Begin: 2025-02-17 15:00 UTC+2

☆ Week #4 - (Number Theory)

End: 2025-02-24 15:00 UTC+2

Ended

Overview | Problem | Status | Rank (7:00:00:00) | Discuss

Setting | Clone

	Stat	#	Origin	Title
	6 / 9	A	Gym 499179ZA	GCD Queries
	7 / 13	B	SPOJ FACT0	Integer Factorization (15 digits)
	7 / 31	C	Gym 499179ZD	PerfectPerfect
	5 / 8	D	Gym 499179ZF	Simple Subtraction
	4 / 21	E	CSES 1713	Counting Divisors
	4 / 9	F	SPOJ DIVSUM	Divisor Summation
	1 / 13	G	SPOJ CEQU	Crucial Equation
	1 / 5	H	CodeForces 1765M	Minimum LCM
	2 / 3	I	CodeForces 822A	I'm bored with life
	2 / 3	J	CodeForces 230B	T-primes
	1 / 3	K	Gym 499179ZC	Golden Sum
	1 / 1	L	CodeForces 1764B	Doremy's Perfect Math Class

	Stat	#	Origin	Title
	4 / 10	M	CodeForces 762A	k-th divisor
	3 / 7	N	Gym 499179ZE	Count Primes
	2 / 4	O	CSES 1081	Common Divisors
	2 / 11	P	AtCoder abc276_d	Divide by 2 or 3
	1 / 1	Q	CodeForces 1742D	Coprime
	0 / 1	R	AtCoder abc215_d	Coprime 2
		S	CodeForces 1771C	Hossam and Trainees
		T	Gym 499179ZG	Power Factor

Public. Prepared by Mohanned665 @ ICPC Helwan Level 2- Winter 2025, 2025-02-15 22:59:13

Topics covered:

1. Modular Arithmetic

- $(x + y)\%m = (x\%m + y\%m)\%m$
- $(x - y)\%m = ((x\%m - y\%m)\%m + m)\%m$
- $(x \cdot y)\%m = (x\%m \cdot y\%m)\%m$
- $x^n\%m = (x\%m)^n\%m$
- $\frac{x}{d}\%m \neq \frac{x\%m}{d\%m}\%m$ — will be discussed in level 3

2. Factors and Divisors

- A number a is called a factor or a divisor of a number b if a divides b. If a is a factor of b, we write $a \mid b$, and otherwise we write $a \nmid b$. For example, the factors of 24 are $1, 2, 3, 4, 6, 8, 12 \text{ and } 24$.
- Any number can be represented as a product of some prime number, these are known as **prime factors** and are unique for every number (no two number has the same prime factors).
- Similar to prime checking, prime factors of n can calculated by iterating over all numbers less than or equal to n.

$$n = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \dots$$

Where p_i are prime factors and α_i is the number of occurrences of the i-th prime factor.

```cpp
vector<int> factors(int n) {
    vector<int> f;
    for (int x = 2; x*x <= n; x++) {
        while (n%x == 0) {
            f.push_back(x);
            n /= x;
        }
    }
    if (n > 1) f.push_back(n);
    return f;
}
```

- Number of divisors of a number can be calculated from prime factors using the given formula: $\prod (\alpha_i + 1)$, where α_i is the number of occurences of the i-th prime factor.
  The sum of divisors can be calculated using the following formula: $\prod \frac{{p_i}^{\alpha_i+1}-1}{p_i-1}$

3. Primes

- A prime number is a number that is only divisible by 1 and itself
- All prime numbers are odd except for 2, 1 is not prime
- We can check if a number is prime by just looping and checking if it's divisible by any other number, this takes $O(n)$.
- If a number n is composite (not prime), it will always have a divisor smaller than or equal to $\sqrt{n}$, This can optimize prime checking to $O(\sqrt{n})$.
- Here is the implementation for prime checking in $O(\sqrt{n})$

```cpp
bool isPrime(int n){
    if (n < 2) return false;
    for (int i = 2; i*i <= n; ++i){
        if (n % i == 0) return false;
    }
    return true;
}
```

Prime Factorizatoin:

Every positive integer can be expressed as a product of its prime factors. Understanding operations on numbers in terms of their prime factorizations can simplify calculations and provide deeper insights into number properties.

```cpp
int number = 123248;
map<int, int> primes; // prime, frequency
for (ll i = 2; i * i <= number; i++){
        while (number % i == 0){
                primes[i]++;
                number /= i;
        }
}
if (number != 1)
        primes[number]++;
for (auto [x, y] : primes)
        cout << "Prime: " << x << "
,Occurrences: " << y << '\n';
```

Operations on Prime Factors

1- Multiplication: Addition of Exponents:

If two numbers are given in their prime factored form:
$a = \prod x^{f_a(x)}, \quad b = \prod x^{f_b(x)}$
Then their product is given by:
$a \times b = \prod x^{f_a(x)+f_b(x)}$
Multiplication corresponds to the addition of exponents of common prime factors.

2- Division: Subtraction of Exponents:

For division:
$\frac{a}{b} = \prod x^{f_a(x) - f_b(x)}$
Thus, division corresponds to the subtraction of exponents.

- If The subtraction of the frequency of primes is negative, then the division can't be done.

3- Exponentiation: Multiplication of Exponents:

If a number is raised to an exponent:
$(\prod x^{f_a(x)})^k = \prod x^{k f_a(x)}$
Exponentiation corresponds to multiplying the exponents.

4- Roots: Division of Exponents:

For an integer root:
$\sqrt[k]{\prod x^{f_a(x)}} = \prod x^{f_a(x)/k}$
That is, taking a root corresponds to dividing the exponents.

5- Checking Divisibility Using Prime Factorization:

A number a is divisible by another number b if every prime factor of b appears in a with at least the same exponent. Mathematically, if:
$a = \prod x^{f_a(x)}, \quad b = \prod x^{f_b(x)}$
then a is divisible by b if and only if:
$f_a(x) \geq f_b(x) \quad \forall x.$

```cpp
#include <iostream>
#include <map>
using namespace std;

map<int, int> primeFactorize(int n) {
    map<int, int> factors;
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            factors[i]++;
            n /= i;
        }
    }
    if (n > 1) factors[n]++;
    return factors;
}

map<int, int> multiplyFactors(const map<int,
int>& a, const map<int, int>& b) {
    map<int, int> result = a;
    for (auto [p, exp] : b) {
        result[p] += exp;
    }
    return result;
}

map<int, int> divideFactors(const map<int,
int>& a, const map<int, int>& b) {
    map<int, int> result = a;
    for (auto [p, exp] : b) {
        result[p] -= exp;
        if (result[p] == 0) result.erase(p);
    }
    return result;
}

int main() {
    int a, b;
    cin >> a >> b;

    auto fa = primeFactorize(a);
```

```
    auto fb = primeFactorize(b);

    auto product = multiplyFactors(fa, fb);
    auto quotient = divideFactors(fa, fb);

    cout << "Prime factors of a * b:" << endl;
    for (auto [p, exp] : product) {
        cout << p << "^" << exp << " ";
    }
    cout << endl;

    cout << "Prime factors of a / b:" << endl;
    for (auto [p, exp] : quotient) {
        cout << p << "^" << exp << " ";
    }
    cout << endl;
}
```

Sieve of Eratosthenes

- Sieve of Eratosthenes can be used to check if a number is prime in $O(1)$, but it requires $O(n \cdot log(n))$ precomputation, it usually works for numbers up to 10^7.
- The precomputation is done by iterating through all multiples of all prime numbers that are less than or equal to n and marking them as non-prime in an array.

```
const int LIMIT = 1e7 + 9;
vector<bool> isPrime(LIMIT + 1, true);
void initializeSieve(){
    isPrime[0] = isPrime[1] = false;
    for (int x = 2; x <= LIMIT; x++) {
        if (!isPrime[x]) continue;
        for (int u = 2 * x; u <= LIMIT; u += x)
{
            isPrime[u] = false;
        }
    }
}
```

- Sieve can be used with square root loop to optimize prime checking for larger numbers by generating all the primes less than or equal $\sqrt{(n)}$ and iterating over them.

```cpp
const int LIMIT = 1e7 + 9;
vector<bool> isPrime(LIMIT + 1, true);
vector<int> primes;
void initializeSieve(){
    isPrime[0] = isPrime[1] = false;
    for (int x = 2; x <= LIMIT; x++) {
        if (!isPrime[x]) continue;
        for (int u = 2 * x; u <= LIMIT; u += x)
{
            isPrime[u] = false;
        }
        primes.push_back(x);
    }
}

bool checkPrime(int n){
    if (n <= 1) return false;
    for (auto prime: primes){
        if (prime * prime > n) return true;
        if (n % prime == 0) return false;
    }
    return true;
}
```

