michaelyql

Erdős-Szekeres Theorem

Every sequence of length n^2+1 has a monotonic subsequence of length at least $n+1$. Furthermore, there exists sequences of n^2 numbers that do not have a monotonic subsequence of length $n+1$.

Monotonic sequence: non-increasing or non-decreasing sequence e.g. [1,3,5,5], [-1,-3,-3]

Proof by Pigeonhole Principle (PHP):

Given a sequence $a_1, a_2, \ldots, a_{n^2+1}$, make a table

a_1	a_2	$\ldots$	a_{n^2+1}
d_1	d_2	$\ldots$	d_{n^2+1}
e_1	e_2	$\ldots$	e_{n^2+1}

where d_i = length of longest **non-increasing** subsequence starting at a_i and e_i = length of longest **non-decreasing** subsequence starting at a_i.

For $i \neq j$, it is not possible for $d_i = d_j$ and $e_i = e_j$ at the same time. (Proposition 1)

Case 1: if $a_i < a_j$, you can prepend a_i to the longest non-decreasing subsequence starting at a_j, so $e_i \geq e_j + 1$.

Case 2: if $a_i \geq a_j$, you can prepend a_i to the longest non-increasing subsequence starting at a_j, so $d_i \geq d_j + 1$.

Assume that there are no monotonic subsequences of length $n+1$, i.e. the longest monotone subsequence is of length n. Then the possible values of d_i and e_i range from 1 to n. Then there are a total of n^2 possible distinct pairs (d_i, e_i) for $1 \leq d_i, e_i \leq n$. But there are n^2+1 pairs in total. By PHP, at least two pairs (d_i, e_i) and (d_j, e_j) where $i \neq j$ have the same values, i.e. $d_i = d_j$ and $e_i = e_j$, which cannot be true by Proposition 1. Hence, our initial assumption that there exist no monotonic subsequence of length $n+1$ is wrong.

Consider the sequence

$$\begin{gathered}
n, n-1, n-2, \ldots, 1, \\
2n, 2n-1, \ldots, n+1, \\
3n, 3n-1, \ldots, 2n+1, \\
\vdots \\
n^2, n^2-1, \ldots, n^2-n+1
\end{gathered}$$

The longest monotonic subsequence is of length n (decreasing across a row or increasing down a column). You can guarantee a monotonic subsequence of length $n+1$ by adding just one more term. But for n^2 terms, you cannot guarantee that it has a monotonic subsequence of length $n+1$.

Dilworth's Lemma is a generalization of the Erdős-Szekeres Theorem. **Ramsey's Theorem** is a generalization of Dilworth's Lemma.

Ramsey Theory

On a social media platform, two people can be friends or not. Pick 6 arbitary users. Ramsey's Theory says that we can always find a group of 3 users who are all friends with the other two, or a group of 3 users which none of them are friends with each other, or both.

Reformulating using graphs: Construct a graph with 6 vertices, one for each person. Use a blue edge between two vertices if two people are friends and a red edge if they are not. The resulting graph is K_6 (a complete graph of order 6) with coloured edges. The claim is that there exists at least one **monochromatic triangle**, i.e. a triangle with all 3 edges of the same colour.

Arrow Notation:

Claim: If you colour the edges of K_6 arbitrarily with red and blue, you are guaranteed a red K_3 or a blue K_3.

Shorthand: $K_6 \to K_3, K_3$.

Proof: Consider any vertex a in the graph. It has 5 coloured edges. WLOG, by PHP, it will have at least 3 red (or 3 blue) edges. (If it has less than 3 red edges, then it has at least 3 blue edges, since all edges must be coloured, vice versa). Let the vertices at ends of these 3 red edges be b, c and d. Since the edge (a, b) and (a, c) are red, then the edge (b, c) must be blue. Similarly, (a, d) is red, so (b, d) must be blue. And lastly, (c, d) must be blue as well. Then the triangle formed by (b, c), (c, d) and (b, d) are all blue, thus forming a monochromatic triangle.

Proposition: $K_5 \not\to K_3, K_3$ i.e. it is possible to colour the edges of K_5 with red and blue such that there is no red or blue K_3. So 6 is a "threshold". Denote this by $r(3, 3) = 6$. $r(3, 3)$ is a **Ramsey number**.

