property 1

if n is a common multiple of a and b, then $\text{lcm}(a, b) \mid n$.

Proof. Suppose $m = \text{lcm}(a, b)$. We write $n = qm + r$ such that $0 \leq r < m$.
If we can somehow show $r = 0$, then $n = qm$ and we are done.

- As $a \mid n$ and $a \mid m$, $a \mid (n - qm) = r$. Similarly, $b \mid r$.
- Therefore, r is a common multiple of a and b.
- Now, as $0 \leq r < m$, if $r \neq 0$, then $r < m$ is a smaller common multiple of a and b, contradiction.
- This forces $r = 0$(!) and so $m \mid n$.

□

Least Common Multiple (LCM) - Properties

Property 2

$\text{lcm}(ka, kb) = k \times \text{lcm}(a, b)$

Property 3

For any positive integers a, b, we have $ab = \gcd(a, b) \times \text{lcm}(a, b)$.

Proof. Leave as exercise for the readers. (hint: how can we show both $LHS \geq RHS$ and $LHS \leq RHS$ using divisibility?) □
Alternatively, we will show how to prove them with more ease using other technique later on.

Least Common Multiple (LCM) - >2 Numbers

What is $\text{lcm}(a_1, a_2, \ldots, a_N)$?

As the order doesn't matter, we can simply calculate them one by one:

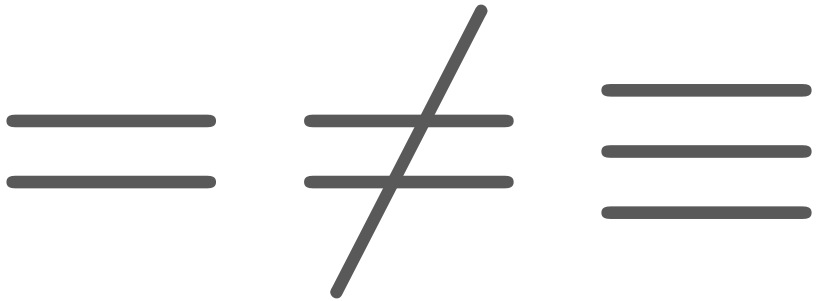
$$\text{lcm}(a_1, a_2, \ldots, a_N) = \text{lcm}(\text{lcm}(\ldots \text{lcm}(\text{lcm}(a_1, a_2), a_3) \cdots a_{N-1}), a_N)$$

Table of Contents

Modular Arithmetic - Introduction

Recall what you have learnt in primary about division:
$17 \div 5 = 3 \ldots 2$ (Dividend $\div$ Divisor $=$ Quotient $\ldots$ Reminder)
So we say the remainder of $17 \div 5$ is 2

Now, we may say $17 \equiv 2 \pmod 5$
"17 and 2 are congruent modulo 5"



Modular Arithmetic - Notation

Generally, if a and b are congruent modulo m where a, b are integers, m is a positive integer, we write $a \equiv b \pmod{m}$

Translate into programming language:

- Pascal: `a mod m = b mod m`
- C++: `a % m == b % m`

The remainder of the division of both a and b by m are the same

Modular Arithmetic - Caution

Take care of the sign of the dividend when writing programs.
If the dividend is negative, the remainder becomes non-positive!

For example, $-5 \equiv 2 \pmod 7$, but `-5 % 7` gives result `-5`

Hence, you may need to write this:

```
((a % b) + b) % b
```

Modular Arithmetic - Meaning

Instead of saying a and b have the same remainder when divided by m, we can say that:

1. $a - b$ is divisible by m.
   Mathematical Notation: $m \mid (a - b)$

2. There exists some integer k such that $a = km + b$.
   Mathematical Notation: $\exists k \in \mathbb{Z}$ s.t. $a = km + b$

Modular Arithmetic - Addition / Subtraction / Multiplication

If $a \equiv b \pmod{m}, c \equiv d \pmod{m}$, then for $x \in \mathbb{Z}$,

- $a \pm x \equiv b \pm x \pmod{m}$
- $a \pm c \equiv b \pm d \pmod{m}$
- $ax \equiv bx \pmod{m}$
- $ax \equiv bx \pmod{mx}$ if $x > 0$
- $ac \equiv bd \pmod{m}$

Proof. Leave as exercise for readers.
(Hint: You can show $m \mid (LHS - RHS)$) □

This basically means when we are doing $+, -, \times$ under modulo, we can replace any number with anything else, as long as they are congruent modulo m.

Modular Division - Introduction

How about division? Can we carry out division as 'normal' arithmetic also?

Sadly, that is not the case. Let's illustrate the issues with some examples:

1. What is the value of $4 \div 2 \pmod{6}$?
2. What is the value of $3 \div 5 \pmod{7}$?

For (1), you probably will say $2 \pmod{6}$ - but $5 \pmod{6}$ is also a possible value.

For (2), even it is a 'fraction', it has the value of $2 \pmod{7}$.

Problem: What actually *is* a modular division anyways?

Modular Division - Definition

Definition of Modular Division

Let a, b, m be integers which $b \neq 0$ and m is positive.
Then, $a \div b$ modulo m is defined to be the integral solutions x for

$$bx \equiv a \pmod{m}$$

For example, $3 \div 5 \equiv 2 \pmod{7}$ as $2 \times 5 = 10 \equiv 3 \pmod{7}$.
Meanwhile, $4 \div 2 \equiv 2, 5 \pmod{6}$ or $2 \pmod{3}$ as $2 \times 2 \equiv 5 \times 2 \equiv 4 \pmod{6}$.

More questions:

1. When does it have solutions?
2. How can we find solutions *quickly* when it exists?

Modular Division - With Extended Euclidean Algorithm

Turns out we had already learnt about this!
Can you recall anything that looks similar?

