

Cheatsheet de Aritmética y Fórmulas

Para Programación Competitiva

Competitive Programming Reference

Abstract

Este documento contiene una colección exhaustiva de fórmulas aritméticas, progresiones, sumatorias, identidades y algoritmos útiles para programación competitiva. Incluye implementaciones en C++ y ejemplos prácticos.

Contents

1 Progresiones y Secuencias

1.1 Progresión Aritmética (PA)

Una progresión aritmética es una secuencia donde la diferencia entre términos consecutivos es constante.

Forma general:

$$a_n = a_1 + (n - 1)d$$

donde:

- a_1 = primer término
- d = diferencia común
- n = posición del término

Suma de los primeros n términos:

$$S_n = \frac{n}{2}(a_1 + a_n) = \frac{n}{2}(2a_1 + (n - 1)d)$$

Término general conociendo dos términos:

$$d = \frac{a_m - a_k}{m - k}$$

```
1 // Progresion Aritmetica
2 struct ArithmeticProgression {
3     long long a1, d; // primer termino y diferencia
4
5     // n-esimo termino (1-indexed)
6     long long term(long long n) {
7         return a1 + (n - 1) * d;
8     }
9
10    // Suma de primeros n terminos
11    long long sum(long long n) {
12        return n * (2 * a1 + (n - 1) * d) / 2;
13    }
14
15    // Suma desde termino i hasta j (inclusive)
16    long long rangeSum(long long i, long long j) {
17        return sum(j) - sum(i - 1);
18    }
19
20    // Encontrar cuantos terminos hay hasta llegar a valor x
21    long long countUntil(long long x) {
22        if ((x - a1) % d != 0) return -1; // No es termino
23        long long n = (x - a1) / d + 1;
24        return n >= 1 ? n : -1;
25    }
26 };
```

1.2 Progresión Geométrica (PG)

Una progresión geométrica es una secuencia donde el cociente entre términos consecutivos es constante.

Forma general:

$$a_n = a_1 \cdot r^{n-1}$$

donde:

- a_1 = primer término
- r = razón común
- n = posición del término

Suma de los primeros n términos:

$$S_n = \begin{cases} a_1 \cdot \frac{r^n - 1}{r - 1} & \text{si } r \neq 1 \\ n \cdot a_1 & \text{si } r = 1 \end{cases}$$

Suma infinita (si $|r| < 1$):

$$S_\infty = \frac{a_1}{1 - r}$$

```
1 // Progresion Geometrica
2 template<typename T>
3 struct GeometricProgression {
4     T a1, r; // primer termino y razon
5
6     // n-esimo termino
7     T term(int n) {
8         return a1 * pow(r, n - 1);
9     }
10
11     // Suma de primeros n terminos
12     T sum(int n) {
13         if (abs(r - 1.0) < 1e-9) return n * a1;
14         return a1 * (pow(r, n) - 1) / (r - 1);
15     }
16
17     // Suma infinita (solo si |r| < 1)
18     T infiniteSum() {
19         if (abs(r) >= 1) return -1; // No converge
20         return a1 / (1 - r);
21     }
22 };
23
24 // Version con exponenciacion binaria para enteros
25 struct GeometricProgressionMod {
26     long long a1, r, mod;
27
28     long long binpow(long long base, long long exp) {
29         long long res = 1;
30         base %= mod;
31         while (exp > 0) {
32             if (exp & 1) res = (res * base) % mod;
33             base = (base * base) % mod;
34             exp >>= 1;
35         }
36         return res;
37     }
38
39     long long term(long long n) {
40         return (a1 * binpow(r, n - 1)) % mod;
41     }
42
43     // Suma usando formula geometrica modular
44     long long sum(long long n) {
45         if (r == 1) return (n * a1) % mod;
46
47         long long rn = binpow(r, n);
48         long long numerator = (a1 * (rn - 1 + mod)) % mod;
49
50         // Necesitamos inverso modular de (r - 1)
51         long long denominator = (r - 1 + mod) % mod;
52         long long inv = binpow(denominator, mod - 2); // Si mod es primo
53
54         return (numerator * inv) % mod;
55     }
56 };
```

1.3 Progresión Armónica

Una progresión armónica es una secuencia cuyos recíprocos forman una progresión aritmética.

Forma general:

$$\frac{1}{a_n} = \frac{1}{a_1} + (n-1)d$$

Serie armónica:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + \gamma$$

donde $\gamma \approx 0.5772$ es la constante de Euler-Mascheroni.

```
1 // Serie Armonica
2 double harmonicSum(int n) {
3     double sum = 0;
4     for (int i = 1; i <= n; i++) {
5         sum += 1.0 / i;
6     }
7     return sum;
8 }
9
10 // Aproximacion usando logaritmo
11 double harmonicApprox(int n) {
12     const double EULER_GAMMA = 0.5772156649;
13     return log(n) + EULER_GAMMA;
14 }
15
16 // Serie armonica generalizada: sum(1/i^p)
17 double generalizedHarmonic(int n, double p) {
18     double sum = 0;
19     for (int i = 1; i <= n; i++) {
20         sum += 1.0 / pow(i, p);
21     }
22     return sum;
23 }
```

1.4 Secuencia de Fibonacci

Definición recursiva:

$$F_n = F_{n-1} + F_{n-2}, \quad F_0 = 0, \quad F_1 = 1$$

Fórmula de Binet (forma cerrada):

$$F_n = \frac{\phi^n - \psi^n}{\sqrt{5}}$$

donde $\phi = \frac{1+\sqrt{5}}{2}$ (proporción áurea) y $\psi = \frac{1-\sqrt{5}}{2}$.

Propiedades importantes:

$$\begin{aligned} F_{n+m} &= F_n F_{m+1} + F_{n-1} F_m \\ F_{2n} &= F_n (2F_{n+1} - F_n) \\ F_n^2 + F_{n+1}^2 &= F_{2n+1} \\ \gcd(F_m, F_n) &= F_{\gcd(m, n)} \end{aligned}$$

Suma de Fibonacci:

$$\sum_{i=0}^n F_i = F_{n+2} - 1$$

```
1 // Fibonacci con matriz (O(log n))
2 const long long MOD = 1e9 + 7;
3
4 struct Matrix2x2 {
5     long long m[2][2];
6
7     Matrix2x2() {
```

```

8      memset(m, 0, sizeof(m));
9  }
10
11  Matrix2x2 operator*(const Matrix2x2& other) const {
12      Matrix2x2 res;
13      for (int i = 0; i < 2; i++) {
14          for (int j = 0; j < 2; j++) {
15              for (int k = 0; k < 2; k++) {
16                  res.m[i][j] = (res.m[i][j] +
17                               m[i][k] * other.m[k][j]) % MOD;
18              }
19          }
20      }
21      return res;
22  }
23 };
24
25 Matrix2x2 matpow(Matrix2x2 base, long long exp) {
26     Matrix2x2 res;
27     res.m[0][0] = res.m[1][1] = 1;
28
29     while (exp > 0) {
30         if (exp & 1) res = res * base;
31         base = base * base;
32         exp >>= 1;
33     }
34     return res;
35 }
36
37 long long fibonacci(long long n) {
38     if (n == 0) return 0;
39     if (n == 1) return 1;
40
41     Matrix2x2 base;
42     base.m[0][0] = base.m[0][1] = base.m[1][0] = 1;
43     base.m[1][1] = 0;
44
45     Matrix2x2 res = matpow(base, n - 1);
46     return res.m[0][0];
47 }
48
49 // Suma de primeros n fibonacci
50 long long fibonacciSum(long long n) {
51     return (fibonacci(n + 2) - 1 + MOD) % MOD;
52 }

```

1.5 Números de Catalan

Los números de Catalan aparecen en muchos problemas combinatorios.

Fórmula:

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

Recurrencia:

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

Aplicaciones:

- Número de formas de triangular un polígono de $n + 2$ lados
- Número de árboles binarios con n nodos
- Número de formas válidas de colocar n pares de paréntesis
- Caminos en una cuadrícula que no cruzan la diagonal

```

1 // Numeros de Catalan
2 vector<long long> catalan(int n, long long mod) {
3     vector<long long> C(n + 1);
4     C[0] = 1;
5
6     for (int i = 1; i <= n; i++) {
7         C[i] = 0;
8         for (int j = 0; j < i; j++) {
9             C[i] = (C[i] + C[j] * C[i - 1 - j]) % mod;
10        }
11    }
12
13    return C;
14 }
15
16 // Formula directa usando combinatoria
17 long long catalanDirect(int n, long long mod) {
18     // C_n = (2n)! / ((n+1)! * n!)
19     // 0 equivalente: C_n = C(2n, n) / (n+1)
20
21     vector<long long> fact(2*n + 1);
22     fact[0] = 1;
23     for (int i = 1; i <= 2*n; i++) {
24         fact[i] = (fact[i-1] * i) % mod;
25     }
26
27     auto binpow = [&](long long a, long long b) {
28         long long res = 1;
29         while (b > 0) {
30             if (b & 1) res = (res * a) % mod;
31             a = (a * a) % mod;
32             b >>= 1;
33         }
34         return res;
35     };
36
37     auto inv = [&](long long a) {
38         return binpow(a, mod - 2); // Si mod es primo
39     };
40
41     long long num = fact[2*n];
42     long long den = (fact[n+1] * fact[n]) % mod;
43
44     return (num * inv(den)) % mod;
45 }

```

2 Sumatorias Fundamentales

2.1 Suma de Naturales

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Derivación: Sea $S = 1 + 2 + \dots + n$. Escribiendo al revés:

$$\begin{aligned}
 S &= 1 + 2 + 3 + \dots + n \\
 S &= n + (n-1) + (n-2) + \dots + 1 \\
 2S &= (n+1) + (n+1) + \dots + (n+1) = n(n+1)
 \end{aligned}$$

```

1 // Suma de 1 a n
2 long long sumN(long long n) {
3     return n * (n + 1) / 2;
4 }