Frank Ramsey's main idea was that, in a complex and large enough system, you can find any pattern - there is order in chaos. $r(a, b)$ is the threshold where your 'wish' for a pattern of (a, b) comes true. So $r(5, 8)$ is the smallest n such that $K_n \to K_5, K_8$. In general, $K_s \to K_n, K_m$ is a claim (which could be true or false) that if you arbitrarily colour the edges of K_s using two colours, red or blue, then you are **guaranteed** either a blue K_n or a red K_m (or both).

Some properties and notation:

1. $r(n) = r(n, n)$

2. $r(n, m) = r(m, n)$

3. $r(1, m) = 1$

4. $r(2, m) = m$ for $m > 1$ (Use blue at least once, or use red everywhere)

$K_5 \not\to K_2, K_6$, because even though you can get a blue K_2, you are not guaranteed a red K_6, since a red K_6 is cannot be formed.

$r(2, 2) = 2$, $r(3, 3) = 6$, $r(4, 4) = 18$, $r(5, 5) = ?$

$r(5, 5)$ is unknown, but has been proven to lie between $43 \leq r(5, 5) \leq 48$ (most recently proven to be ≤ 46 by computation, arXiv:2409.15709). Most Ramsey numbers are unknown. $36 \leq r(4, 6) \leq 40$, $102 \leq r(6, 6) \leq 161$. Erdős' fable:

> "If an alien force, vastly more powerful than us, landing on Earth and demanding the value of R(5,5) or they will destroy our planet, we should marshal all our computers and mathematicians and attempt to find the value. But if they ask for R(6,6), we should attempt to destroy the aliens."

Formally, Ramsey's Theorem says that, for each pair of positive integers k and l there exists an integer $R(k, l)$ (known as the Ramsey number) such that any graph with $R(k, l)$ nodes contains a **clique** with at least k nodes or an **independent set** with at least l nodes.

Another statement of the theorem is that for integers $k, l \geq 2$, there exists a least positive integer $R(k, l)$ such that no matter how the complete graph $K_{R(k,l)}$ is two-colored, it will contain a green subgraph K_k or a red subgraph K_l.

Vieta's Formula

Let $P(X) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$ be any polynomial with complex coefficients with roots $r_1, r_2, \ldots, r_n$ and let s_j be the j^{th} elementary symmetric polynomial of the roots. Then

$$s_1 = r_1 + r_2 + \cdots + r_n = -\frac{a_{n-1}}{a_n}$$

$$s_2 = r_1 r_2 + r_2 r_3 + \cdots + r_{n-1} r_n = \frac{a_{n-2}}{a_n}$$

$$\vdots$$

$$s_n = r_1 r_2 r_3 \ldots r_n = (-1)^n \frac{a_0}{a_n}$$

This can be compactly summarized as $s_j = (-1)^j \frac{a_{n-j}}{a_n}$ for some j such that $1 \leq j \leq n$.

For quadratic equations, if $f(x) = ax^2 + bx + c$ with roots α and β, then $\alpha + \beta = -\frac{b}{a}$, $\alpha\beta = \frac{c}{a}$. If the sum and product of roots are given, then $x^2 - (\alpha + \beta)x + (\alpha\beta) = x^2 - bx + c = 0$. Then using the discriminant formula, $D = b^2 - 4ac$, the roots will be $\alpha = \frac{b-\sqrt{D}}{2}$ and $\beta = \frac{b+\sqrt{D}}{2}$. If $D < 0$, there are no real roots. If $D = 0$, $\alpha = \beta$.

LCM Sum Function

Given n, calculate the sum $LCM(1, n) + LCM(2, n) + \cdots + LCM(n, n)$ where $LCM(i, n)$ denotes the least common multiple of i and n.

$$S(n) = \frac{n}{2}(\sum_{d|n}(\phi(d) \times d) + 1)$$

where $\phi(n)$ is Euler's totient function which calculates the number of positive integers less than or equal to n which are coprime with n.

Proof (taken from https://forthright48.com/spoj-lcmsum-lcm-sum/):

Let $S(n) = LCM(1, n) + \cdots + LCM(n, n)$.

$$\begin{aligned}
&S(n) - LCM(n, n) = LCM(1, n) + \cdots + LCM(n-1, n) \\
&S(n) - n = LCM(1, n) + \cdots + LCM(n-1, n) \\
&S(n) - n = LCM(n-1, n) + \cdots + LCM(1, n) \\
&2(S(n) - n) = (LCM(1, n) + LCM(n-1, n)) + \cdots + (LCM(n-1, n) + LCM(1, n)) \\
&\text{Let } x = LCM(a, n) + LCM(n-a, n) = \frac{an}{gcd(a, n)} + \frac{(n-a)n}{gcd(n-a, n)}
\end{aligned}$$