Linear Diophantine Equation

Given integers a, b, c. Then, solution to $ax + by = c$ exists iff $\gcd(a, b) \mid c$, and all the solutions can be found by Extended Euclidean Algorithm.

If we put $b = m$, then $ax + my = c$ actually just means $ax \equiv c \pmod{m}$!

Modular Division - With Extended Euclidean Algorithm

Linear Diophantine Equation

Given integers a, b, c. Then, solution to $ax + by = c$ exists iff $\gcd(a, b) \mid c$, and all the solutions can be found by Extended Euclidean Algorithm.

If we put $b = m$, then $ax + my = c$ actually just means $ax \equiv c \pmod{m}$!

Solutions would be $x = x_0 + \frac{m}{\gcd(a,m)}k$ for a specific solution x_0, y_0 and all $k \in \mathbb{Z}$.

Rewriting it in modular arithmetic, we know

$$x \equiv x_0 \pmod{\frac{m}{\gcd(a, m)}}$$

Which is exactly what we need!

Modular Inverse - Definition

Modular Inverse is another concept related to division in modular arithmetic:

Definition of Modular Inverse

Let a, m be integers which $a \neq 0$ and m is positive.
Then, a^{-1} modulo m is defined to be the integral solutions x for

$$ax \equiv 1 \pmod{m}$$

Indeed, we can do modular division with modular inverse (when it exists):

$$ax \equiv c \pmod{m} \iff a^{-1}ax \equiv x \equiv a^{-1}c \pmod{m}$$

Modular Inverse - Existence

For some integer a and positive integer m, modular inverse might not exist.

- If $a = 6, m = 4$, for any integer b, ab is always even.
- That is, $ab \equiv 0$ or $2 \pmod 4 \Rightarrow ab \not\equiv 1 \pmod 4$

Modular inverse only exists if $\gcd(a, m) = 1$, that is, a, m are relatively prime.

- Notably, when m is a *prime* while a is not a multiple of m, then modular inverse of a exists.

Note that a *prime* as a positive integer that has exactly two positive divisors, namely 1 and itself.

Modular Inverse - Reality

In fact, most of the modulo (except perhaps in the problem set) you will ever have to deal with in OI problems are primes.

Most common examples:

- $10^9 + 7$
- $998244353 = 7 \times 17 \times 2^{23} + 1$

Majority of time, modulo are there simply to make the answer small enough to fit in 32-bit integer. It is less relevant to the problem itself.

Therefore, most likely when you have to calculate modular inverse, the modulo is a prime.

Modular Inverse - Computation

Two main ways can be used to calculate modular inverse for prime modulo p.

1. Extended Euclidean Algorithm - solve $ax + kp = 1$.
   - Technically also works for non-prime modulo.

2. Fermat's Little Theorem - (when $p \nmid a$) $a^{-1} \equiv a^{p-2} \pmod{p}$. $\hookleftarrow$ Our target
   - Can also be extended for non-prime modulo using the Euler's totient function φ - that to be discussed later.

Modular Inverse - Fermat's Little Theorem

Fermat's Little Theorem

Given integer a and prime p which $p \nmid a$. Then

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

Proof. Notice that

$$a \times 2a \times \cdots \times (p-1)a \equiv 1 \times 2 \times \cdots \times (p-1) \pmod{p}$$

as $\{a, 2a, \ldots, (p-1)a\}$ is simply a permutation of $\{1, 2, \ldots, p-1\}$ modulo p (Why?). By canceling $(p-1)! = 1 \times 2 \times \cdots \times (p-1)$ on both sides, we obtain

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

□

Modular Inverse - Fermat's Little Theorem

Therefore, for integer a such that $p \nmid a$, we have

$$a \times a^{p-2} \equiv a^{p-1} \equiv 1 \pmod{p}$$

which means a^{p-2} is the modular inverse of a!

Here comes the problem: How can we compute exponential quick enough?

Fast Exponential - Problem

Given a, b, m, find $a^b \pmod{m}$.

Code #1:

```
int product = 1;
for (int i = 1; i <= n; i++)
    product = (product * a) % m;
return product;
```

Two problems:

1. Time Complexity: $O(b)$, not fast enough for large b
2. Also, use `long long`!

Fast Exponential - Big Mod

You will first need to know that the law of indices:

$$a^{n+m} = a^n \times a^m$$

Then, there are two methods to proceed:

1. Iterative method
2. Recursive method

Fast Exponential - Big Mod (Iterative)

We can express b as sum of power of 2 as

$$b = 2^{s_1} + 2^{s_2} + \cdots + 2^{s_k}$$

which $s_1 < s_2 < \cdots < s_k$.
Then,

$$a^b = a^{\sum_{i=1}^{k} 2^{s_i}} = \prod_{i=1}^{k} a^{2^{s_i}}$$

Time Complexity: $O(\log n)$.

Fast Exponential - Big Mod (Iterative)

Example: $7^{77} \mod 23 \equiv 22, 77_{10} = 1001101_2$

$$
\begin{aligned}
7^1 &\equiv 7 \pmod{23} \\
7^2 &\equiv 3 \pmod{23} \\
7^4 \equiv 3^2 &\equiv 9 \pmod{23} \\
7^8 \equiv 9^2 &\equiv 12 \pmod{23} \\
7^{16} \equiv 12^2 &\equiv 6 \pmod{23} \\
7^{32} \equiv 6^2 &\equiv 13 \pmod{23} \\
7^{64} \equiv 13^2 &\equiv 8 \pmod{23}
\end{aligned}
$$

$$
\begin{aligned}
7^{77} &\equiv 7^{64+8+4+1} \\
&\equiv 8 \times 12 \times 9 \times 7 \\
&\equiv 22 \pmod{23}
\end{aligned}
$$

Fast Exponential - Big Mod (Recursive)

We can also recursively calculate the answer.