```

```

5 // Suma de a hasta b
6 long long sumRange(long long a, long long b) {
7     return sumN(b) - sumN(a - 1);
8 }
9
10 // Inverso: dado S, encontrar n tal que 1+2+...+n = S
11 long long inverseSumN(long long S) {
12     // n(n+1)/2 = S => n^2 + n - 2S = 0
13     // n = (-1 + sqrt(1 + 8S)) / 2
14     long long n = (-1 + sqrt(1 + 8*S)) / 2;
15     return n;
16 }
17

```

2.2 Suma de Cuadrados

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Demostración: Usando $(k+1)^3 - k^3 = 3k^2 + 3k + 1$ y telescoping.

```

1 // Suma de cuadrados
2 long long sumSquares(long long n) {
3     return n * (n + 1) * (2 * n + 1) / 6;
4 }
5
6 // Con modulo
7 long long sumSquaresMod(long long n, long long mod) {
8     n %= mod;
9     long long res = (n * (n + 1) % mod) * (2 * n + 1) % mod;
10
11     // Dividir entre 6 (necesitamos inverso modular)
12     long long inv6 = binpow(6, mod - 2, mod);
13     return (res * inv6) % mod;
14 }

```

2.3 Suma de Cubos

$$\sum_{i=1}^n i^3 = \left[\frac{n(n+1)}{2} \right]^2$$

¡Nota que la suma de cubos es el cuadrado de la suma de naturales!

```

1 // Suma de cubos
2 long long sumCubes(long long n) {
3     long long s = n * (n + 1) / 2;
4     return s * s;
5 }

```

2.4 Suma de Potencias k-ésimas

Fórmulas de Faulhaber:

$$\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{i=1}^n i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

Polinomios de Bernoulli: Para cualquier k , existe un polinomio de grado $k+1$ que da la suma.

```

1 // Suma de potencias usando precalculo
2 vector<long long> powerSum(int n, int k, long long mod) {
3     // Calcular suma de i^k para i de 1 a n
4     // usando programacion dinamica
5
6     vector<vector<long long>> dp(n + 1, vector<long long>(k + 1));
7
8     for (int i = 1; i <= n; i++) {
9         long long pow_i = 1;
10        for (int j = 0; j <= k; j++) {
11            dp[i][j] = (dp[i-1][j] + pow_i) % mod;
12            pow_i = (pow_i * i) % mod;
13        }
14    }
15
16    return dp[n];
17 }

```

2.5 Suma de Productos Consecutivos

$$\sum_{i=1}^n i(i+1) = \frac{n(n+1)(n+2)}{3}$$

Generalización:

$$\sum_{i=1}^n i(i+1)(i+2) \cdots (i+k) = \frac{n(n+1)(n+2) \cdots (n+k+1)}{k+2}$$

```

1 // Suma de productos consecutivos
2 long long sumConsecutiveProducts(long long n, int k) {
3     long long product = 1;
4     for (int i = 0; i <= k + 1; i++) {
5         product *= (n + i);
6     }
7     return product / (k + 2);
8 }

```

3 Manipulación de Sumatorias

3.1 Técnicas de Telescoping

Una sumatoria telescópica se colapsa cuando términos consecutivos se cancelan.

Ejemplo:

$$\sum_{i=1}^n \frac{1}{i(i+1)} = \sum_{i=1}^n \left(\frac{1}{i} - \frac{1}{i+1} \right) = 1 - \frac{1}{n+1} = \frac{n}{n+1}$$

Patrón general:

$$\sum_{i=a}^b (f(i) - f(i+1)) = f(a) - f(b+1)$$

```

1 // Suma telescopica: 1/(i(i+1))
2 double telescopicSum(int n) {
3     return (double)n / (n + 1);
4 }
5
6 // Suma: 1/(i(i+k)) usando fracciones parciales
7 double telescopicSumK(int n, int k) {
8     double sum = 0;
9     for (int i = 1; i <= n; i++) {
10        sum += 1.0 / (i * (i + k));
11    }

```

```

12     return sum;
13 }
14
15 // Formula cerrada: (1/k) * (H_n+k - H_k)
16 // donde H_n es el n-esimo numero armonico

```

3.2 Cambio de Orden de Sumatorias

Doble sumatoria:

$$\sum_{i=1}^n \sum_{j=1}^i f(i, j) = \sum_{j=1}^n \sum_{i=j}^n f(i, j)$$

Ejemplo:

$$\sum_{i=1}^n i^2 = \sum_{i=1}^n \sum_{j=1}^i (2j-1) = \sum_{j=1}^n (2j-1)(n-j+1)$$

3.3 Suma de Suma

$$\sum_{i=1}^n \sum_{j=1}^i j = \sum_{i=1}^n \frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{6}$$

```

1 // Suma de suma: sum(sum(j for j in 1..i) for i in 1..n)
2 long long sumOfSum(long long n) {
3     return n * (n + 1) * (n + 2) / 6;
4 }
5
6 // Generalizacion: suma k veces anidada
7 long long nestedSum(long long n, int k) {
8     // C(n + k, k + 1)
9     long long result = 1;
10    for (int i = 0; i <= k; i++) {
11        result *= (n + k - i);
12        result /= (i + 1);
13    }
14    return result;
15 }

```

4 Fórmulas Combinatorias

4.1 Coeficientes Binomiales

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n}{n-k}$$

Identidades importantes:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad (\text{Identidad de Pascal})$$

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

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

$$\sum_{k=0}^n k \binom{n}{k} = n \cdot 2^{n-1}$$

$$\binom{n}{k} \binom{k}{r} = \binom{n}{r} \binom{n-r}{k-r}$$

```

1 // Coeficientes binomiales con DP (Triangulo de Pascal)
2 vector<vector<long long>> binomialCoeffs(int n, long long mod) {
3     vector<vector<long long>> C(n + 1, vector<long long>(n + 1));
4
5     for (int i = 0; i <= n; i++) {
6         C[i][0] = 1;
7         for (int j = 1; j <= i; j++) {
8             C[i][j] = (C[i-1][j-1] + C[i-1][j]) % mod;
9         }
10    }
11
12    return C;
13 }
14
15 // Binomial individual optimizado
16 long long binomial(int n, int k) {
17     if (k > n - k) k = n - k;
18
19     long long result = 1;
20     for (int i = 0; i < k; i++) {
21         result *= (n - i);
22         result /= (i + 1);
23     }
24
25     return result;
26 }
27
28 // Binomial con modulo usando factoriales precalculados
29 struct BinomialMod {
30     vector<long long> fact, inv_fact;
31     long long mod;
32
33     BinomialMod(int n, long long mod) : mod(mod) {
34         fact.resize(n + 1);
35         inv_fact.resize(n + 1);
36
37         fact[0] = 1;
38         for (int i = 1; i <= n; i++) {
39             fact[i] = (fact[i-1] * i) % mod;
40         }
41
42         inv_fact[n] = binpow(fact[n], mod - 2);
43         for (int i = n - 1; i >= 0; i--) {
44             inv_fact[i] = (inv_fact[i+1] * (i+1)) % mod;
45         }
46     }
47
48     long long binpow(long long a, long long b) {
49         long long res = 1;
50         while (b > 0) {
51             if (b & 1) res = (res * a) % mod;
52             a = (a * a) % mod;
53             b >>= 1;
54         }
55         return res;
56     }
57
58     long long C(int n, int k) {
59         if (k > n || k < 0) return 0;
60         return (fact[n] * inv_fact[k] % mod) * inv_fact[n-k] % mod;
61     }
62 };

```

4.2 Identidad de Vandermonde

$$\binom{m+n}{r} = \sum_{k=0}^r \binom{m}{k} \binom{n}{r-k}$$

Esta identidad cuenta el número de formas de elegir r elementos de dos conjuntos disjuntos.

4.3 Números de Stirling

Stirling de primera especie $s(n, k)$: número de permutaciones de n elementos con k ciclos.

Stirling de segunda especie $S(n, k)$: número de formas de partir n elementos en k subconjuntos no vacíos.

Recurrencia:

$$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1)$$

```
1 // Numeros de Stirling de segunda especie
2 vector<vector<long long>> stirling2nd(int n, long long mod) {
3     vector<vector<long long>> S(n + 1, vector<long long>(n + 1));
4
5     S[0][0] = 1;
6     for (int i = 1; i <= n; i++) {
7         for (int j = 1; j <= i; j++) {
8             S[i][j] = (j * S[i-1][j] + S[i-1][j-1]) % mod;
9         }
10    }
11
12    return S;
13 }
14
15 // Numero de Bell: B_n = sum(S(n,k) for k=0..n)
16 // Numero de particiones de conjunto con n elementos
17 long long bellNumber(int n, long long mod) {
18     auto S = stirling2nd(n, mod);
19     long long result = 0;
20     for (int k = 0; k <= n; k++) {
21         result = (result + S[n][k]) % mod;
22     }
23     return result;
24 }
```

5 Teoría de Números Básica

5.1 Máximo Común Divisor (GCD)

Algoritmo de Euclides:

$$\gcd(a, b) = \gcd(b, a \bmod b), \quad \gcd(a, 0) = a$$

Propiedades:

$$\gcd(a, b) = \gcd(|a|, |b|)$$

$$\gcd(a, b) = \gcd(a - b, b)$$

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

$$\gcd(a + kb, b) = \gcd(a, b)$$

Identidad de Bézout: Existen enteros x, y tales que:

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

```
1 // GCD basico
2 long long gcd(long long a, long long b) {
3     return b == 0 ? a : gcd(b, a % b);
4 }
5
6 // GCD extendido (encuentra x, y tales que ax + by = gcd(a,b))
```

```

7 long long extgcd(long long a, long long b, long long& x, long long& y) {
8     if (b == 0) {
9         x = 1;
10        y = 0;
11        return a;
12    }
13
14    long long x1, y1;
15    long long d = extgcd(b, a % b, x1, y1);
16    x = y1;
17    y = x1 - (a / b) * y1;
18    return d;
19 }
20
21 // LCM usando GCD
22 long long lcm(long long a, long long b) {
23     return a / gcd(a, b) * b; // Evita overflow
24 }
25
26 // GCD de un array
27 long long gcd_array(vector<long long>& arr) {
28     long long result = arr[0];
29     for (int i = 1; i < arr.size(); i++) {
30         result = gcd(result, arr[i]);
31         if (result == 1) break;
32     }
33     return result;
34 }
35
36 // LCM de un array
37 long long lcm_array(vector<long long>& arr) {
38     long long result = arr[0];
39     for (int i = 1; i < arr.size(); i++) {
40         result = lcm(result, arr[i]);
41     }
42     return result;
43 }

```

5.2 Función de Euler (Totient)

La función $\phi(n)$ cuenta cuántos números menores o iguales a n son coprimos con n .