If c divides a and b, then c divides $a + b$ and $a - b$. So if $g = gcd(a, n)$ divides a and n, then g also divides $n - a$. So $gcd(a, n) = gcd(n - a, n)$. Then

$$x = \frac{an + (n-a)n}{gcd(a, n)} = \frac{an + n^2 - an}{gcd(a, n)} = \frac{n^2}{gcd(a, n)}$$

Substituting in,

$$2(S(n) - n) = \sum_{i=1}^{n-1} \frac{n^2}{gcd(i, n)}$$

$$2(S(n) - n) = n \sum_{i=1}^{n-1} \frac{n}{gcd(i, n)}$$

Since $g = gcd(i,n)$ divides n, we can list the possible values of $gcd(i,n)$ by finding the divisors of n. The number of times that $d = gcd(i,n)$ occurs is $\phi(\frac{n}{d})$. This is because $gcd(i,n) = d$ can be rewritten as $gcd(kd,n) = d$ for some k. Dividing throughout by d, $gcd(k, \frac{n}{d}) = 1$. The number of positive integers that are coprime with $\frac{n}{d}$, i.e. $gcd(k, \frac{n}{d}) = 1$ is exactly $\phi(\frac{n}{d})$. Hence

$$2(S(n) - n) = n\sum_{i=1}^{n-1} \frac{n}{gcd(i,n)}$$

$$2(S(n) - n) = n\sum_{d|n, d\neq n} \phi(\frac{n}{d}) \times \frac{n}{d}$$

$$\text{Let } d' = \frac{n}{d}\text{, then } 2(S(n) - n) = n\sum_{d'|n, d'\neq 1} \phi(d') \times d'$$

$$\text{since } d' = \frac{n}{d} \text{ is also a divisor of } n$$

$$2(S(n) - n) = n\sum_{d|n, d\neq 1} \phi(d) \times d$$

$$2(S(n) - n) = n(\sum_{d|n}(\phi(d) \times d) - 1) \text{ (since } \phi(1) \times 1 = 1)$$

$$2(S(n) - n) = n\sum_{d|n}(\phi(d) \times d) - n$$

$$2S(n) = n + n\sum_{d|n}(\phi(d) \times d)$$

$$2S(n) = n(\sum_{d|n}(\phi(d) \times d) + 1)$$

$$\therefore S(n) = \frac{n}{2}(\sum_{d|n}(\phi(d) \times d) + 1)$$

The sequence of LCM sum values can be found at https://oeis.org/A051193.

Computation wise, we can first precompute the values of $\phi(d)$, then iterate through the divisors of n.

See LightOJ: LCM Extreme (https://lightoj.com/problem/lcm-extreme)

GCD Sum Function

Convolution

$$(f * g)(t) := \int_{-\infty}^{\infty} f(x)g(t-x)dx$$

Dirichlet Convolution

$$(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$$

For the identity function $I(n) = n$ and the constant function $\mathbf{1}(n) = 1$, for all natural numbers n,

$$(I * \mathbf{1})(n) = \sum_{d|n} I(d)\mathbf{1}(\frac{n}{d}) = \sum_{d|n} d \cdot 1 = \sum_{d|n} d = \sigma(n)$$

where $\sigma(n)$ is the sum of the positive divisors of n. We write $I * \mathbf{1} = \sigma$

Properties:

1. Convolution is commutative: $f * g = g * f$

2. It is associative: $(f * g) * h = f * (g * h)$

3. It is distributive over addition: $f * (g + h) = (f * g) + (f * h)$

4. It has an identity: define $e(n) = \lfloor \frac{1}{n} \rfloor = \begin{cases} 1 \text{ if } n = 1 \\ \\ 0 \text{ otherwise} \end{cases}$. Then $e * f = f * e = f$ for any f.

5. The Dirichlet convolution of two multiplicative functions is multiplicative.

Sum Convolution

GCD Convolution

Circular Convolution

Discrete Convolution

Laplace Transform

Fourier Transform

Z-Transform

Division Algorithm

For $a, b \in \mathbb{Z}$ and $b > 0$, we can always write $a = qb + r$ where $0 \leq r < b$ and $q \in \mathbb{Z}$. Moreover, given a, b, there is only one pair q, r which satisfy the constraints.