- If $b = 0$, then $a^b \equiv 1 \pmod{m}$.
- If $b = 2k$, then $a^b \equiv a^k \times a^k \pmod{m}$.
- If $b = 2k + 1$, then $a^b \equiv a^k \times a^k \times a \pmod{m}$.

Therefore, we can reduce the problem of finding $a^b \pmod{m}$ to $a^{\lfloor \frac{b}{2} \rfloor} \pmod{m}$.
Time Complexity: $O(\log n)$.

Actually both methods is not limited to modular exponential - and can be used for other stuff as long as the multiplication are associative.

We obtain yet another method to calculate $a^{-1} \equiv a^{p-2} \pmod{p}$, p prime.

Modular Inverse - $(n!)^{-1} \mod p$

Let's say you want to precompute $n! \mod p$ and $(n!)^{-1} \mod p$ for all $0 \leq n \leq N$ for combinatorial stuff. How would you do?
Recall that for $n \geq 1$,

$$n! = n \times (n-1) \times \cdots \times 1 = n \times (n-1)!$$

and so we can do partial product on factorial, then take inverse one-by-one:

```
fac[0] = 1;
for (int i = 1; i <= N; i++)
    fac[i] = fac[i - 1] * i % m;
for (int i = 0; i <= N; i++)
    fac_inv[i] = modinv(fac[i], m);
```

Time Complexity: $O(N \log m)$ (Taking inverse $O(\log m)$)

Modular Inverse - $(n!)^{-1} \mod p$

Taking inverse is much more costly than just modular multiplication - can we transform the algorithm to just using *one* modular inverse?

Turns out yes - by taking inverse of $N!$ and note that for all $1 \leq n \leq N$,

$$\frac{1}{(n-1)!} = \frac{1}{n!} \times n$$

Time Complexity:
$O(N + \log m)$

```
fac[0] = 1;
for (int i = 1; i <= N; i++)
    fac[i] = fac[i - 1] * i % m;
fac_inv[N] = modinv(fac[N], m);
for (int i = N - 1; i >= 0; i--)
    fac_inv[i] = fac_inv[i + 1] * (i + 1) % m;
```

Chinese Remainder Theorem - Introduction

Task C If $\gcd(m, n) = 1$, solve

$$\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n} \end{cases}$$

Chinese Remainder Theorem

There exists a unique solution $0 \leq c < mn$ such that $x \equiv c \pmod{mn}$.

Chinese Remainder Theorem - Idea

Task A If $\gcd(m, n) = 1$, solve

$$\begin{cases} x \equiv 1 \pmod{m} \\ x \equiv 0 \pmod{n} \end{cases}$$

Solution:

- $x \equiv 0 \pmod{n} \Rightarrow x \equiv dn \pmod{mn}$ for some integer d
- The problem is reduced to $x \equiv 1 \pmod{m} \Rightarrow dn \equiv 1 \pmod{m}$
- d is the modular inverse of n modulo m, that is, $d \equiv n^{-1} \pmod{m}$
- How to find d? Extended Euclidean Algorithm!

Chinese Remainder Theorem - Idea

Task B If $\gcd(m, n) = 1$, solve

$$\begin{cases} x \equiv 0 \pmod{m} \\ x \equiv 1 \pmod{n} \end{cases}$$

Solution:

- This is exactly the same as the previous task - and we can do it similarly.
- $x \equiv 0 \pmod{m} \Rightarrow x \equiv cm \pmod{mn}$ for some integer c
- The problem is reduced to $x \equiv 1 \pmod{n} \Rightarrow cm \equiv 1 \pmod{n}$
- c is the modular inverse of m modulo n, that is, $c \equiv m^{-1} \pmod{n}$
- How to find d? Extended Euclidean Algorithm!

Chinese Remainder Theorem - Idea

Task C If $\gcd(m, n) = 1$, solve

$$\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n} \end{cases}$$

How can we use idea from Task A and B?
Solution: From Task A and B, we have

$$\begin{cases} dn \equiv 1 \pmod{m} \\ dn \equiv 0 \pmod{n} \end{cases}, \begin{cases} cm \equiv 0 \pmod{m} \\ cm \equiv 1 \pmod{n} \end{cases}$$

and so

$$x \equiv adn + bcm \pmod{mn}$$

Chinese Remainder Theorem - Generalisation

Task D Solve

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

where $m_1, m_2, \ldots, m_k$ are pairwise coprime.

How can we use idea from Task C?

Algorithmic Approach: Simply ‘merge’ the equations successively!

Chinese Remainder Theorem - Generalisation

Example:

$$\begin{cases} x \equiv 2 \pmod 3 \\ x \equiv 3 \pmod 5 \\ x \equiv 2 \pmod 7 \end{cases}$$

Chinese Remainder Theorem - Generalisation

Solution. From equation (1) and (2),

$$\begin{aligned} x &\equiv 2 \times (5^{-1} \bmod 3) \times 5 + 3 \times (3^{-1} \bmod 5) \times 3 \\ &\equiv 2 \times 2 \times 5 + 3 \times 2 \times 3 \\ &\equiv 38 \equiv 8 \pmod{15} \end{aligned}$$

From above and equation (3),

$$\begin{aligned} x &\equiv 8 \times (7^{-1} \bmod 15) \times 7 + 2 \times (15^{-1} \bmod 7) \times 15 \\ &\equiv 8 \times 13 \times 7 + 2 \times 1 \times 15 \\ &\equiv 758 \equiv 23 \pmod{105} \end{aligned}$$

Table of Contents

Prime Number - Definition

Prime numbers are positive integers which has exactly 2 positive divisors.