Fórmula:

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

donde el producto es sobre todos los primos p que dividen a n .

Propiedades:

$$\phi(p^k) = p^{k-1}(p-1) \quad (\text{si } p \text{ es primo})$$

$$\phi(mn) = \phi(m)\phi(n) \quad (\text{si } \gcd(m, n) = 1)$$

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

$$\sum_{k=1}^n \gcd(k, n) = \sum_{d|n} d \cdot \phi(n/d)$$

```

1 // Phi de un numero
2 long long phi(long long n) {
3     long long result = n;
4
5     for (long long p = 2; p * p <= n; p++) {
6         if (n % p == 0) {
7             while (n % p == 0) n /= p;
8             result -= result / p;

```

```

9         }
10    }
11
12    if (n > 1) result -= result / n;
13
14    return result;
15 }
16
17 // Phi de 1 a n con criba
18 vector<long long> phiSieve(int n) {
19     vector<long long> phi(n + 1);
20     for (int i = 0; i <= n; i++) phi[i] = i;
21
22     for (int i = 2; i <= n; i++) {
23         if (phi[i] == i) { // i es primo
24             for (int j = i; j <= n; j += i) {
25                 phi[j] -= phi[j] / i;
26             }
27         }
28     }
29
30     return phi;
31 }
32
33 // Suma de phi(i) para i de 1 a n
34 long long sumPhi(long long n) {
35     // Formula: sum(phi(i)) = (n(n+1))/2 + sum recursiva
36     // Para valores grandes, usar formula cerrada o memoization
37     long long result = 0;
38     for (long long i = 1; i <= n; i++) {
39         result += phi(i);
40     }
41     return result;
42 }

```

5.3 Divisores

Número de divisores:

Si $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, entonces:

$$\tau(n) = (a_1 + 1)(a_2 + 1) \cdots (a_k + 1)$$

Suma de divisores:

$$\sigma(n) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{a_2+1} - 1}{p_2 - 1} \cdots \frac{p_k^{a_k+1} - 1}{p_k - 1}$$

Propiedades:

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

$$\sum_{d|n} \frac{1}{d} = \frac{\sigma(n)}{n}$$

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

```

1 // Encontrar todos los divisores
2 vector<long long> divisors(long long n) {
3     vector<long long> divs;
4
5     for (long long i = 1; i * i <= n; i++) {
6         if (n % i == 0) {
7             divs.push_back(i);
8             if (i != n / i) {

```

```

9         divs.push_back(n / i);
10     }
11 }
12 }
13
14 sort(divs.begin(), divs.end());
15 return divs;
16 }
17
18 // Numero de divisores
19 int countDivisors(long long n) {
20     int count = 0;
21     for (long long i = 1; i * i <= n; i++) {
22         if (n % i == 0) {
23             count += (i * i == n) ? 1 : 2;
24         }
25     }
26     return count;
27 }
28
29 // Suma de divisores
30 long long sumDivisors(long long n) {
31     long long sum = 0;
32     for (long long i = 1; i * i <= n; i++) {
33         if (n % i == 0) {
34             sum += i;
35             if (i != n / i) {
36                 sum += n / i;
37             }
38         }
39     }
40     return sum;
41 }
42
43 // Criba para contar divisores de 1 a n
44 vector<int> divisorsSieve(int n) {
45     vector<int> d(n + 1, 0);
46
47     for (int i = 1; i <= n; i++) {
48         for (int j = i; j <= n; j += i) {
49             d[j]++;
50         }
51     }
52
53     return d;
54 }

```

5.4 Exponenciación Modular

Teorema de Fermat: Si p es primo y $\gcd(a, p) = 1$:

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

Teorema de Euler: Si $\gcd(a, n) = 1$:

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

Pequeño teorema de Fermat para inverso:

$$a^{-1} \equiv a^{p-2} \pmod{p} \quad (\text{si } p \text{ es primo})$$

```

1 // Exponenciacion binaria modular
2 long long binpow(long long a, long long b, long long mod) {
3     long long res = 1;
4     a %= mod;
5

```

```

6     while (b > 0) {
7         if (b & 1) res = (res * a) % mod;
8         a = (a * a) % mod;
9         b >>= 1;
10    }
11
12    return res;
13 }
14
15 // Inverso modular (cuando mod es primo)
16 long long modinv(long long a, long long mod) {
17     return binpow(a, mod - 2, mod);
18 }
19
20 // Inverso modular usando GCD extendido (funciona siempre)
21 long long modinv_gcd(long long a, long long mod) {
22     long long x, y;
23     long long g = extgcd(a, mod, x, y);
24
25     if (g != 1) return -1; // No existe inverso
26
27     return (x % mod + mod) % mod;
28 }
29
30 // Division modular: (a / b) mod m
31 long long moddiv(long long a, long long b, long long mod) {
32     return (a * modinv(b, mod)) % mod;
33 }

```

5.5 Números Primos

Criba de Eratóstenes: Encuentra todos los primos hasta n en $O(n \log \log n)$.

Teorema de los números primos:

$$\pi(n) \approx \frac{n}{\ln n}$$

donde $\pi(n)$ es el número de primos menores o iguales a n .

```

1 // Criba de Eratostenes
2 vector<bool> sieve(int n) {
3     vector<bool> is_prime(n + 1, true);
4     is_prime[0] = is_prime[1] = false;
5
6     for (int i = 2; i * i <= n; i++) {
7         if (is_prime[i]) {
8             for (int j = i * i; j <= n; j += i) {
9                 is_prime[j] = false;
10            }
11        }
12    }
13
14    return is_prime;
15 }
16
17 // Obtener lista de primos
18 vector<int> getPrimes(int n) {
19     vector<bool> is_prime = sieve(n);
20     vector<int> primes;
21
22     for (int i = 2; i <= n; i++) {
23         if (is_prime[i]) primes.push_back(i);
24     }
25
26     return primes;
27 }
28
29 // Test de primalidad basico

```

```

30 bool isPrime(long long n) {
31     if (n < 2) return false;
32     if (n == 2) return true;
33     if (n % 2 == 0) return false;
34
35     for (long long i = 3; i * i <= n; i += 2) {
36         if (n % i == 0) return false;
37     }
38
39     return true;
40 }
41
42 // Factorizacion prima
43 vector<pair<long long, int>> primeFactors(long long n) {
44     vector<pair<long long, int>> factors;
45
46     for (long long p = 2; p * p <= n; p++) {
47         if (n % p == 0) {
48             int exp = 0;
49             while (n % p == 0) {
50                 n /= p;
51                 exp++;
52             }
53             factors.push_back({p, exp});
54         }
55     }
56
57     if (n > 1) factors.push_back({n, 1});
58
59     return factors;
60 }

```

6 Congruencias y Ecuaciones Diofánticas

6.1 Congruencia Lineal

Resolver: $ax \equiv b \pmod{m}$

Solución: Existe solución si y solo si $\gcd(a, m) | b$.

Si $g = \gcd(a, m)$, hay exactamente g soluciones módulo m .

```

1 // Resolver ax ≡ b (mod m)
2 vector<long long> solveLinearCongruence(long long a, long long b,
3                                         long long m) {
4     long long g = gcd(a, m);
5
6     if (b % g != 0) return {}; // No hay solucion
7
8     // Reducir a a'x ≡ b' (mod m')
9     a /= g;
10    b /= g;
11    m /= g;
12
13    long long x, y;
14    extgcd(a, m, x, y);
15
16    x = (x * b % m + m) % m;
17
18    // Las g soluciones son x, x + m, x + 2m, ..., x + (g-1)m
19    vector<long long> solutions;
20    long long original_m = m * g;
21
22    for (int i = 0; i < g; i++) {
23        solutions.push_back((x + i * m) % original_m);
24    }
25

```

```

26     return solutions;
27 }

```

6.2 Teorema Chino del Resto (CRT)

Sistema de congruencias:

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

$$\vdots$$

$$x \equiv a_k \pmod{m_k}$$

Si los m_i son coprimos dos a dos, existe una única solución módulo $M = m_1 m_2 \cdots m_k$.

$$x = \sum_{i=1}^k a_i M_i y_i \pmod{M}$$

donde $M_i = M/m_i$ y y_i es el inverso de M_i módulo m_i .

```

1  // Teorema Chino del Resto
2  long long CRT(vector<long long>& a, vector<long long>& m) {
3      int n = a.size();
4      long long M = 1;
5
6      for (int i = 0; i < n; i++) M *= m[i];
7
8      long long x = 0;
9      for (int i = 0; i < n; i++) {
10         long long Mi = M / m[i];
11         long long yi = modinv_gcd(Mi, m[i]);
12         x = (x + a[i] * Mi % M * yi % M) % M;
13     }
14
15     return (x + M) % M;
16 }
17
18 // CRT para dos ecuaciones
19 pair<long long, long long> CRT_two(long long a1, long long m1,
20                                   long long a2, long long m2) {
21     // x ≡ a1 (mod m1), x ≡ a2 (mod m2)
22     long long x, y;
23     long long g = extgcd(m1, m2, x, y);
24
25     if ((a2 - a1) % g != 0) return {-1, -1}; // No hay solución
26
27     long long lcm_m = m1 / g * m2;
28     long long solution = (a1 + m1 * ((a2 - a1) / g * x % (m2 / g))) % lcm_m;
29
30     if (solution < 0) solution += lcm_m;
31
32     return {solution, lcm_m};
33 }

```

6.3 Ecuación Diofántica Lineal

Resolver: $ax + by = c$

Solución: Existe solución entera si y solo si $\gcd(a, b) | c$.