Conjectures

Goldbach's conjecture

- Goldbach's conjecture is one of the oldest and best-known unsolved problems in number theory and all of mathematics. It states that every even natural number greater than 2 is the sum of two prime numbers.
- The conjecture has been shown to hold for all integers less than 4×10^{18} but remains unproven despite considerable effort.

Twin prime conjecture

- There is an infinite number of pairs of the form $\{p, p+2\}$, where both p and $p+2$ are primes

Legendre's conjecture

- There is always a prime between numbers n^2 and $(n+1)^2$, where n is any positive integer.

Sieve Prime Factorization (SPF)

- Sieve can be modified and used for prime factorization.
- Instead of storing whether a number is prime or not, we store a prime factor for the number

```
const int LIMIT = 1e7 + 9;
vector<int> sieve(LIMIT + 1);
void initializeSieve(){
    for (int x = 2; x <= LIMIT; x++) {
        if (sieve[x]) continue;
        for (int u = 2*x; u <= LIMIT; u += x) {
            sieve[u] = x;
        }
    }
}
```

- Here `sieve[x]` is 0 if x is prime, else it's the largest prime factor of x, we can use this to optimize factorization like this:

```
vector<int> factors(int n){
    vector<int> f;
    while (sieve[n]){
        f.push_back(sieve[n]);
        n /= sieve[n];
    }
    if (n > 1) f.push_back(n);
    return f;
}
```

- This works in $O(log_2(n))$, since in the worst case the number will be a power of two.

4. GCD and LCM:

- $gcd(a, b)$ is the largest number that divides both a and b.
- $gcd(a, b, c) = gcd(a, gcd(b, c))$
- $gcd(x \cdot a, x \cdot b) = x \cdot gcd(a, b)$.
- $lcm(a, b)$ is the smallest number that both a and b divides.
- $lcm(a, b) = \frac{a \cdot b}{gcd(a,b)}$.
- $lcm(a, b, c) = lcm(a, lcm(b, c))$
- $gcd(a, b) = gcd(b, a\%b)$, if $b \neq 0$, $gcd(a, 0) = a$
- $gcd(a, a + 1) = 1$
- Let $f_a(x)$ be the power of x in a and $f_b(x)$ the power of x in b (the number of the times x appears in the prime factorization)
  - $gcd(a, b) = \prod x^{min(f_a(x), f_b(x))}$, for each factor x in a and b
  - $lcm(a, b) = \prod x^{max(f_a(x), f_b(x))}$, for each factor x in a or b
- $a \cdot x + b \cdot y = c$ such that a and b are constants, c will always be a multiple of $gcd(a, b)$, this is known as Linear Diophantine Equation

Sieve-like loop

We can use Sieve-like loop in finding all multiples of i in $O(n \cdot log(n))$.

```
for (int x = 2; x <= LIMIT; x++) {
    vector<int> multiplesOfX;
    for (int u = x; u <= LIMIT; u += x) {
        multiplesOfX.push_back(u);
    }
}
```

Resources

Must See:

- Marwan Nabeel - Number theory S1
- Marwan Nabeel - Number theory S2
- Marwan Nabeel - Number theory S3

Additional resources:

- https://www.youtube.com/playlist?list=PLD4iSlm2_YO2t6qH1R0eZN6rREtrpSgkb
- https://www.youtube.com/playlist?list=PLPt2dINI2MIY7l5zyFd1W28rei3b-AXaJ
- https://www.youtube.com/watch?v=-ptnoz7Us_I
- https://drive.google.com/file/d/1BcJTLjQaelZ4w_70oHKyImC2I8zLfyrt/view
- https://davidaltizio.web.illinois.edu/Divisors and Divisibility Overview.pdf
- https://cp-algorithms.com/algebra/binary-exp.html
- https://usaco.guide/gold/divisibility?lang=cpp
- Chapter 21 In this book (other chapters are good too)

Server Time: 2025-04-08 11:51:00 UTC+2