- The first few prime numbers are $2, 3, 5, 7, 11, 13, \ldots$

Composite numbers are positive integers with more than 2 positive divisors.

- The first few composite numbers are $4, 6, 8, 9, 10, 12, \ldots$.

Note that 1 is neither a prime nor a composite number.

Prime Number - Facts

Fact 1

If $p = ab$ which a, b are positive integers and $a \leq b$, then $a = 1$ and $b = p$.

Fact 2

By Euclid's Lemma, if $p \mid ab$, at least one of $p \mid a$ and $p \mid b$ holds.

Primality Test - Brute force

How can we check whether a positive integer is prime or not?

Observation 1

For each prime p, it has exactly 2 positive divisors, namely 1 and p.

Code 1:

```
for (int i = 2; i < n; i++)
    if (n % i == 0)
        return false;
return true
```

Time Complexity: $O(n)$:(

Primality Test - Brute Force Till $\sqrt{n}$

Observation 2

If n is composite (not prime), there exists a divisor d of n such that $1 < d \leq \sqrt{n}$.

Proof. Suppose d' is any divisor of n that is neither 1 or n.

- If $d' \leq \sqrt{n}$, then we already find it.
- Otherwise, $d' > \sqrt{n}$. Notice that $\frac{n}{d'}$ is also a factor of n, in which $\frac{n}{d'} \leq \sqrt{n}$.

□

Primality Test - Brute Force Till $\sqrt{n}$

Observation 2

If n is composite, there exists a divisor d of n such that $1 < d \leq \sqrt{n}$.

Code 2:

```
for (int i = 2; i * i <= n; i++) // must be <=, not <
    if (n % i == 0)
        return false;
return true
```

Time Complexity: $O(\sqrt{n})$:|

Primality Test - Miller-Rabin Algorithm (Extra)

Can we do even better than $O(\sqrt{n})$?
Certainly yes! Let's briefly introduce the Miller-Rabin Algorithm.

We will use the following theorem:

Fermat's little Theorem

Given integer a and prime p which $p \nmid a$. Then

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

We shall note that this is a consequence of Euler's theorem - and will leave it here for now.

Primality Test - Miller-Rabin Algorithm (Extra)

Fermat's little Theorem

Given integer a and prime p which $p \nmid a$. Then

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

1. Write $p - 1 = 2^s \times d$, which d is odd.
2. We can factorise $a^{p-1} - 1$ as

$$a^{p-1} - 1 = a^{2^s d} - 1 = (a^d - 1)(a^d + 1)(a^{2d} + 1) \cdots (a^{2^{s-1} d} + 1)$$

3. If p is prime, then at least one term is divisible by p.
4. Why? Euclid's Lemma!

Primality Test - Miller-Rabin Algorithm (Extra)

Therefore, if a natural number n fails this test, then it must be composite. Note that this doesn't mean if n passes this test, it must be a prime - it just *probably* a prime.

Question: Is there any *liar* composite n that can pass the test for all $1 < a < n$?
Answer: Luckily not for Miller-Rabin!

In fact, for $n \leq 10^{18}$, it is sufficient to check the first 12 primes ($a = 2, 3, \ldots, 37$).

Primality Test - Miller-Rabin Algorithm (Extra)

```
bool check_comp(int a, int n){
    int d = n - 1, s = 0;
    while (d % 2 == 0)
        d /= 2, s++;
    int r = powmod(a, d, n);
    if (r == 1)
        return false;
    for (int i = 0; i < s; i++) {
        if (r == n - 1)
            return false;
        r = r * r % n;
    }
}
```

```
bool is_prime(int n){
    if (n == 1) return false;
    for (a in {2, 3, ..., 37}) {
        if (n == a) return true;
        if (check_comp(a, n))
            return false;
    }
    return true;
}
```

Time Complexity: $O(k \log^3(n))$ (k the number of rounds) :)

Prime Sieve - Brute Force Till $\sqrt{n}$

Now - how about finding ALL the prime numbers between 1 and 10^6?

Alternatively, how can we *sieve away* all the composite numbers?

Code 2*:

```
for (int j = 1; j <= 1000000; j++)
    if (is_prime(j)) output j; // Using Code #2
```

Time Complexity: $O(n\sqrt{n})$:(

Prime Sieve - Sieve of Eratosthenes

Observation 3

For any composite number, it is divisible by some prime number.
For any prime number, the only prime divisor is itself.

Proof. For composite numbers n,

- Consider its smallest divisor $p > 1$. We want to show that p must be prime.
- Suppose otherwise, i.e. p is composite.
- Then we can find a divisor of p that is neither 1 and p, say a.
- But then $a \mid n$ also, contradicting the fact that p is the smallest divisor (greater than 1) of n.

The prime number part is straight from definition.

□

Prime Sieve - Sieve of Eratosthenes

Hence, we may store a list of prime numbers we have found.

- For each integer, check if it is divided by any of the prime numbers found.
- If not, then it is a prime, which we can add it to the prime list.
- We may use store a Boolean array which indicates whether each integer is known to be composite.