Bezout's Identity

Let a and b be integers with greatest common divisor d. Then there exist integers x and y such that $ax + by = d$. Moreover, the integers of the form $az + bt$ are exactly the multiples of d. In general,

$$\text{If } gcd(a_1, a_2, \ldots, a_n) = d, \text{ then } \exists x_1, x_2, \ldots, x_n \in \mathbb{Z} \text{ s.t. } d = a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

where d is the smallest number of this form, and every number of this form is a multiple of d.

Euclidean Algorithm for GCD

Define $gcd(a, b) = \max\{k > 0 : (k|a) \text{ and } (k|b)\}$.

Also define $gcd(0, p) = p$ and $gcd(0, 0) = 0$ to preserve the associativity of gcd.

Since the function is associative, $gcd(a, b, c) = gcd(a, gcd(b, c))$.

If g divides a and b, then g also divides $a - b$.

If g divides $a - b$ and b, then it also divides $a = (a - b) + b$.

So the set of common divisors coincide.

b is subtracted from a at least $\lfloor \frac{a}{b} \rfloor$ times before a becomes smaller than b, at which point we can swap the two, i.e. $gcd(a, b) = gcd(a - b, b)$. To speed it up,

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

Extended Euclidean Algorithm

The Euclidean Algorithm computes the GCD for two numbers a and b. The Extended Euclidean Algorithm finds a way to represent the GCD as a linear combination of a and b, i.e. $ax + by = gcd(a, b)$. Since the Euclidean Algorithm ends with $a = g$ and $b = 0$, for the base case, $x = 1$ and $y = 0$, so that $ax + by = g \cdot 1 + 0 \cdot 0 = g$.

Now we want to transition from $(b, a \bmod b)$ back to (a, b). Assume we know (x_1, y_1) for $(b, a \bmod b)$, then $b \cdot x_1 + (a \bmod b) \cdot y_1 = g$.

So $b \cdot x_1 + (a - \lfloor \frac{a}{b} \rfloor \cdot b) \cdot y_1 = g$

Rearranging, $g = a \cdot y_1 + b \cdot (x_1 - y_1 \cdot \lfloor \frac{a}{b} \rfloor)$. So $\begin{cases} x = y_1 \\ y = x_1 - y_1 \cdot \lfloor \frac{a}{b} \rfloor \end{cases}$

```cpp
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

Lowest Common Multiple

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

Diophantine Equations

Diophantine equation = an equation where solutions are restricted to integers. For example, $ax + by = c$, $w^3 + x^3 = y^3 + z^3$, $x^n + y^n = z^n$ (Fermat's Last Theorem).

Linear Diophantine Equation

$ax + by = c$ has solutions iff c is a multiple of $gcd(a, b)$. If (x, y) is a solution, then $(x + kv, y - ku)$ are also solutions for any arbitrary k where u and v are the quotients of a and b divided by $gcd(a, b)$.

Digital Roots

Prime Factorization Theorem (Fundamental Theorem of Arithmetic)

Every integer greater than 1 can be represented uniquely as a product of prime powers.

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

Prime Counting Function

Sieve of Erastosthenes

```
vector<int> primes;
void sieve(n) {
  vector<bool> isPrime(n + 1);
  for (int i = 2; i * i <= n; i++) {
    if (isPrime[i]) {
      for (int j = i * i; j <= n; j += i) isPrime[j] = false;
    }
  }
  for (int i = 2; i <= n; i++) {
    if (isPrime[i]) primes.push_back(i);
  }
}
```

Any composite number can be written as $a \times b$, where a and b are factors of n. If both of them were greater than $\sqrt{n}$, then $ab > n$ which is a contradiction. The algorithm iterates until `i * i <= n`, since all primes less than `i` and their multiples would have been marked as non-prime in previous iterations. The time complexity is $O(n \log \log n)$, bounded by the prime harmonic series.

$$\sum_{p \text{ prime}} \frac{1}{p} = \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \cdots = \infty$$

Linear Sieve

The idea is to mark numbers with their smallest prime factor. For example, $3 \times 7 = 21$, it's smallest prime factor is 3, so we can mark `spf[21]=3`.