Solución general:

Si (x_0, y_0) es una solución particular:

$$\begin{aligned} x &= x_0 + k \frac{b}{g} \\ y &= y_0 - k \frac{a}{g} \end{aligned} \quad \text{para todo } k \in \mathbb{Z}$$

donde $g = \gcd(a, b)$.

```

1 // Ecuacion diofantica: ax + by = c
2 bool solveDiophantine(long long a, long long b, long long c,
3     long long& x, long long& y) {
4     long long g = extgcd(abs(a), abs(b), x, y);
5
6     if (c % g != 0) return false;
7
8     x *= c / g;
9     y *= c / g;
10
11     if (a < 0) x = -x;
12     if (b < 0) y = -y;
13
14     return true;
15 }
16
17 // Encontrar solucion no negativa minima
18 bool solveDiophantineNonNeg(long long a, long long b, long long c,
19     long long& x, long long& y) {
20     if (!solveDiophantine(a, b, c, x, y)) return false;
21
22     long long g = gcd(a, b);
23     a /= g; b /= g;
24
25     // x = x0 + k*b, y = y0 - k*a
26     // Queremos x >= 0 y y >= 0
27
28     // Para x >= 0: k >= -x0/b
29     // Para y >= 0: k <= y0/a
30
31     long long k_min = (x >= 0) ? 0 : (-x + b - 1) / b;
32     long long k_max = (y >= 0) ? 0 : y / a;
33
34     if (k_min > k_max) return false;
35
36     x = x + k_min * b;
37     y = y - k_min * a;
38
39     return true;
40 }
41
42 // Contar soluciones no negativas
43 long long countDiophantineSolutions(long long a, long long b, long long c) {
44     long long x, y;
45     if (!solveDiophantine(a, b, c, x, y)) return 0;
46
47     long long g = gcd(a, b);
48     a /= g; b /= g;
49
50     // Necesitamos x >= 0 y y >= 0
51     long long k_min = (x >= 0) ? -(x / b) : (-x + b - 1) / b;
52     long long k_max = (y >= 0) ? y / a : -((-y + a - 1) / a);
53
54     if (k_min > k_max) return 0;
55
56     return k_max - k_min + 1;
57 }

```

7 Particiones de Números

7.1 Particiones de Enteros

$p(n)$ = número de formas de escribir n como suma de enteros positivos (sin orden).

Función generadora:

$$\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k}$$

Recurrencia de Euler:

$$p(n) = \sum_{k=1}^{\infty} (-1)^{k+1} \left[p\left(n - \frac{k(3k-1)}{2}\right) + p\left(n - \frac{k(3k+1)}{2}\right) \right]$$

```
1 // Numero de particiones usando programacion dinamica
2 vector<long long> partitions(int n, long long mod) {
3     vector<long long> p(n + 1, 0);
4     p[0] = 1;
5
6     for (int k = 1; k <= n; k++) {
7         for (int i = k; i <= n; i++) {
8             p[i] = (p[i] + p[i - k]) % mod;
9         }
10    }
11
12    return p;
13 }
14
15 // Particiones con exactamente k partes
16 vector<vector<long long>> partitionsK(int n, int k, long long mod) {
17     vector<vector<long long>> dp(n + 1, vector<long long>(k + 1, 0));
18     dp[0][0] = 1;
19
20     for (int i = 1; i <= n; i++) {
21         for (int j = 1; j <= min(i, k); j++) {
22             // Particion con minimo = 1
23             dp[i][j] = dp[i-1][j-1];
24             // Particiones donde cada parte >= 2
25             if (i >= j) {
26                 dp[i][j] = (dp[i][j] + dp[i-j][j]) % mod;
27             }
28         }
29     }
30
31     return dp;
32 }
33
34 // Particiones con partes distintas
35 long long partitionsDistinct(int n, long long mod) {
36     vector<long long> dp(n + 1, 0);
37     dp[0] = 1;
38
39     for (int k = 1; k <= n; k++) {
40         for (int i = n; i >= k; i--) {
41             dp[i] = (dp[i] + dp[i - k]) % mod;
42         }
43     }
44
45     return dp[n];
46 }
```

7.2 Particiones de Conjuntos (Números de Bell)

Ya vistos anteriormente con los números de Stirling de segunda especie.

Fórmula de Dobinski:

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}$$

7.3 Composiciones

Una composición es una partición ordenada. El número de composiciones de n en k partes es:

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

El número total de composiciones de n es 2^{n-1} .

```
1 // Numero de composiciones de n en k partes
2 long long compositions(int n, int k) {
3     if (k > n || k <= 0) return 0;
4     return binomial(n - 1, k - 1);
5 }
6
7 // Generar todas las composiciones de n en k partes
8 void generateCompositions(int n, int k, vector<int>& current,
9     vector<vector<int>>& result) {
10     if (k == 1) {
11         current.push_back(n);
12         result.push_back(current);
13         current.pop_back();
14         return;
15     }
16
17     for (int i = 1; i <= n - k + 1; i++) {
18         current.push_back(i);
19         generateCompositions(n - i, k - 1, current, result);
20         current.pop_back();
21     }
22 }
```

8 Manipulación de Dígitos

8.1 Extracción y Conversión de Dígitos

```
1 // Obtener digitos de un numero
2 vector<int> getDigits(long long n) {
3     if (n == 0) return {0};
4
5     vector<int> digits;
6     while (n > 0) {
7         digits.push_back(n % 10);
8         n /= 10;
9     }
10    reverse(digits.begin(), digits.end());
11    return digits;
12 }
13
14 // Construir numero desde digitos
15 long long fromDigits(const vector<int>& digits) {
16     long long num = 0;
17     for (int d : digits) {
18         num = num * 10 + d;
19     }
20     return num;
21 }
22
23 // Suma de digitos
24 int digitSum(long long n) {
```

```

25     int sum = 0;
26     while (n > 0) {
27         sum += n % 10;
28         n /= 10;
29     }
30     return sum;
31 }
32
33 // Producto de digitos
34 long long digitProduct(long long n) {
35     if (n == 0) return 0;
36
37     long long prod = 1;
38     while (n > 0) {
39         prod *= n % 10;
40         n /= 10;
41     }
42     return prod;
43 }
44
45 // Numero de digitos
46 int countDigits(long long n) {
47     if (n == 0) return 1;
48     return floor(log10(n)) + 1;
49 }
50
51 // Raiz digital (suma iterada de digitos hasta obtener un digito)
52 int digitalRoot(long long n) {
53     return n == 0 ? 0 : 1 + (n - 1) % 9;
54 }

```

8.2 Palíndromos Numéricos

```

1 // Verificar si es palindromo
2 bool isPalindrome(long long n) {
3     string s = to_string(n);
4     int left = 0, right = s.length() - 1;
5
6     while (left < right) {
7         if (s[left] != s[right]) return false;
8         left++;
9         right--;
10    }
11
12    return true;
13 }
14
15 // Generar palindromo de n digitos
16 long long generatePalindrome(int n, int k) {
17     // k-esimo palindromo de n digitos
18     string half = to_string(k);
19     while (half.length() < (n + 1) / 2) {
20         half = "0" + half;
21     }
22
23     string palindrome = half;
24     int start = (n % 2 == 0) ? half.length() - 1 : half.length() - 2;
25
26     for (int i = start; i >= 0; i--) {
27         palindrome += half[i];
28     }
29
30     return stoll(palindrome);
31 }
32

```

```

33 // Siguiente palindromo mayor que n
34 long long nextPalindrome(long long n) {
35     string s = to_string(n + 1);
36     int len = s.length();
37     int mid = len / 2;
38
39     // Copiar primera mitad a segunda mitad (reflejada)
40     for (int i = 0; i < mid; i++) {
41         s[len - 1 - i] = s[i];
42     }
43
44     long long candidate = stoll(s);
45
46     if (candidate > n) return candidate;
47
48     // Incrementar la parte media
49     bool carry = true;
50     for (int i = (len - 1) / 2; i >= 0 && carry; i--) {
51         if (s[i] == '9') {
52             s[i] = '0';
53         } else {
54             s[i]++;
55             carry = false;
56         }
57         s[len - 1 - i] = s[i];
58     }
59
60     if (carry) {
61         s = "1" + string(len - 1, '0') + "1";
62     }
63
64     return stoll(s);
65 }

```

8.3 Bases Numéricas

```

1 // Convertir de base 10 a base b
2 string toBase(long long n, int b) {
3     if (n == 0) return "0";
4
5     string digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
6     string result = "";
7
8     while (n > 0) {
9         result = digits[n % b] + result;
10        n /= b;
11    }
12
13    return result;
14 }
15
16 // Convertir de base b a base 10
17 long long fromBase(const string& s, int b) {
18     long long result = 0;
19     long long power = 1;
20
21     for (int i = s.length() - 1; i >= 0; i--) {
22         int digit;
23         if (s[i] >= '0' && s[i] <= '9') {
24             digit = s[i] - '0';
25         } else {
26             digit = s[i] - 'A' + 10;
27         }
28         result += digit * power;
29         power *= b;
30     }
31 }

```

```

30     }
31
32     return result;
33 }
34
35 // Suma de digitos en base b
36 int digitSumBase(long long n, int b) {
37     int sum = 0;
38     while (n > 0) {
39         sum += n % b;
40         n /= b;
41     }
42     return sum;
43 }
44
45 // Numero de digitos en base b
46 int countDigitsBase(long long n, int b) {
47     if (n == 0) return 1;
48     return floor(log(n) / log(b)) + 1;
49 }

```

9 Fórmulas de Recurrencia

9.1 Resolver Recurrencias Lineales

Forma general:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$$

Método de ecuación característica:

La ecuación característica es:

$$r^k = c_1 r^{k-1} + c_2 r^{k-2} + \dots + c_k$$

Ejemplo - Fibonacci:

$$F_n = F_{n-1} + F_{n-2} \implies r^2 = r + 1 \implies r = \frac{1 \pm \sqrt{5}}{2}$$