Prime Sieve - Sieve of Eratosthenes

Current Step: List out all the numbers													
1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98
99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate 1													
	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98
99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 2 except 2													
	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98
99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 2 except 2												
	2	3		5		7		9		11		13	
15		17		19		21		23		25		27	
29		31		33		35		37		39		41	
43		45		47		49		51		53		55	
57		59		61		63		65		67		69	
71		73		75		77		79		81		83	
85		87		89		91		93		95		97	
99		101		103		105		107		109		111	
113		115		117		119		121		123		125	
127		129		131		133		135		137		139	
141		143		145		147		149		151		153	

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 3 except 3												
	2	3		5		7		9		11		13
15		17		19		21		23		25		27
29		31		33		35		37		39		41
43		45		47		49		51		53		55
57		59		61		63		65		67		69
71		73		75		77		79		81		83
85		87		89		91		93		95		97
99		101		103		105		107		109		111
113		115		117		119		121		123		125
127		129		131		133		135		137		139
141		143		145		147		149		151		153

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 3 except 3													
	2	3		5		7				11		13	
		17		19				23		25			
29		31				35		37				41	
43				47		49				53		55	
		59		61				65		67			
71		73				77		79				83	
85				89		91				95		97	
		101		103				107		109			
113		115				119		121				125	
127				131		133				137		139	
		143		145				149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 5 except 5													
	2	3		5		7				11		13	
		17		19				23		25			
29		31				35		37				41	
43				47		49				53		55	
		59		61				65		67			
71		73				77		79				83	
85				89		91				95		97	
		101		103				107		109			
113		115				119		121				125	
127				131		133				137		139	
		143		145				149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 5 except 5													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47		49				53			
		59		61						67			
71		73				77		79				83	
				89		91						97	
		101		103				107		109			
113						119		121					
127				131		133				137		139	
		143						149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 7 except 7													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47		49				53			
		59		61						67			
71		73				77		79				83	
				89		91						97	
		101		103				107		109			
113						119		121					
127				131		133				137		139	
		143						149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 7 except 7													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47						53			
		59		61						67			
71		73						79				83	
				89								97	
		101		103				107		109			
113								121					
127				131						137		139	
		143						149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 11 except 11													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47						53			
		59		61						67			
71		73						79				83	
				89								97	
		101		103				107		109			
113								121					
127				131						137		139	
		143						149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Eliminate multiples of 11 except 11													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47						53			
		59		61						67			
71		73						79				83	
				89								97	
		101		103				107		109			
113													
127				131						137		139	
								149		151			

Prime Sieve - Sieve of Eratosthenes

Current Step: Done since $13 \times 13 > 154$													
	2	3		5		7				11		13	
		17		19				23					
29		31						37				41	
43				47						53			
		59		61						67			
71		73						79				83	
				89								97	
		101		103				107		109			
113													
127				131						137		139	
								149		151			

Prime Sieve - Sieve of Eratosthenes

```
// initialise boolean array comp[] with false;
for (int i = 2; i * i <= n; i++) {
    if (comp[i] == false) {
        for (int j = i * i; j <= n; j += i)
            comp[j] = true;
    }
}
```

Time Complexity: $O(n \log \log n)$

Question: What modification can be made to maintain the least prime factors of each number also?

Prime Sieve - Linear Sieve (Extra)

Can we do it *even faster*? Of course *yes*!

Observation 4

For any positive integer n, we can represent it uniquely as $n = pa$ which p is the smallest prime factor of n.

Subtle difference between the Sieve of Eratosthenes and the Linear Sieve:

- Sieve of Eratosthenes: we loop p first, then loop through all $n = pa$.
- Linear Sieve: instead, we loop a first, then loop through all $n = pa$.

In the Sieve of Eratosthenes, some numbers n are counted several times as we cannot 'skip' those a that we counted already.

Prime Sieve - Linear Sieve (Extra)

We try to ensure that all positive integers n is counted once only.

Question: When is a prime p being the smallest prime factor of $n = pa$?
Answer: When p does not exceed the smallest prime factor of a.

Therefore, we can do the following:

- We use a vector to store the prime numbers found already.
- We also need to store the least prime factor of each number in an array.
- For each a, loop all the primes p does not exceed the smallest prime factor of a *only*.

Note: Despite its faster time complexity, practically the performance of this is similar to the Sieve of Eratosthenes for *small* n ($n \leq 10^7$).

Prime Sieve - Linear Sieve (Extra)

```
// initialise int array lp[] with 0;
int lp[]; vector<int> pr;
for (int i = 2; i <= n; i++) {
    if (lp[i] == 0) {
        lp[i] = i; pr.push_back(i);
    }
    for (int p : pr) {
        if (p > lp[i] || i * p > n) break;
        lp[i * p] = p;
    }
}
```

Time Complexity: $O(n)$

Table of Contents

Prime Factorisation - Definition

Fundamental Theorem of Arithmetic

For any positive integer n, we can represent n uniquely as

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

which $p_1 < p_2 < \cdots < p_k$ are primes while $\alpha_1, \alpha_2, \ldots, \alpha_k$ are positive integers.

Proof. In fact, to prove this statement, there are two aspects we need to show:

1. Existence. There exist such a way to write n as products of primes.
2. Uniqueness. There does not exists two different ways to represent n.

Prime Factorisation - Definition

1. Existence. There exist such a way to write n as products of primes.

We try to demonstrate this *recursively*.

- Notice that the smallest divisor $p > 1$ of n must be a prime.
- Then, we can write $n = pa$ which $a < n$ is a positive integer.
- Repeat this process over and over, and then we obtain the *prime factorisation* of n.

Prime Factorisation - Definition

2. Uniqueness. There does not exists two different ways to represent n.

- Suppose we can represent n in two distint ways:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = q_1^{\beta_1} q_2^{\beta_2} \cdots q_l^{\beta_l}$$

  which we pick the minimal such n.
- Consider p_1. As $p_1 \mid n$, by Euclid's lemma, there exists a q_i such that $p_1 \mid q_i$.
- Furtherfore, as p_1 and q_i are both primes, $p_1 = q_i$.
- That means we can divide the equation by p_1 to obtain two distinct prime factorisation of $\frac{n}{p_1}$. This contradicts the minimality of n.

□

Prime Factorisation - Using Sieve

We can modify the Sieve of Eratosthenes to maintain smallest prime factor:

- Replace boolean array with integer array.
- When we mark an integer as composite, store the current prime factor.