```
const int N = 2e5 + 7;
bool isPrime[N];
vector<int> primes;
int spf[N];

void fast_sieve() {
  rep(i, 2, N) isPrime[i] = true;
  for (int i = 2; i < N; i++) {
    if (isPrime[i]) {
      primes.push_back(i);
      spf[i] = i;
    }

    for (int j = 0; j < primes.size() && i * primes[j] < N && primes[j] <= spf[i]; j++) {
      isPrime[i * primes[j]] = false;
      spf[i * primes[j]] = primes[j];
    }
  }
}
```

Legendre's Formula

Number of Divisors

$$\tau(n) = \prod_{i=1}^{k} (\alpha_i + 1)$$

```
long long numberOfDivisors(long long num) {
    long long total = 1;
    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);
            total *= e + 1;
        }
    }
    if (num > 1) {
        total *= 2;
    }
    return total;
}
```

Sum of Divisors

$$\begin{aligned}
\sigma(n) &= (1 + p_1 + p_1^2 + \dots + p_1^{\alpha_1}) + p_2(1 + p_1 + p_1^2 + \dots + p_1^{\alpha_1}) + p_2^2(1 + p_1 + \dots) + \dots \\
&= (1 + p_1 + p_1^2 + \dots + p_1^{\alpha_1}) \cdot (1 + p_2 + p_2^2 + \dots + p_2^{\alpha_2}) \cdot (\dots) \\
&= \prod_{i=1}^{k} (1 + p_i + p_i^2 + \dots + p_i^{\alpha_i}) \\
&= \prod_{i=1}^{k} \frac{p_i^{\alpha_i + 1} - 1}{p_i - 1}
\end{aligned}$$

```
long long SumOfDivisors(long long num) {
    long long total = 1;

    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) {
        total *= (1 + num);
    }
    return total;
}
```

Product of Divisors

$$\mu(n) = \prod_{d|n} d = \begin{cases} n^{\tau(n)/2}, & \text{if } \tau(n) \text{ is even, i.e. } n \text{ is not a perfect square} \\ n^{(\tau(n)-1)/2} \cdot \sqrt{n}, & \text{if } \tau(n)/2 \text{ is odd, i.e. } n \text{ is a perfect square} \end{cases}$$

$\tau(n)$ gives the number of factors of n. For each divisors of n, it can be "paired" with $\frac{n}{d}$, so there are, in total, $\tau(n)/2$ such pairs whose product is exactly n.

Perfect Number

A perfect number n is a number such that $n = \sigma(n) - n$, i.e. n is equal to the sum of its divisors from 1 to $n-1$.

E.g. $28 = 1 + 2 + 4 + 7 + 14$

Euler's Totient Function

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

$$\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \dots (1 - \frac{1}{p_k}) = \prod_{i=1}^{k} p_i^{\alpha_i - 1}(p_i - 1)$$

where $\phi(n)$ denotes the number of positive integers less than or equal to n that are *coprime* with n, i.e. $gcd(i, n) = 1$.

```
int phi(int n) { // O(sqrt(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;
}

void phi_1_to_n(int n) { // O(n log log 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;
        }
    }
}
```

```python
from math import floor

def prime_factors(n):
    """Returns a list of distinct prime factors of n."""
    i = 2
    factors = set()
    while i * i <= n:
        while n % i == 0:
            factors.add(i)
            n //= i
        i += 1
    if n > 1:
        factors.add(n)
    return list(factors)

def euler_totient(n): # O(sqrt(n))
    primes = prime_factors(n)
    result = n
    for p in primes:
        result *= (1 - 1 / p)
    return int(result)
```

For prime numbers,

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

For powers of prime numbers,

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

The function is multiplicative. For any two coprime numbers a and b,

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

For semiprime numbers $s = pq$,

$$\phi(s) = \phi(p)\phi(q) = (p - 1)(q - 1) = pq - p - q + 1 = n - (p + q) + 1$$

The function can be used to calculate the sum of divisors.

$$\sum_{d|n} \phi(d) = n$$

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

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}
```

Totient Numbers

A totient number is a value of Euler's totient function: that is, an m for which there is at least one n for which $\phi(n) = m$.

The *valency* or *multiplicity* of a totient number m is the number of solutions to this equation.

A nontotient is a natural number which is not a totient number.

Every odd integer exceeding 1 is trivially a nontotient.

There are infinitely many even nontotients, and every positive integer has a multiple which is an even nontotient.

Ford (1999) proved that for every integer $k \geq 2$ there is a totient number m of multiplicity k: that is, for which the equation $\phi(n) = m$ has exactly k solutions.

However, no number m is known with multiplicity $k = 1$. Carmichael's totient function *conjecture* is the statement that there is no such m (For every n there is at least one other integer $m \neq n$ such that $\phi(m) = \phi(n)$).