Solución:

$$F_n = A\phi^n + B\psi^n$$

donde A y B se determinan con condiciones iniciales.

```

1 // Recurrencia lineal con matriz (O(k^3 log n))
2 template<typename T>
3 struct LinearRecurrence {
4     vector<T> coeffs; // c_1, c_2, ..., c_k
5     vector<T> initial; // a_0, a_1, ..., a_{k-1}
6     T mod;
7
8     LinearRecurrence(vector<T> c, vector<T> init, T mod = 1e9+7)
9         : coeffs(c), initial(init), mod(mod) {}
10
11     // Multiplicar dos matrices k x k
12     vector<vector<T>> matmul(const vector<vector<T>>& A,
13                             const vector<vector<T>>& B) {
14         int k = A.size();
15         vector<vector<T>> C(k, vector<T>(k, 0));
16
17         for (int i = 0; i < k; i++) {
18             for (int j = 0; j < k; j++) {
19                 for (int p = 0; p < k; p++) {
20                     C[i][j] = (C[i][j] + A[i][p] * B[p][j]) % mod;
21                 }
22             }
23         }
24     }

```

```

25     return C;
26 }
27
28 // Exponenciacion de matriz
29 vector<vector<T>> matpow(vector<vector<T>> base, long long exp) {
30     int k = base.size();
31     vector<vector<T>> res(k, vector<T>(k, 0));
32
33     // Matriz identidad
34     for (int i = 0; i < k; i++) res[i][i] = 1;
35
36     while (exp > 0) {
37         if (exp & 1) res = matmul(res, base);
38         base = matmul(base, base);
39         exp >>= 1;
40     }
41
42     return res;
43 }
44
45 T nth(long long n) {
46     int k = coeffs.size();
47
48     if (n < k) return initial[n];
49
50     // Construir matriz de transicion
51     vector<vector<T>> trans(k, vector<T>(k, 0));
52
53     // Primera fila: coeficientes
54     for (int i = 0; i < k; i++) {
55         trans[0][i] = coeffs[i];
56     }
57
58     // Resto: matriz identidad desplazada
59     for (int i = 1; i < k; i++) {
60         trans[i][i-1] = 1;
61     }
62
63     // Calcular trans^(n-k+1)
64     auto result = matpow(trans, n - k + 1);
65
66     // Multiplicar con condiciones iniciales (invertidas)
67     T ans = 0;
68     for (int i = 0; i < k; i++) {
69         ans = (ans + result[0][i] * initial[k - 1 - i]) % mod;
70     }
71
72     return ans;
73 }
74 };

```

9.2 Recurrencias Famosas

Números de Lucas:

$$L_n = L_{n-1} + L_{n-2}, \quad L_0 = 2, \quad L_1 = 1$$

Relación con Fibonacci: $L_n = F_{n-1} + F_{n+1}$

Números de Tribonacci:

$$T_n = T_{n-1} + T_{n-2} + T_{n-3}, \quad T_0 = 0, \quad T_1 = 0, \quad T_2 = 1$$

Números de Pell:

$$P_n = 2P_{n-1} + P_{n-2}, \quad P_0 = 0, \quad P_1 = 1$$

```

1 // Numeros de Lucas
2 long long lucas(int n, long long mod) {

```

```

3     if (n == 0) return 2;
4     if (n == 1) return 1;
5
6     long long a = 2, b = 1;
7     for (int i = 2; i <= n; i++) {
8         long long c = (a + b) % mod;
9         a = b;
10        b = c;
11    }
12
13    return b;
14 }
15
16 // Numeros de Tribonacci
17 long long tribonacci(int n, long long mod) {
18     if (n == 0 || n == 1) return 0;
19     if (n == 2) return 1;
20
21     long long a = 0, b = 0, c = 1;
22
23     for (int i = 3; i <= n; i++) {
24         long long d = (a + b + c) % mod;
25         a = b;
26         b = c;
27         c = d;
28     }
29
30     return c;
31 }
32
33 // Numeros de Pell
34 long long pell(int n, long long mod) {
35     if (n == 0) return 0;
36     if (n == 1) return 1;
37
38     long long a = 0, b = 1;
39
40     for (int i = 2; i <= n; i++) {
41         long long c = (2 * b + a) % mod;
42         a = b;
43         b = c;
44     }
45
46     return b;
47 }

```

10 Probabilidad y Valor Esperado

10.1 Probabilidad Básica

Regla del producto:

$$P(A \cap B) = P(A) \cdot P(B|A)$$

Regla de la suma:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Probabilidad condicional:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Eventos independientes:

$$P(A \cap B) = P(A) \cdot P(B)$$

10.2 Valor Esperado

Definición:

$$E[X] = \sum_i x_i \cdot P(X = x_i)$$

Linealidad del valor esperado:

$$E[aX + bY] = aE[X] + bE[Y]$$

Esta propiedad es válida incluso si X e Y no son independientes.

Valor esperado del producto (variables independientes):

$$E[XY] = E[X] \cdot E[Y]$$

10.3 Problemas Clásicos

Problema del coleccionista de cupones:

¿Cuántas cajas hay que abrir en promedio para obtener todos los n cupones diferentes?

$$E = n \sum_{i=1}^n \frac{1}{i} = n \cdot H_n \approx n \ln n$$

Problema de las parejas:

Probabilidad de que al menos una pareja se siente junta en una permutación aleatoria:

$$P = 1 - \frac{D_n}{n!}$$

donde D_n es el número de desarreglos.

```
1 // Valor esperado discreto
2 double expectedValue(const vector<double>& values,
3                       const vector<double>& probabilities) {
4     double E = 0;
5     for (int i = 0; i < values.size(); i++) {
6         E += values[i] * probabilities[i];
7     }
8     return E;
9 }
10
11 // Coleccionista de cupones
12 double couponCollector(int n) {
13     double E = 0;
14     for (int i = 1; i <= n; i++) {
15         E += (double)n / i;
16     }
17     return E;
18 }
19
20 // Desarreglos (permutaciones sin puntos fijos)
21 long long derangements(int n, long long mod) {
22     if (n == 0) return 1;
23     if (n == 1) return 0;
24
25     long long a = 1, b = 0;
26
27     for (int i = 2; i <= n; i++) {
28         long long c = ((i - 1) * ((a + b) % mod)) % mod;
29         a = b;
30         b = c;
31     }
32
33     return b;
34 }
35
36 // Formula de desarreglos: D_n = n! * sum((-1)^k / k!)
37 long long derangementsFormula(int n, long long mod) {
```

```

38     vector<long long> fact(n + 1);
39     fact[0] = 1;
40     for (int i = 1; i <= n; i++) {
41         fact[i] = (fact[i-1] * i) % mod;
42     }
43
44     long long sum = 0;
45     long long sign = 1;
46
47     for (int k = 0; k <= n; k++) {
48         long long inv_fact = binpow(fact[k], mod - 2, mod);
49         sum = (sum + sign * inv_fact + mod) % mod;
50         sign = -sign;
51     }
52
53     return (fact[n] * sum) % mod;
54 }

```

11 Funciones Especiales

11.1 Función de Möbius

$$\mu(n) = \begin{cases} 1 & \text{si } n = 1 \\ 0 & \text{si } n \text{ tiene un factor cuadrado} \\ (-1)^k & \text{si } n \text{ es producto de } k \text{ primos distintos} \end{cases}$$

Propiedad importante:

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{si } n = 1 \\ 0 & \text{si } n > 1 \end{cases}$$

Fórmula de inversión de Möbius:

$$g(n) = \sum_{d|n} f(d) \iff f(n) = \sum_{d|n} \mu(d)g(n/d)$$

```

1 // Funcion de Mobius
2 vector<int> mobiusSieve(int n) {
3     vector<int> mu(n + 1, 1);
4     vector<bool> is_prime(n + 1, true);
5
6     for (int i = 2; i <= n; i++) {
7         if (is_prime[i]) {
8             for (int j = i; j <= n; j += i) {
9                 is_prime[j] = false;
10                mu[j] *= -1;
11            }
12
13            // Marcar multiplos de i^2
14            if ((long long)i * i <= n) {
15                for (long long j = (long long)i * i; j <= n; j += (long long)i * i) {
16                    mu[j] = 0;
17                }
18            }
19        }
20    }
21
22    mu[0] = 0;
23    return mu;
24 }
25
26 // Mobius de un numero
27 int mobius(long long n) {
28     int mu = 1;
29

```

```

30     for (long long p = 2; p * p <= n; p++) {
31         if (n % p == 0) {
32             n /= p;
33             mu *= -1;
34
35             if (n % p == 0) return 0; // Tiene factor cuadrado
36         }
37     }
38
39     if (n > 1) mu *= -1;
40
41     return mu;
42 }

```

11.2 Función Tau (Función Divisor)

Ya vista anteriormente. Cuenta el número de divisores.

Valor promedio:

$$\frac{1}{n} \sum_{k=1}^n \tau(k) \approx \ln n$$

11.3 Función Sigma (Suma de Divisores)

Números perfectos: n es perfecto si $\sigma(n) = 2n$.