Then, for each integer, we can get the smallest prime divisor instantly, and we can factorise the number recursively.

Prime Factorisation - Pollard-Rho (Extra)

What if the number to be factorised is as large as 10^{18}? Can we still factorise it?

We surely can - let's first introduce a seemingly unrelated theorem:

Birthday Paradox

Given a sequence $x_1, x_2, \ldots, x_n, \ldots$ which each x_i takes a uniformly random integer between 0 and $p-1$ (inclusive) with p is a positive integer. Then, the expected value of k such that the first collision $x_i = x_k$ which $i \leq k$ occurs is $O(\sqrt{p})$.

Prime Factorisation - Pollard-Rho (Extra)

Proof.

- Consider fixed $1 \leq i < j \leq k$. The probability that $x_i = x_j$ is $\frac{1}{p}$.
- There are in total $\frac{k(k-1)}{2}$ pairs of (i, j) which all of them are independent.
- Therefore, for all these pairs not to collide, the probability is

$$\left(1 - \frac{1}{p}\right)^{\frac{k(k-1)}{2}}$$

- When picking $k = \lambda\sqrt{p}$, this probability becomes

$$\approx \left(1 - \frac{1}{p}\right)^{\frac{\lambda^2 p}{2}} = \left(1 - \frac{1}{p}\right)^{p(\frac{\lambda^2}{2})} \approx e^{-\frac{\lambda^2}{2}}$$

  which $e = 2.71828...$.
- For example, when $\lambda = 5\sqrt{p}$, probability of failing is about 3.73×10^{-6}. □

Prime Factorisation - Pollard-Rho (Extra)

Suppose the number to be factorised be $n > 1$.

- Use Miller-Rabin to check whether n is a prime first - if so we are done.
- Otherwise, n must be a composite number. It has at least one *non-trivial* divisor $1 < d < n$.

Now, suppose we have a sequence of *pseudo*-random numbers modulo n:

$$a \bmod n, f(a) \bmod n, f(f(a)) \bmod n, \ldots$$

There are two requirements our function f has to satisfy:

1. For every divisor $d \mid n$, if $x \equiv y \pmod{d}$, then $f(x) \equiv f(y) \pmod{d}$.
2. It generates a sequence of *pseudo*-random numbers.

Prime Factorisation - Pollard-Rho (Extra)

There are two requirements our function f has to satisfy:

1. For every divisor $d \mid n$, if $x \equiv y \pmod{d}$, then $f(x) \equiv f(y) \pmod{d}$.
2. It generates a sequence of *pseudo*-random numbers.

Why do we want that?

1. This means for *each* $d \mid n$, the sequence modulo d will eventually enter a cycle. For instance, the sequence modulo d can be like

$$x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_l, y_1, y_2, \ldots$$

2. This means we can expect the Birthday Paradox holds - that is, under modulo d, the expected time to enter a cycle is $O(\sqrt{d})$.

Prime Factorisation - Pollard-Rho (Extra)

Intuition: If we catch two values, say a and b, that are at the same position of a cycle for some $1 < d < n$ but not the same position for n, then we will have

$$a \equiv b \pmod{d}$$

but

$$a \not\equiv b \pmod{n}$$

Then $\gcd(a - b, n)$ will give us a non-trivial divisor of n!

Now let's say p is the smallest prime factor of n. We know that $p \leq \sqrt{n}$. (Why?) Hence, by the Birthday Paradox, the entry to cycle would likely occur after $O(\sqrt{p}) \leq O(\sqrt[4]{n})$ elements. For each element, we will check once the GCD which costs $O(\log n)$, so the final time complexity is $O(\sqrt[4]{n} \log n)$.

Prime Factorisation - Pollard-Rho (Extra)

- In practice, we may use function like $f(x) = (x^2 + c)$ mod n for some arbitrary integer c (which satisfies both conditions).
- We can use the Floyd's cycle-finding algorithm to find the cycle.
  - Each step: (Tortoise) a to $f(a)$, (Hare) b to $f(f(b))$ - See code for details.
- There is a small chance that it fails - that is, $a \equiv b \pmod{n}$ before any other thing happens. In this case, simply repeat again with other c.
- Note that this is not likely due to the Birthday Paradox (Why?)

In general, prime factorisation is very hard. Hence, it is widely use in cryptography, e.g., the RSA encryption. You will see in the problem set one special occasion which allows you to factorise a *large* number.

Prime Factorisation - Pollard-Rho (Extra)

```
int c = 20240309;
int f(int x, int n){
    return (x * x) % n + c;
}

void find_factor(int n){
    if (n == 4) { // special case
        output(2); output(2); return;
    }
    if (is_prime(n)) { // Miller-Rabin
        output(n); return;
    }
    while (true) {
        int a = rand() % n;
        int b = a;
        while (gcd(a, b) == 1) {
            a = f(a, n); b = f(f(b, n), n);
        }
        if (gcd(a, b) == n) { // bad case
            --c; continue; // change c
        }
        int d = gcd(a, b);
        find_factor(d);
        find_factor(n / d);
    }
}
```

Time Complexity: $O(\sqrt[4]{n}\log n)$

p-adic valuation - Definition

Actually, given the prime factorisation of an positive integer n,

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

There are much more we can do!
Let's introduce the p-adic valuation of an integer:

Definition of p-adic valuation

Given an integer a and a prime p. Then, the p-adic valuation of a is defined to be

$$\nu_p(a) = \max\{k \mid p^k \text{ divides } a\}$$

Notice that it is a greek alphabet nu in lower case.

p-adic valuation - Properties

For example, $\nu_2(24) = 3$ as $24 = 2^3 \times 3^1$. Similarly, $\nu_3(24) = 1$. Meanwhile, for all the other primes $p \geq 5$, $\nu_p(24) = 0$.