A *perfect totient number* is an integer that is equal to the sum of its iterated totients. That is, we apply the totient function to a number n, apply it again to the resulting totient, and so on, until the number 1 is reached, and add together the resulting sequence of numbers; if the sum equals n, then n is a perfect totient number. For example, $\phi(9) = 6$, $\phi(6) = 2$, $\phi(2) = 1$, $9 = 6 + 2 + 1$, so 9 is a perfect totient number.

$$\phi^i(n) = \begin{cases} \phi(n) & \text{if } i = 1 \\ \phi(\phi^{i-1}(n)) & \text{if } i \geq 2 \end{cases}$$

Then if c is the integer such that $\phi^c(n) = 2$, n is a perfect totient number if

$$n = \sum_{i=1}^{c+1} \phi^i(n)$$

It can be observed that many perfect totient are multiples of 3; in fact, 4375 is the smallest perfect totient number that is not divisible by 3. All powers of 3 are perfect totient numbers, as may be seen by induction using the fact that

$$\phi(3^k) = \phi(2 \times 3^k) = 2 \times 3^{k-1}$$

Euler's Theorem

If a and n are coprime,

$$a^{\phi(n)} \equiv 1 \mod n$$

The special case where n is prime is called Fermat's Little Theorem. See

`https://www.youtube.com/watch?v=5pswKNgVZSg&t=159s&ab_channel=MuPrimeMath`

for an elegant proof.

Other Formulae of Totient Function

$$a \mid b \Rightarrow \phi(a) \mid \phi(b)$$

$$m \mid \phi(a^m - 1)$$

$$\phi(mn) = \phi(m)\phi(n) \cdot \frac{d}{\phi(d)} \text{ where } d = gcd(m, n)$$

$$\phi(2m) = \begin{cases} 2\phi(m) & \text{if } m \text{ is even} \\ \phi(m) & \text{if } m \text{ is odd} \end{cases}$$

$$\phi(lcm(m, n)) \cdot \phi(gcd(m, n)) = \phi(m) \cdot \phi(n), \text{ from } lcm(m, n) \cdot gcd(m, n) = m \cdot n$$

$$\phi(n) \text{ is even for } n \geq 3. \text{ If } n \text{ has } r \text{ distinct odd prime factors, } 2^r \mid \phi(n)$$

$$\frac{\phi(n)}{n} = \frac{\phi(\text{rad}(n))}{\text{rad}(n)}, \text{ where rad}(n) \text{ is the radical of } n \text{ (product of all distinct primes of } n)$$

$$\sum_{d \mid n} \frac{\mu^2(d)}{\phi(d)} = \frac{n}{\phi(n)}$$

$$\sum_{\substack{1 \leq k \leq n-1 \\ gcd(k,n)=1}} k = \frac{1}{2} n\phi(n) \text{ for } n > 1$$

Euler's totient function is a type of **Arithmetic Function**.

Carmichael Function

Totient Summary Function

Ramanujan's Sum

Menon's Identity

$$\sum_{\substack{1 \leq k \leq n \\ gcd(k,n)=1}} \gcd(k-1, n) = \phi(n)d(n)$$

where $d(n)$ is the number of divisors of n.

Fermat's Little Theorem

If p is a prime number, then for any integer a, the number $ap - a$ is an integer multiple of p.

$$a^p \equiv a \mod p$$

If a and p are coprime, then $a^{p-1} - 1$ is an integer multiple of p,

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

Combinatorial proof: Suppose there is a necklace of p beads, each of which can be coloured any one of a colours. Then there are in total $a \cdot a \ldots a = a^p$ ways to colour the beads. There are exactly a ways where all the beads are of the same colour. For each of the remaining $a^p - a$ ways, there are $p - 1$ configurations that are rotationally equivalent to that configuration, which form a group of p configurations. Hence $a^p - a$ must be divisible by p. Hence $a^p \equiv a \mod p$.

Fermat's little theorem is the basis for the Fermat primality test.

Pseudoprimes

Prime Powers

Primorial Primes

A primorial prime is a prime number of the form $p_n\#+1$, where $p_n\#$ is the product of the first n prime numbers.

Primality Tests

(From Wikipedia) A primality test is an algorithm for determining whether an input number is prime. Unlike integer factorization, primality tests do not generally give prime factors, only stating whether the input number is prime or not. Factorization is thought to be a computationally difficult problem, whereas primality testing is comparatively easy (its running time is polynomial in the size of the input). Some primality tests prove that a number is prime, while others like Miller–Rabin prove that a number is composite. Therefore, the latter might more accurately be called compositeness tests instead of primality tests.