Ejemplo: $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$

Teorema de Euclides-Euler: Un número par es perfecto si y solo si tiene la forma:

$$2^{p-1}(2^p - 1)$$

donde $2^p - 1$ es primo (primo de Mersenne).

```

1  // Verificar si es numero perfecto
2  bool isPerfect(long long n) {
3      return sumDivisors(n) == 2 * n;
4  }
5
6  // Generar numeros perfectos pares
7  vector<long long> generatePerfectNumbers(int count) {
8      vector<long long> perfects;
9
10     // Verificar primos de Mersenne 2^p - 1
11     for (int p = 2; perfects.size() < count && p < 64; p++) {
12         long long mersenne = (1LL << p) - 1;
13
14         if (isPrime(mersenne)) {
15             long long perfect = (1LL << (p - 1)) * mersenne;
16             perfects.push_back(perfect);
17         }
18     }
19
20     return perfects;
21 }
22
23 // Numeros abundantes: sigma(n) > 2n
24 bool isAbundant(long long n) {
25     return sumDivisors(n) > 2 * n;
26 }
27
28 // Numeros deficientes: sigma(n) < 2n
29 bool isDeficient(long long n) {
30     return sumDivisors(n) < 2 * n;
31 }

```

12 Identidades Algebraicas Útiles

12.1 Productos Notables

$$\begin{aligned}(a+b)^2 &= a^2 + 2ab + b^2 \\ (a-b)^2 &= a^2 - 2ab + b^2 \\ (a+b)(a-b) &= a^2 - b^2 \\ (a+b)^3 &= a^3 + 3a^2b + 3ab^2 + b^3 \\ (a-b)^3 &= a^3 - 3a^2b + 3ab^2 - b^3 \\ a^3 + b^3 &= (a+b)(a^2 - ab + b^2) \\ a^3 - b^3 &= (a-b)(a^2 + ab + b^2) \\ a^n - b^n &= (a-b)(a^{n-1} + a^{n-2}b + \dots + ab^{n-2} + b^{n-1})\end{aligned}$$

12.2 Binomio de Newton

$$(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$$

Casos especiales:

$$\begin{aligned}(1+x)^n &= \sum_{k=0}^n \binom{n}{k} x^k \\ (1-x)^n &= \sum_{k=0}^n \binom{n}{k} (-1)^k x^k \\ 2^n &= \sum_{k=0}^n \binom{n}{k}\end{aligned}$$

```
1 // Expandir (a + b)^n modulo m
2 vector<long long> binomialExpansion(long long a, long long b, int n,
3                                     long long mod) {
4     BinomialMod binom(n, mod);
5     vector<long long> result(n + 1);
6
7     long long pow_a = 1, pow_b = bpow(b, n, mod);
8
9     for (int k = 0; k <= n; k++) {
10         result[k] = (binom.C(n, k) * pow_a % mod) * pow_b % mod;
11
12         pow_a = (pow_a * a) % mod;
13         pow_b = (pow_b * modinv(b, mod)) % mod;
14     }
15
16     return result;
17 }
18
19 // Coeficiente de x^k en (a + bx)^n
20 long long binomialCoeff(long long a, long long b, int n, int k,
21                         long long mod) {
22     BinomialMod binom(n, mod);
23     long long pow_a = bpow(a, n - k, mod);
24     long long pow_b = bpow(b, k, mod);
25
26     return (binom.C(n, k) * pow_a % mod) * pow_b % mod;
27 }
```

12.3 Factorización de Polinomios

Suma de potencias:

$$a^n + b^n = (a + b)(a^{n-1} - a^{n-2}b + a^{n-3}b^2 - \dots + b^{n-1}) \quad (n \text{ impar})$$

Fórmula de Sophie Germain:

$$a^4 + 4b^4 = (a^2 + 2b^2 + 2ab)(a^2 + 2b^2 - 2ab)$$

13 Optimización con Fórmulas

13.1 Suma de Subconjuntos

Problema: Encontrar si existe un subconjunto de suma S .

Suma total de todos los subconjuntos: Si tenemos elementos a_1, a_2, \dots, a_n , la suma de todos los posibles subconjuntos es:

$$2^{n-1} \sum_{i=1}^n a_i$$

Cada elemento aparece en exactamente 2^{n-1} subconjuntos.

```
1 // Suma de todos los subconjuntos
2 long long sumAllSubsets(const vector<int>& arr, long long mod) {
3     long long sum = 0;
4     for (int x : arr) sum = (sum + x) % mod;
5
6     long long power = binpow(2, arr.size() - 1, mod);
7     return (sum * power) % mod;
8 }
9
10 // Contar subconjuntos con suma divisible por k
11 int countSubsetsDivisible(const vector<int>& arr, int k) {
12     vector<int> dp(k, 0);
13     dp[0] = 1;
14
15     for (int x : arr) {
16         vector<int> new_dp = dp;
17         for (int rem = 0; rem < k; rem++) {
18             new_dp[(rem + x % k + k) % k] += dp[rem];
19         }
20         dp = new_dp;
21     }
22
23     return dp[0] - 1; // Restar subconjunto vacío
24 }
```

13.2 XOR de Subconjuntos

XOR de todos los subconjuntos:

Si un elemento a_i aparece k veces en los subconjuntos, su contribución al XOR total es: - 0 si k es par - a_i si k es impar

Cada elemento aparece en 2^{n-1} subconjuntos, entonces:

$$\text{XOR total} = \begin{cases} 0 & \text{si } n > 1 \\ a_1 & \text{si } n = 1 \end{cases}$$

```
1 // XOR de todos los subconjuntos
2 int xorAllSubsets(const vector<int>& arr) {
3     if (arr.size() == 1) return arr[0];
4     return 0; // Si n > 1, cada elemento aparece par veces
5 }
6
7 // Suma de XOR de todos los subconjuntos bit por bit
```

```

8 long long sumXORSubsets(const vector<int>& arr, long long mod) {
9     int n = arr.size();
10    long long result = 0;
11    long long power = binpow(2, n - 1, mod);
12
13    for (int bit = 0; bit < 30; bit++) {
14        int count = 0;
15        for (int x : arr) {
16            if (x & (1 << bit)) count++;
17        }
18
19        // Este bit contribuye count * 2^(n-1) * 2^bit al total
20        long long contribution = (count * power % mod) * (1LL << bit) % mod;
21        result = (result + contribution) % mod;
22    }
23
24    return result;
25 }

```

13.3 Producto de Elementos

Producto de todos los subconjuntos no vacíos:

Si tenemos n elementos a_1, a_2, \dots, a_n , el producto de todos los subconjuntos es:

$$\prod_{i=1}^n a_i^{2^{n-1}}$$

```

1 // Producto de todos los subconjuntos (modular)
2 long long productAllSubsets(const vector<long long>& arr, long long mod) {
3     int n = arr.size();
4     long long result = 1;
5     long long exponent = binpow(2, n - 1, mod - 1); // Teorema de Fermat
6
7     for (long long x : arr) {
8         result = (result * binpow(x, exponent, mod)) % mod;
9     }
10
11    return result;
12 }

```

14 Desigualdades Importantes

14.1 Desigualdad Triangular

$$|a + b| \leq |a| + |b|$$

$$|a - b| \geq ||a| - |b||$$

14.2 Desigualdades de Medias

Para números positivos a_1, a_2, \dots, a_n :

Media Armónica \leq Media Geométrica \leq Media Aritmética \leq Media Cuadrática

$$\frac{n}{\sum \frac{1}{a_i}} \leq \sqrt[n]{\prod a_i} \leq \frac{\sum a_i}{n} \leq \sqrt{\frac{\sum a_i^2}{n}}$$

```

1 // Calcular diferentes medias
2 double harmonicMean(const vector<double>& arr) {
3     double sum = 0;
4     for (double x : arr) sum += 1.0 / x;
5     return arr.size() / sum;

```

```

6 }
7
8 double geometricMean(const vector<double>& arr) {
9     double product = 1;
10    for (double x : arr) product *= x;
11    return pow(product, 1.0 / arr.size());
12 }
13
14 double arithmeticMean(const vector<double>& arr) {
15     double sum = 0;
16     for (double x : arr) sum += x;
17     return sum / arr.size();
18 }
19
20 double quadraticMean(const vector<double>& arr) {
21     double sum = 0;
22     for (double x : arr) sum += x * x;
23     return sqrt(sum / arr.size());
24 }

```

14.3 Desigualdad de Cauchy-Schwarz

$$\left(\sum_{i=1}^n a_i b_i\right)^2 \leq \left(\sum_{i=1}^n a_i^2\right) \left(\sum_{i=1}^n b_i^2\right)$$

La igualdad se da cuando a_i y b_i son proporcionales.

```

1 // Verificar desigualdad de Cauchy-Schwarz
2 bool verifyCauchySchwarz(const vector<double>& a,
3                          const vector<double>& b) {
4     double sum_ab = 0, sum_a2 = 0, sum_b2 = 0;
5
6     for (int i = 0; i < a.size(); i++) {
7         sum_ab += a[i] * b[i];
8         sum_a2 += a[i] * a[i];
9         sum_b2 += b[i] * b[i];
10    }
11
12    return sum_ab * sum_ab <= sum_a2 * sum_b2 + 1e-9;
13 }

```

14.4 Desigualdad de Bernoulli

Para $x \geq -1$ y $n \geq 0$:

$$(1+x)^n \geq 1+nx$$

Con igualdad solo cuando $x = 0$ o $n = 1$.

15 Manipulación de Módulos

15.1 Propiedades Básicas

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

$$(a - b) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod m$$

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

$$a^b \bmod m = \text{usar exponenciación binaria}$$

```

1 // Operaciones modulares seguras
2 long long addMod(long long a, long long b, long long mod) {
3     return ((a % mod) + (b % mod)) % mod;
4 }

```

```

5
6 long long subMod(long long a, long long b, long long mod) {
7     return ((a % mod) - (b % mod) + mod) % mod;
8 }
9
10 long long mulMod(long long a, long long b, long long mod) {
11     return ((a % mod) * (b % mod)) % mod;
12 }
13
14 // Multiplicacion modular segura contra overflow
15 long long mulModSafe(long long a, long long b, long long mod) {
16     long long result = 0;
17     a %= mod;
18
19     while (b > 0) {
20         if (b & 1) result = (result + a) % mod;
21         a = (a * 2) % mod;
22         b >>= 1;
23     }
24
25     return result;
26 }
27
28 // Usando __int128 para evitar overflow
29 long long mulMod128(long long a, long long b, long long mod) {
30     return (__int128)a * b % mod;
31 }

```

15.2 Raíz Cuadrada Modular

Encontrar x tal que $x^2 \equiv a \pmod{p}$ (donde p es primo).