Property 1

For any integers a and b, $\nu_p(ab) = \nu_p(a) + \nu_p(b)$.

Property 2

For any integers a, b, x and y, $\nu_p(ax \pm by) \geq \min\{\nu_p(a), \nu_p(b)\}$.

Proof. Straight from definition - or simply trust your intuition. □

p-adic valuation - Properties

We shall introduce two claims regarding the p-adic valuation with gcd and lcm:

Relationship between ν_p and gcd

For all prime p, $\nu_p(\gcd(a, b)) = \min(\nu_p(a), \nu_p(b))$.

Relationship between ν_p and lcm

For all prime p, $\nu_p(\text{lcm}(a, b)) = \max(\nu_p(a), \nu_p(b))$.

Proof. Leave as exercise for readers. □

p-adic valuation - Properties

Which means for all prime p,

$$\begin{aligned}\nu_p(\gcd(a,b)\operatorname{lcm}(a,b)) &= \nu_p(\gcd(a,b)) + \nu_p(\operatorname{lcm}(a,b)) \\ &= \min(\nu_p(a),\nu_p(b)) + \max(\nu_p(a),\nu_p(b)) \\ &= \nu_p(a) + \nu_p(b) \\ &= \nu_p(ab)\end{aligned}$$

Which means

$$\gcd(a,b)\operatorname{lcm}(a,b) = ab$$

This is the proof we promised back then - and is much simpler than with divisibility!

Factorial Factors - Introduction

Now, how about finding $\nu_p(n!)$, that is, the highest power k such that $p^k \mid n!$?

Obviously, it is impractical to compute the exact value of $n!$: $20!$ already exceeds `long long` range.

However, recall that for any integers a and b, $\nu_p(ab) = \nu_p(a) + \nu_p(b)$.

Factorial Factors - Introduction

A trivial implementation gives
Code #1:

```
int k = 0;
for (int i = 1; i <= n; i++) {
    int num = i;
    while (num % p == 0) {
        k++;
        num /= p;
    }
}
return k;
```

Time Complexity: $O(n \log n)$

Factorial Factors - More Observations

Example: $n = 20, k = 2$																			
Divisibility	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2		✓		✓		✓		✓		✓		✓		✓		✓		✓		✓
4				✓				✓				✓				✓				✓
8								✓								✓				
16																✓				
Highest Power	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0	1	0	2
Sum of Highest Powers = 18																				

Is there another way to come up the result 18?
(Or equivalently, any other way to count the number of ticks?)

Instead of counting the number of ticks per column,
try counting it per row, which is also meaningful!

Factorial Factors - More Observation

Row r repersents: the number of integers between 1 and n in which they are divisible by p^r.

How many? It's easy! :) Answer: $\left\lfloor \frac{n}{p^r} \right\rfloor$

In fact, this is called Legendre's Formula:

Legendre's Formula

$$\nu_p(n!) = \sum_{r=1}^{\infty} \left\lfloor \frac{n}{p^r} \right\rfloor$$

Factorial Factors - Implementation

Code #2:

```
int k = 0;
while (n) {
    n /= p;
    k += n;
}
return k;
```

Time Complexity: $O(\log n)$

Factorial Factors - Extensions

How about the highest power of p^a such that it divides $n!$ for prime p?
That is, output the highest power k such that $(p^a)^k \mid n!$

Answer: Simply $\left\lfloor \frac{\nu_p(n!)}{a} \right\rfloor$.

Now, how about the highest power of m such that it divides $n!$?
That is, output the highest power k such that $m^k \mid n!$

Hint: Prime factorisation of m.

Application: For instance, find the trailing zeros of $n! \Rightarrow m = 10$.

Table of Contents

Number-Theoric Functions - Definition

Let's first define what a number-theoric function is.

Definition of Number-Theoric Functions

A number-theoric function is any function that takes positive integers as input, and return a integer (or formally, a complex number).

In the following slides, we will mainly talk about

- Divisor function σ and
- Euler's totient function φ.

Number-Theoric Functions - Multiplicative Function

Let's first introduce the speciality of *multiplicative function*:

Multiplicative Function

A number-theoric function f is called *multiplicative*, if

$$f(a)f(b) = f(ab)$$

holds for any positive integers a, b such that $\gcd(a, b) = 1$.

Note that for a multiplicative function f, $f(1) = 1$.

Completely Multiplicative Function: no restriction of $\gcd(a, b) = 1$.

Number-Theoric Functions - Multiplicative Function

Say we consider the prime factorsiation of n:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

Then, for multiplicative function f, we know that

$$f(n) = f(p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}) = f(p_1^{\alpha_1}) f(p_2^{\alpha_2}) \cdots f(p_k^{\alpha_k}) = \prod_{i=1}^{k} f(p_i^{\alpha_i})$$

must hold!

This is quite an important relationship for now as both the Divisor function σ and Euler's totient function φ are in fact multiplicative.

Divisor Function σ_k - Definition

Now, let's investigate each function in some details.
For every positive integer n, the divisor function $\sigma_k(n)$ is defined as

$$\sigma_k(n) = \sum_{d|n} d^k$$

When $k = 0$, we have $\tau(n) = \sigma_0(n) = \sum_{d|n} 1 =$ number of divisors of n.
When $k = 1$, we have $\sigma(n) = \sigma_1(n) = \sum_{d|n} d =$ sum of divisors of n.