Fermat's Primality Test

Miller-Rabin Primality Test

Integer Factorization

Trial Division

Fermat's Factorization Method

Pollard-Rho

Quadratic Sieve

Elliptic Curve Factorization

General Number Field Sieve

Modular Arithmetic

Modular Inverse

Find an integer x such that $a \cdot x \equiv 1 \mod m$. Denote the modular inverse with x^{-1}.

This is equivalent to saying $a \cdot x + m \cdot y = 1$. This is a Linear Diophantine equation in two variables, where $\gcd(a, b) = 1$. Taking modulo m on both sides of the equation, it reduces to $a \cdot x \equiv 1 \mod m$. Thus we can find x^{-1} by the Extended Euclidean Algorithm.

```
int x, y;
int g = extended_euclidean(a, m, x, y);
if (g != 1) {
    cout << "No solution!";
}
else {
    x = (x % m + m) % m;
    cout << x << endl;
}
```

For prime modulus m,

$$a^{m-2} \equiv a^{-1} \mod m$$

For an arbitrary m but a is coprime with m,

$$a^{\phi(m)-1} \equiv a^{-1} \mod m$$

The modular inverse can then be computed by binary exponentiation.

```cpp
ll bpow(ll a, ll b) {
  a %= MOD;
  ll res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % MOD;
    b >>= 1;
    a = a * a % MOD;
  }
  return res;
}
```

We can also find the modular inverse using Euclidean Division. Given prime modulus $m > a$, $m = k \cdot a + r$, where $k = \lfloor \frac{m}{a} \rfloor$ and $r = m \mod a$, then

$\Leftrightarrow 0 \equiv k \cdot a + r \mod m$

$\Leftrightarrow r \equiv -k \cdot a \mod m$

$\Leftrightarrow r \cdot a^{-1} \equiv -k \mod m$

$\Leftrightarrow a^{-1} \equiv -k \cdot r^{-1} \mod m$

This only holds if m is prime, as otherwise r^{-1} is not guaranteed to exist.

```cpp
int inv(int a) {
  return a <= 1 ? a : m - (long long)(m/a) * inv(m % a) % m;
}
```

The run time is between $O(\frac{\log m}{\log \log m})$ and $O(m^{\frac{1}{3}-\frac{2}{177}+\epsilon})$. In practice this implementation is fast, e.g. for the modulus 10^9+7 it will always finish in less than 50 iterations. We can precompute the modular inverse for every number in the range $[1, m-1]$ in $O(m)$.

```
inv[1] = 1;
for(int a = 2; a < m; ++a)
    inv[a] = m - (long long)(m/a) * inv[m%a] % m;
```

Factorial Modulo

Chinese Remainder Theorem

Arithmetic Functions

Divisor Sum

Group Theory

Abelian Group

Ring Theory

Combinatorics

(Restricted) Inclusion-Exclusion Principle

To compute the size of a union of multiple sets, it is necessary to sum the sizes of these sets **separately**, and then subtract the sizes of all **pairwise** intersections of the sets, then add back the size of the intersections of **triples** of the sets, subtract the size of **quadruples** of the sets, and so on, up to the intersection of **all** sets.

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i\le j\le n} |A_i \cap A_j| + \sum_{1\le i\le j\le k\le n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n-1}|A_1 \cap \cdots \cap A_n|$$

It can be used to solve tasks asking to "find the number of ways".

Inclusion-Exclusion Principle

Assume we are given a set X called the *universe*, and a set $E = \{e_1, e_2, \dots, e_n\}$ of **properties** that the elements of X may or may not possess. Let A_i be the subset of elements that possess the property e_i (elements in this set may possess other properties as well). Then $|X \setminus \bigcup_{i=1}^{n} A_i|$ is the number of elements that possess none of the properties. Use the notation

$$N_{\supseteq T} := \#\{x \in X : x \text{ possesses } \textit{at least} \text{ the properties of } T\}$$

$$N_{=T} := \#\{x \in X : x \text{ possesses } \textit{precisely} \text{ the properties of } T\}$$

$$N_{=\emptyset} = \sum_{T\subseteq E} (-1)^{|T|} N_{\supseteq T} = \sum_{k=0}^{n} (-1)^k \sum_{T:|T|=k} N_{\supseteq T}$$

When $N_{\supseteq T}$ depends only on the size $|T| = k$, we can write $N_{\supseteq T} = N_{\ge k}$ for $|T| = k$, and call E a *homogeneous* set of properties.

$$N_{=\emptyset} = N_{=0} = \sum_{k=0}^{n} (-1)^k \binom{n}{k} N_{\ge k}$$

(since it doesn't matter what properties T includes as long as $|T| = k$, so we can choose any k out of the n properties, thus we have to multiply $N_{\ge k}$ by $\binom{n}{k}$). By letting $e_i = \{i \in A_i\}$, we arrive at the restricted IEP.