Símbolo de Legendre:

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p} = \begin{cases} 1 & \text{si } a \text{ es residuo cuadrático} \\ -1 & \text{si } a \text{ no es residuo cuadrático} \\ 0 & \text{si } a \equiv 0 \pmod{p} \end{cases}$$

```

1 // Simbolo de Legendre
2 int legendre(long long a, long long p) {
3     long long result = binpow(a, (p - 1) / 2, p);
4     return result == p - 1 ? -1 : result;
5 }
6
7 // Raiz cuadrada modular (Algoritmo de Tonelli-Shanks)
8 long long sqrtMod(long long a, long long p) {
9     a %= p;
10    if (a == 0) return 0;
11
12    if (p == 2) return a;
13
14    // Verificar si tiene solucion
15    if (legendre(a, p) != 1) return -1;
16
17    // Caso especial: p ≡ 3 (mod 4)
18    if (p % 4 == 3) {
19        return binpow(a, (p + 1) / 4, p);
20    }
21
22    // Tonelli-Shanks para caso general
23    // (Implementacion simplificada)
24    long long s = 0, q = p - 1;
25    while (q % 2 == 0) {
26        q /= 2;
27        s++;
28    }

```

```

29
30 // Encontrar un no-residuo z
31 long long z = 2;
32 while (legendre(z, p) != -1) z++;
33
34 long long m = s;
35 long long c = binpow(z, q, p);
36 long long t = binpow(a, q, p);
37 long long r = binpow(a, (q + 1) / 2, p);
38
39 while (t != 1) {
40     long long i = 1;
41     long long temp = (t * t) % p;
42
43     while (temp != 1 && i < m) {
44         temp = (temp * temp) % p;
45         i++;
46     }
47
48     long long b = binpow(c, 1LL << (m - i - 1), p);
49     m = i;
50     c = (b * b) % p;
51     t = (t * c) % p;
52     r = (r * b) % p;
53 }
54
55 return r;
56 }

```

16 Trucos y Técnicas Avanzadas

16.1 Suma de Rango en $O(1)$

Precalcular sumas prefijas para responder consultas de rango en $O(1)$.

```

1 // Suma de rango con prefix sum
2 struct RangeSum {
3     vector<long long> prefix;
4
5     RangeSum(const vector<int>& arr) {
6         int n = arr.size();
7         prefix.resize(n + 1, 0);
8
9         for (int i = 0; i < n; i++) {
10             prefix[i + 1] = prefix[i] + arr[i];
11         }
12     }
13
14     // Suma de [l, r] (0-indexed)
15     long long query(int l, int r) {
16         return prefix[r + 1] - prefix[l];
17     }
18 };
19
20 // Suma 2D
21 struct RangeSum2D {
22     vector<vector<long long>> prefix;
23
24     RangeSum2D(const vector<vector<int>>& matrix) {
25         int n = matrix.size();
26         int m = matrix[0].size();
27
28         prefix.assign(n + 1, vector<long long>(m + 1, 0));
29
30         for (int i = 0; i < n; i++) {

```

```

31         for (int j = 0; j < m; j++) {
32             prefix[i+1][j+1] = matrix[i][j]
33                 + prefix[i][j+1]
34                 + prefix[i+1][j]
35                 - prefix[i][j];
36         }
37     }
38 }
39
40 // Suma del rectangulo [(r1,c1), (r2,c2)]
41 long long query(int r1, int c1, int r2, int c2) {
42     return prefix[r2+1][c2+1]
43         - prefix[r1][c2+1]
44         - prefix[r2+1][c1]
45         + prefix[r1][c1];
46 }
47 };

```

16.2 Diferencia Finita

Técnica para actualizaciones de rango en $O(1)$.

```

1 // Diferencia finita para updates de rango
2 struct RangeUpdate {
3     vector<long long> diff;
4     int n;
5
6     RangeUpdate(int n) : n(n) {
7         diff.assign(n + 1, 0);
8     }
9
10    // Agregar val a [l, r]
11    void update(int l, int r, long long val) {
12        diff[l] += val;
13        diff[r + 1] -= val;
14    }
15
16    // Construir array final
17    vector<long long> build() {
18        vector<long long> arr(n);
19        long long curr = 0;
20
21        for (int i = 0; i < n; i++) {
22            curr += diff[i];
23            arr[i] = curr;
24        }
25
26        return arr;
27    }
28 };

```

16.3 Suma de Subarreglos

Contribución de cada elemento a todas las sumas de subarreglos.

$$\text{Contribución de } a_i = a_i \times (i + 1) \times (n - i)$$

```

1 // Suma de todos los subarreglos
2 long long sumAllSubarrays(const vector<int>& arr, long long mod) {
3     long long result = 0;
4     int n = arr.size();
5
6     for (int i = 0; i < n; i++) {
7         long long contribution = (long long)arr[i] * (i + 1) % mod
8                                 * (n - i) % mod;

```

```

9         result = (result + contribution) % mod;
10     }
11
12     return result;
13 }
14
15 // Producto de todos los subarreglos
16 long long productAllSubarrays(const vector<int>& arr, long long mod) {
17     long long result = 1;
18     int n = arr.size();
19
20     for (int i = 0; i < n; i++) {
21         long long exponent = (long long)(i + 1) * (n - i) % (mod - 1);
22         result = (result * binpow(arr[i], exponent, mod)) % mod;
23     }
24
25     return result;
26 }

```

17 Series y Límites Importantes

17.1 Series de Potencias

Serie geométrica:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad (|x| < 1)$$

Serie exponencial:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Serie de seno y coseno:

$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

$$\cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!}$$

Serie logarítmica:

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^k}{k} \quad (|x| < 1)$$

```

1 // Calcular e^x con serie de Taylor
2 double exp_taylor(double x, int terms = 20) {
3     double result = 1.0;
4     double term = 1.0;
5
6     for (int k = 1; k < terms; k++) {
7         term *= x / k;
8         result += term;
9     }
10
11     return result;
12 }
13
14 // Calcular sin(x) con serie de Taylor
15 double sin_taylor(double x, int terms = 20) {
16     double result = 0.0;
17     double term = x;
18
19     for (int k = 0; k < terms; k++) {
20         result += (k % 2 == 0 ? 1 : -1) * term;
21         term *= x * x / ((2*k + 2) * (2*k + 3));

```

```

22     }
23
24     return result;
25 }
26
27 // Calcular ln(1+x) con serie de Taylor
28 double ln_taylor(double x, int terms = 100) {
29     if (abs(x) >= 1) return NAN; // No converge
30
31     double result = 0.0;
32     double term = x;
33
34     for (int k = 1; k < terms; k++) {
35         result += (k % 2 == 1 ? 1 : -1) * term / k;
36         term *= x;
37     }
38
39     return result;
40 }

```

17.2 Límites Fundamentales

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = e$$

$$\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$$

$$\lim_{n \rightarrow \infty} \frac{a^n}{n!} = 0 \quad (\text{para cualquier } a)$$

$$\lim_{n \rightarrow \infty} \sqrt[n]{n} = 1$$

17.3 Aproximaciones de Stirling

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

$$\ln(n!) \approx n \ln(n) - n + \frac{1}{2} \ln(2\pi n)$$

```

1 // Aproximacion de Stirling para ln(n!)
2 double stirlingLogFactorial(int n) {
3     const double PI = acos(-1.0);
4     return n * log(n) - n + 0.5 * log(2 * PI * n);
5 }
6
7 // Aproximacion de Stirling para n!
8 double stirlingFactorial(int n) {
9     const double PI = acos(-1.0);
10    return sqrt(2 * PI * n) * pow(n / M_E, n);
11 }

```

18 Problemas Clásicos y Aplicaciones

18.1 Problema de Cambio de Monedas

¿De cuántas formas podemos dar cambio de n usando monedas de denominaciones dadas?

```

1 // Cambio de monedas - numero de formas
2 long long coinChange(int n, const vector<int>& coins, long long mod) {
3     vector<long long> dp(n + 1, 0);
4     dp[0] = 1;
5

```

```

6     for (int coin : coins) {
7         for (int i = coin; i <= n; i++) {
8             dp[i] = (dp[i] + dp[i - coin]) % mod;
9         }
10    }
11
12    return dp[n];
13 }
14
15 // Minimo numero de monedas para hacer n
16 int minCoins(int n, const vector<int>& coins) {
17     vector<int> dp(n + 1, INT_MAX);
18     dp[0] = 0;
19
20     for (int i = 1; i <= n; i++) {
21         for (int coin : coins) {
22             if (coin <= i && dp[i - coin] != INT_MAX) {
23                 dp[i] = min(dp[i], dp[i - coin] + 1);
24             }
25         }
26     }
27
28     return dp[n] == INT_MAX ? -1 : dp[n];
29 }

```

18.2 Subsecuencia Creciente Más Larga (LIS)

Teorema de Erdős-Szekeres:

En cualquier secuencia de $n^2 + 1$ números distintos, existe una subsecuencia creciente o decreciente de longitud al menos $n + 1$.

```

1 // LIS con DP - O(n^2)
2 int LIS_dp(const vector<int>& arr) {
3     int n = arr.size();
4     vector<int> dp(n, 1);
5
6     for (int i = 1; i < n; i++) {
7         for (int j = 0; j < i; j++) {
8             if (arr[j] < arr[i]) {
9                 dp[i] = max(dp[i], dp[j] + 1);
10            }
11        }
12    }
13
14    return *max_element(dp.begin(), dp.end());
15 }
16
17 // LIS con busqueda binaria - O(n log n)
18 int LIS_binary(const vector<int>& arr) {
19     vector<int> lis;
20
21     for (int x : arr) {
22         auto it = lower_bound(lis.begin(), lis.end(), x);
23         if (it == lis.end()) {
24             lis.push_back(x);
25         } else {
26             *it = x;
27         }
28     }
29
30    return lis.size();
31 }
32
33 // Numero de LIS
34 int countLIS(const vector<int>& arr, long long mod) {
35     int n = arr.size();

```

```

36     vector<int> length(n, 1);
37     vector<long long> count(n, 1);
38
39     int maxLen = 1;
40
41     for (int i = 1; i < n; i++) {
42         for (int j = 0; j < i; j++) {
43             if (arr[j] < arr[i]) {
44                 if (length[j] + 1 > length[i]) {
45                     length[i] = length[j] + 1;
46                     count[i] = count[j];
47                 } else if (length[j] + 1 == length[i]) {
48                     count[i] = (count[i] + count[j]) % mod;
49                 }
50             }
51         }
52         maxLen = max(maxLen, length[i]);
53     }
54
55     long long result = 0;
56     for (int i = 0; i < n; i++) {
57         if (length[i] == maxLen) {
58             result = (result + count[i]) % mod;
59         }
60     }
61
62     return result;
63 }

```

18.3 Números de Euler y Desarreglos

Ya vistos anteriormente, pero aquí hay más identidades.