Example: $\sigma_0(24) = 8, \sigma_1(24) = 60\ (d = 1, 2, 3, 4, 6, 8, 12, 24)$

Divisor Function σ_k - Properties

Property 1

For prime p,

$$\sigma_0(p) = 2$$

$$\sigma_0(p^\alpha) = \alpha + 1$$

$$\begin{aligned}\sigma_k(p^\alpha) &= p + p^{(\alpha-1)k} + \cdots + p^k + 1 \\ &= \sum_{i=0}^{\alpha} (p^k)^i = \frac{p^{(\alpha+1)k} - 1}{p^k - 1}\end{aligned}$$

Divisor Function σ_k - Properties

Property 2

σ_k is multiplicative, i.e.

$$\sigma_k(mn) = \sigma_k(m)\sigma_k(n)$$

whenever $\gcd(n, m) = 1$.

Proof. Note that

$$\begin{aligned}\sigma_k(mn) &= \sum_{d|mn} d^k = \sum_{d|mn} \gcd(d, m)^k \gcd(d, n)^k \\ &= \sum_{d_1|m} \sum_{d_2|n} d_1^k d_2^k = \sum_{d_1|m} d_1^k \sum_{d_2|n} d_2^k \\ &= \sigma_k(m)\sigma_k(n)\end{aligned}$$

□

Divisor Function σ_k - Formula

With σ_k being multiplicative, we can obtain a closed-form formula for σ_k:

Formula for Divisor Function

$$\sigma_0(n) = \prod_{i=1}^{r} \sigma_0(p_i^{\alpha_i}) = \prod_{i=1}^{r} (\alpha_i + 1)$$

$$\sigma_k(n) = \prod_{i=1}^{r} \sigma_k(p_i^{\alpha_i}) = \prod_{i=1}^{r} \frac{p_i^{(\alpha_i+1)k} - 1}{p^k - 1}$$

Euler's Totient Function φ - Definition

For every positive integer n, $\varphi(n)$ is defined to be the number of integers k from 1 to n coprime with n. Symbolically,

$$\varphi(n) = |\{k \mid 1 \leq k \leq n \wedge \gcd(k, n) = 1\}|$$

e.g., $\varphi(24) = 8 : k = 1, 5, 7, 11, 13, 17, 19, 23$

Euler's Totient function is also called Euler's φ function.

Euler's Totient Function φ - Properties

Property 1

For prime p,

$$\varphi(p) = p - 1$$

$$\begin{aligned}\varphi(p^\alpha) &= p^\alpha - p^{\alpha-1} \\ &= p^\alpha\left(1 - \frac{1}{p}\right)\end{aligned}$$

Proof. Note that the only way where $\gcd(a, p^\alpha) > 1$ is when $p \mid a$. As there are in total $p^{\alpha-1}$ such number divisible by p and NOT coprime with p^α, the rest are all coprime with p^α and there are in total $p^\alpha - p^{\alpha-1}$ such many. □

Euler's Totient Function φ - Properties

Property 2

φ is multiplicative, i.e.

$$\varphi(mn) = \varphi(m)\varphi(n)$$

whenever $\gcd(n, m) = 1$.

Proof outline: Chinese Remainder Theorem

1. 1-to-1 correspondence between $x \equiv c \pmod{mn}$ and $\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n} \end{cases}$
2. Only consider the co-prime integers:

$$\gcd(x, mn) = 1 \iff \gcd(x, m) = 1 \wedge \gcd(x, n) = 1$$

Euler's Totient Function φ - Properties

Independent Residues: Example

mod 3	mod 8	mod 24	mod 3	mod 8	mod 24
1	1	1	2	1	17
1	3	19	2	3	11
1	5	13	2	5	5
1	7	7	2	7	23

Another Intuition: Notice the pattern in Sieve of Eratosthenes. Crossing out multiples of p do not affect the "frequency" of other primes removing terms.

Euler's Totient Function φ - Formula

Formula for Euler's Totient Function

$$\varphi(n) = \prod_{i=1}^{r} \varphi(p_i^{\alpha_i}) = \prod_{i=1}^{r} p_i^{\alpha_i}\left(1 - \frac{1}{p}\right) = n\prod_{i=1}^{r}\left(1 - \frac{1}{p}\right)$$

Or equivalently,

$$\varphi(n) = n\prod_{p|n}\left(1 - \frac{1}{p}\right)$$

Example: As $24 = 2^3 \times 3, \varphi(24) = 24\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right) = 8.$

Euler's Theorem - Introduction

Euler's (Totient) Theorem

If $\gcd(a, n) = 1$, then

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Note that Fermat's Little Theorem we had mentioned before is actually a special case for this, with $\varphi(p) = p - 1$:

Fermat's Little Theorem

Given integer a and prime p which $p \nmid a$. Then

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

Euler's Theorem - Application

$\varphi(n)$ is NOT necessarily the smallest positive integer satisfying $a^k \equiv 1 \pmod{n}$.

As mentioned before, now for non-prime modulo n, if $\gcd(a, n) = 1$ then we have

$$a \times a^{\varphi(n)-1} \equiv a^{\varphi(n)} \equiv 1 \pmod{n}$$

which means

$$a^{-1} \equiv a^{\varphi(n)-1} \pmod{n}$$

In which we can compute with fast exponentiation.

The proof of Euler's Theorem is similar to that of Fermat's Little Theorem - and this margin is not wide enough, so might as well leave as exercise for readers.

Reference

- Wikipedia
- Past Mathematics in OI (I), (II) sides - 2015 - 2023
- CP Algorithms - Primality Tests, Integer factorization

Practice Problems

HKOJ 20374 Big Mod
HKOJ I0501 Divisor Game (Interactive)
HKOJ M0723 Frog
HKOJ J041 Traffic Lights
HKOJ 01029 N Collinear Planets
HKOJ M1631 A Strange Elevator
HKOJ M1821 Contest Score
HKOJ M1822 Power Tower

More:
Codeforces - Number Theory Tag
Codeforces - CRT Tag

Questions?