Tasks

(3rd Lebanese Collegiate Programming Contest - Task 4135)

Given an integer N and a range $[L, R]$, find the number of integers $L \leq x \leq R$ such that x is coprime with N, i.e. $\gcd(x, N) = 1$. Constraints: $1 \leq N \leq 10^9$, $1 \leq L \leq R \leq 10^{15}$.

(HackerRank - K-Inversion Permutations)

Given an integer N, find the number of permutations of length N that have exactly K inversions mod $10^9 + 7$. (An inversion for a permutation π is defined as an ordered pair (i, j) such that $i < j$ and $\pi(i) > \pi(j)$). Constraints: $1 \leq N \leq 10^5$, $1 \leq K \leq \min(10^5, \binom{n}{2})$.

(2018 Multi-University Training Contest 8, HDU OJ - Character Encoding)

Given an encoding system of a list of characters of size n where each character is assigned a number between 0 and $n - 1$, how many different words of length m are there such that the sum of the encoded numbers of characters in that word is equal to k (mod 998244353)? Constraints: $1 \leq T \leq 400$, $1 \leq n, m \leq 10^5, 0 \leq k \leq 10^5$. The sum of n, sum of m, and sum of k don't exceed 5×10^6.

Lucas' Theorem

$$\binom{m}{n} = \prod_{i=0}^{k} \binom{m_i}{n_i} \mod p$$

where p is a prime, $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$ and $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$ are the base p expansions of m and n respectively, and $\binom{m}{n} = 0$ if $m < n$

A binomial coefficient $\binom{m}{n}$ is divisible by a prime p if and only if at least one of the base p digits of n is greater than the corresponding digit of m.

In particular, $\binom{m}{n}$ is odd iff the bits in the binary expasion of n are a subset of the bits of m, i.e. $n\&m == n$

```
// Precomputing parity of nCk mod 2
int MAX_N = 1e6 + 5;
int c[MAX_N]; // c[n] is the power of 2 in n!

c[0] = c[1] = 0;
for (int i = 2; i <= MAX_N; i++) {
  c[i] = c[i - 1];
  int x = i;
  while (x % 2 == 0) {
    x /= i;
    c[i]++;
  }
}
// if c[n] == c[i] + c[n - i], then the parity is 1, else the parity is 0
```

Binomial Coefficients

Recursive Formula

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Base case: $\binom{n}{0} = \binom{n}{n} = 1$

Multiplicative Formula

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{n(n-1)(n-2)\dots(n-(k-1))}{k(k-1)(k-2)\dots 1} = \prod_{i}^{k} \frac{n+1-i}{i}$$

Mean Value Theorem

Generating Functions

Split $2n$ elements into n unordered pairs

$$\frac{(2n)!}{n! \cdot 2^n}$$

Arrange the $2n$ elements in a sequence $a_1, a_2, \dots, a_{2n-1}, a_{2n}$. There are $(2n)!$ such permutations of sequences.

Form pairs between $(a_1, a_2), (a_3, a_4), \dots, (a_{2n-1}, a_{2n})$. For each pair, their internal order does not matter. (a_1, a_2) is the same pairing as (a_2, a_1). So divide by 2^n.

Then for each of the n pairs, their relative order doesn't matter, so divide by $n!$.

Permutation Involution

Subset Sum

MEX (Minumum Excluded Value)

```
// O(n) implementation for relatively small n (<=1e8)
int mex(vector<int>& a) {
  int n = a.size();
  vector<bool> mark(n + 1, false);
  for (int x : a) {
    if (x < n) mark[x] = true;
  }
  for (int i = 0; i <= n; i++) {
    if (!mark[i]) return i;
  }
  return n + 1; // theoretically unreachable
}
```

Properties of XOR

The minimum XOR pair of an array can be found by first sorting the array, then only comparing adjacent elements in the sorted array. Intuition: elements close together in terms of magnitude and value will have fewer different bits.