Número de permutaciones con exactamente k puntos fijos:

$$\binom{n}{k} D_{n-k}$$

Probabilidad de que una permutación aleatoria sea un desarreglo:

$$\lim_{n \rightarrow \infty} \frac{D_n}{n!} = \frac{1}{e} \approx 0.368$$

18.4 Problema de Josefo

n personas en círculo, eliminando cada k -ésima persona. ¿Quién sobrevive?

Fórmula para $k = 2$:

$$J(n) = 2L + 1$$

donde $n = 2^m + L$ y $0 \leq L < 2^m$.

```

1 // Problema de Josefo - solucion recursiva
2 int josephus(int n, int k) {
3     if (n == 1) return 0;
4     return (josephus(n - 1, k) + k) % n;
5 }
6
7 // Problema de Josefo - solucion iterativa
8 int josephus_iter(int n, int k) {
9     int result = 0;
10    for (int i = 2; i <= n; i++) {
11        result = (result + k) % i;
12    }
13    return result;
14 }
15
16 // Caso especial k=2
17 int josephus_k2(int n) {

```

```

18     int powerOf2 = 1;
19     while (powerOf2 * 2 <= n) powerOf2 *= 2;
20
21     int L = n - powerOf2;
22     return 2 * L + 1;
23 }

```

19 Fórmulas de Conteo Avanzadas

19.1 Principio de Inclusión-Exclusión

$$|A_1 \cup A_2 \cup \dots \cup A_n| = \sum |A_i| - \sum |A_i \cap A_j| + \sum |A_i \cap A_j \cap A_k| - \dots$$

Aplicación: Contar permutaciones sin elementos prohibidos.

```

1  // Inclusion-Exclusion para contar elementos
2  long long inclusionExclusion(int n, const vector<int>& setSizes) {
3      int m = setSizes.size();
4      long long result = 0;
5
6      for (int mask = 1; mask < (1 << m); mask++) {
7          int count = __builtin_popcount(mask);
8          long long size = n;
9
10         for (int i = 0; i < m; i++) {
11             if (mask & (1 << i)) {
12                 size -= setSizes[i];
13             }
14         }
15
16         if (size < 0) continue;
17
18         if (count % 2 == 1) {
19             result += size;
20         } else {
21             result -= size;
22         }
23     }
24
25     return result;
26 }
27
28 // Desarreglos usando inclusion-exclusion
29 long long derangementsIE(int n, long long mod) {
30     vector<long long> fact(n + 1);
31     fact[0] = 1;
32     for (int i = 1; i <= n; i++) {
33         fact[i] = (fact[i-1] * i) % mod;
34     }
35
36     long long result = 0;
37     for (int k = 0; k <= n; k++) {
38         long long term = fact[n] * binpow(fact[k], mod - 2, mod) % mod;
39         if (k % 2 == 0) {
40             result = (result + term) % mod;
41         } else {
42             result = (result - term + mod) % mod;
43         }
44     }
45
46     return result;
47 }

```

19.2 Función de Partición de Euler

Número de particiones de n usando solo números impares = número usando solo números distintos.

Pentagonal de Euler:

$$\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k}$$

19.3 Números de Euler (Zigzag)

E_n cuenta permutaciones alternantes (zigzag) de longitud n .

Función generadora:

$$\sum_{n=0}^{\infty} E_n \frac{x^n}{n!} = \tan(x) + \sec(x)$$

```
1 // Numeros de Euler (permutaciones alternantes)
2 vector<long long> eulerNumbers(int n, long long mod) {
3     vector<vector<long long>> E(n + 1, vector<long long>(n + 1, 0));
4     E[0][0] = 1;
5
6     for (int i = 1; i <= n; i++) {
7         for (int j = 0; j <= i; j++) {
8             if (j > 0) {
9                 E[i][j] = (E[i][j] + (i - j) * E[i-1][j-1]) % mod;
10            }
11            if (j < i) {
12                E[i][j] = (E[i][j] + (j + 1) * E[i-1][j]) % mod;
13            }
14        }
15    }
16
17    vector<long long> result(n + 1);
18    for (int i = 0; i <= n; i++) {
19        for (int j = 0; j <= i; j++) {
20            result[i] = (result[i] + E[i][j]) % mod;
21        }
22    }
23
24    return result;
25 }
```

20 Optimizaciones y Técnicas Finales

20.1 Divide y Vencerás para Recurrencias

Master Theorem:

Para recurrencias de la forma $T(n) = aT(n/b) + f(n)$:

- Si $f(n) = O(n^c)$ donde $c < \log_b a$, entonces $T(n) = \Theta(n^{\log_b a})$
- Si $f(n) = \Theta(n^c \log^k n)$ donde $c = \log_b a$, entonces $T(n) = \Theta(n^c \log^{k+1} n)$
- Si $f(n) = \Omega(n^c)$ donde $c > \log_b a$, entonces $T(n) = \Theta(f(n))$

20.2 Optimización con Monotonía

Si una función es monótona, podemos usar búsqueda binaria.

```
1 // Búsqueda binaria en respuesta
2 long long binarySearchAnswer(long long left, long long right,
3                             function<bool(long long)> check) {
4     while (left < right) {
5         long long mid = left + (right - left) / 2;
6         if (check(mid)) {
7             right = mid;
8         }
9     }
10    return left;
11 }
```

```

8         } else {
9             left = mid + 1;
10        }
11    }
12    return left;
13}
14
15// Búsqueda ternaria para funciones unimodales
16double ternarySearch(double left, double right,
17                     function<double(double)> f) {
18    const double EPS = 1e-9;
19
20    while (right - left > EPS) {
21        double m1 = left + (right - left) / 3;
22        double m2 = right - (right - left) / 3;
23
24        if (f(m1) < f(m2)) {
25            left = m1;
26        } else {
27            right = m2;
28        }
29    }
30
31    return (left + right) / 2;
32}

```

20.3 Trucos con Bits

```

1 // Verificar si n es potencia de 2
2 bool isPowerOfTwo(int n) {
3     return n > 0 && (n & (n - 1)) == 0;
4 }
5
6 // Siguiente potencia de 2
7 int nextPowerOfTwo(int n) {
8     n--;
9     n |= n >> 1;
10    n |= n >> 2;
11    n |= n >> 4;
12    n |= n >> 8;
13    n |= n >> 16;
14    return n + 1;
15}
16
17 // Contar bits en rango [0, n]
18 long long countBitsInRange(long long n, int bit) {
19     long long cycle = 1LL << (bit + 1);
20     long long complete = (n + 1) / cycle;
21     long long remaining = (n + 1) % cycle;
22
23     long long count = complete * (cycle / 2);
24     count += max(0LL, remaining - (cycle / 2));
25
26     return count;
27 }
28
29 // Suma de XOR de pares
30 long long sumXORPairs(const vector<int>& arr) {
31     long long sum = 0;
32
33     for (int bit = 0; bit < 30; bit++) {
34         long long zeros = 0, ones = 0;
35
36         for (int x : arr) {
37             if (x & (1 << bit)) ones++;

```

```

38         else zeros++;
39     }
40
41     sum += zeros * ones * (1LL << bit);
42 }
43
44 return sum;
45 }

```

21 Tabla de Referencia Rápida

21.1 Sumatorias Comunes

Sumatoria	Fórmula Cerrada
$\sum_{i=1}^n i$	$\frac{n(n+1)}{2}$
$\sum_{i=1}^n i^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{i=1}^n i^3$	$\left[\frac{n(n+1)}{2}\right]^2$
$\sum_{i=1}^n (2i-1)$	n^2
$\sum_{i=1}^n i(i+1)$	$\frac{n(n+1)(n+2)}{3}$
$\sum_{i=1}^n \frac{1}{i(i+1)}$	$\frac{n}{n+1}$
$\sum_{i=0}^n 2^i$	$2^{n+1} - 1$
$\sum_{i=0}^n r^i$	$\frac{r^{n+1}-1}{r-1}$

21.2 Complejidades Comunes

Algoritmo	Complejidad
Suma de Gauss	$O(1)$
Fibonacci (matriz)	$O(\log n)$
GCD (Euclides)	$O(\log \min(a, b))$
Exponenciación binaria	$O(\log n)$
Criba de Eratóstenes	$O(n \log \log n)$
Factorización	$O(\sqrt{n})$
Phi (Euler)	$O(\sqrt{n})$
LIS (DP)	$O(n^2)$
LIS (binaria)	$O(n \log n)$
CRT	$O(k \log m)$

21.3 Constantes Útiles

Constante	Valor Aproximado
e	2.71828
π	3.14159
ϕ (proporción áurea)	1.61803
γ (Euler-Mascheroni)	0.57721
$\ln(2)$	0.69314
$\ln(10)$	2.30258
$\sqrt{2}$	1.41421
$\sqrt{3}$	1.73205

22 Conclusión y Recursos

Este cheatsheet cubre las fórmulas y técnicas aritméticas más importantes para programación competitiva. Recuerda:

- Siempre verifica los casos límite y el manejo de módulos
- Usa long long para evitar overflow
- Precalcula factoriales y coeficientes binomiales cuando sea posible

- Conoce las complejidades de los algoritmos
- Practica implementando estas fórmulas desde cero

Recursos adicionales:

- OEIS (Online Encyclopedia of Integer Sequences)
- cp-algorithms.com
- Codeforces EDU
- USACO Guide
- Competitive Programming 3 - Steven Halim

¡Fin del Cheatsheet de Aritmética y Fórmulas!
Domina estas fórmulas y serás imparable en competencias
