2024 Columbia Training Camp

Day 10: Number Theory

Christian Yongwhan Lim

Friday, August 30, 2024

Exit Survey

- **Please spend few minutes to complete the survey here:**
    - https://forms.gle/m268EmZWHFJBUDUa6

Overview

- Totient Function
- Mobius Function
- Binary Exponentiation
- Euclidean Algorithm
- Sieve of Eratosthenes
- Practice Problems
- Some Resources

Totient Function

- Euler's totient function, also known as **ϕ-function** $\phi(n)$, counts the number of integers between 1 and n inclusive, which are coprime to n.
- Two numbers are **coprime** if their greatest common divisor equals 1.

n	1	2	3	4	5	6	7	8	9	10	11	12
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4

Totient Function

$$\phi(p) = p - 1.$$

$$\phi(p^k) = p^k - p^{k-1}.$$

$$\phi(ab) = \phi(a) \cdot \phi(b).$$

$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$$

Totient Function

$$\phi(n) = \phi({p_1}^{a_1}) \cdot \phi({p_2}^{a_2}) \cdots \phi({p_k}^{a_k})$$

$$= \left({p_1}^{a_1} - {p_1}^{a_1-1}\right) \cdot \left({p_2}^{a_2} - {p_2}^{a_2-1}\right) \cdots \left({p_k}^{a_k} - {p_k}^{a_k-1}\right)$$

$$= p_1^{a_1} \cdot \left(1 - \frac{1}{p_1}\right) \cdot p_2^{a_2} \cdot \left(1 - \frac{1}{p_2}\right) \cdots p_k^{a_k} \cdot \left(1 - \frac{1}{p_k}\right)$$

$$= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

Totient Function: Implementation ($n^{1/2}$)

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            result -= result / i;
        }
    }
    if (n > 1) result -= result / n;
    return result;
}
```

Totient Function: Implementation (n log log n)

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

Möbius Inversion Formula

$$g(n) = \sum_{d|n} f(d) \quad \text{for every integer } n \geq 1$$

$$f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right) \quad \text{for every integer } n \geq 1$$

Möbius Function

- $\mu(n) = +1$ if n is a square-free positive integer with an even number of prime factors.
- $\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors.
- $\mu(n) = \quad 0$ if n has a squared prime factor.

Arithmetic Function

An arithmetic function a is

- **completely additive** if $a(mn) = a(m) + a(n)$ for all natural numbers m and n;
- **completely multiplicative** if $a(mn) = a(m)a(n)$ for all natural numbers m and n;

Two whole numbers m and n are called coprime if their greatest common divisor is 1, that is, if there is no prime number that divides both of them.

Then an arithmetic function a is

- **additive** if $a(mn) = a(m) + a(n)$ for all coprime natural numbers m and n;
- **multiplicative** if $a(mn) = a(m)a(n)$ for all coprime natural numbers m and n.

Binary Exponentiation: Main Idea

$$a^n = \begin{cases} 1 & \text{if } n == 0 \\ \left(a^{\frac{n}{2}}\right)^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ \left(a^{\frac{n-1}{2}}\right)^2 \cdot a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$$

Binary Exponentiation: Implementation (Recursive)

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Binary Exponentiation: Implementation (Iterative)

```cpp
long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

Euclidean Algorithm

- Given two non-negative integers a and b, we have to find their GCD (greatest common divisor), i.e. the largest number which is a divisor of both a and `b`. It's commonly denoted by `gcd(a,b)`.

$$\gcd(a,b) = \begin{cases} a, & \text{if } b = 0 \\ \gcd(b, a \bmod b), & \text{otherwise.} \end{cases}$$

Euclidean Algorithm: Implementation (Recursive)

```
int gcd (int a, int b) {
    return b ? gcd (b, a % b) : a;
}
```

Euclidean Algorithm: Implementation (Iterative)

- Since C++17, you may use gcd as a standard function in C++.

```
int gcd (int a, int b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}
```

Sieve of Eratosthenes: Main Idea

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	**11**	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	**11**	~~12~~	**13**	~~14~~	~~15~~	~~16~~

Sieve of Eratosthenes: Implementation

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++)
    if (is_prime[i])
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
```

Practice #1: 1491E: Fib-tree

Practice #2: 1497E2: Square-Free Division (hard version)

Practice #3: 1278F: Cards

Practice #4: 1034C: Region Separation

Practice #5: 1404D: Game of Pairs

Practice #6: 819C: Mister B and Beacons on Field

Practice #7: 1603D: Artistic Partition

Practice #8: 1973F: Maximum GCD Sum Queries

Online Platforms

- CodeForces
- Kattis
- acmicpc.net / solved.ac
- AtCoder
- CSES

Tutorial Sites

- usaco.guide
- cp-algorithms.com

CLI Symposium: https://cli.u.icpc.global/

- **ICPC Library (September 15)**
  - A living library of ICPC history including news, analytics, problem sets, solutions, judge archives, contest archives, awards, and other artifacts of ICPC history.
- **ICPC Compete (September 16)**
  - competitive programming competitions at different levels including news, instructions, and tools for entering ICPC contests, ICPC endorsed contests, and practice contests.
- **ICPC Educate (September 18)**
  - tools and resources for learning about competitive programming, including scholarly works, tutorials, algorithms from theory to application, strategies, how to coach, and programs for quick development.

CLI Symposium: ICPC Library (September 15)

- **Nikolay Kalinin (KAN)**: CodeForces: Contests by the Community, for the Community
- **Riku Kawasaki (maroonrk)**: Fun Facts about AtCoder
- **Suhyun Park (shiftpsh)**: solved.ac – Community Guide for Programming Challenges

CLI Symposium: ICPC Compete (September 16)

- **Antti Laaksonen (pllk)**: Creating Competitive Programming Material
- **Gennady Korotkevich (tourist)**: Training tips
- **Andrey Stankevich (andrewzta)**: Evolving of training tips: from beginners to world champions

CLI Symposium: ICPC Educate (September 18)

- **Erich Baker**: Introducing the Journal of Competitive Learning
- **Miguel Revilla Rodriguez**: ICPC Archive
- **Joshua Andersson**: Enhancements to the Problem Package Format
- **Matt Ellis**: JetBrains for ICPC
- **Christian Lim (yongwoods)**: ICPC Curriculum Committee


THANK YOU