AVANSAT · OLIMPIADĂ · LECȚIA 9

Teoria numerelor pentru concurs: ciur, divizori, ridicare la putere, aritmetică modulară

Durată: 80 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile x-08, xi-06, co-01

CE VEI ȘTI DUPĂ LECȚIE

- Generezi numerele prime cu ciurul lui Eratostene în O(n log log n).
- Descompui numere în factori primi rapid, cu ciurul de cel mai mic factor prim.
- Calculezi a^b mod m în O(log b) cu ridicare la putere rapidă.
- Aplici aritmetica modulară: adunare, înmulțire, scădere, împărțire prin invers modular.
- Calculezi combinări modulo un număr prim.

1. Verificarea a unui singur număr nu ajunge

La clasa a X-a ai testat dacă n este prim în O(√n). Dacă trebuie să găsești **toate** numerele prime până la 10^7, verificarea fiecărui număr costă $10^7 \cdot \sqrt{10^7} \approx 3 \cdot 10^{10}$: mult prea mult. Ciurul face același lucru în $\approx 3 \cdot 10^7$ operații.

2. Ciurul lui Eratostene

Idee. Scrii numerele de la 2 la n. Cel mai mic număr netăiat (2) este prim: îi tai multiplii (4, 6, 8, ...). Următorul netăiat (3) este prim: îi tai multiplii. Continui. Numerele rămase netăiate sunt prime.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

verde = prim; portocaliu = compus (tăiat de un prim p ≤ √30 ≈ 5,4, începând cu p·p); 1 nu este prim

Ciurul

Pentru un prim `p`, multiplii mai mici decât `p·p` au un factor mai mic și au fost deja tăiați: deci tăierea începe de la `p·p`. Un număr compus `x ≤ n` are un divizor prim ≤ √n, deci e suficient să tratăm numai primii `p` cu `p·p ≤ n`.

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

bool compus[1000005];                    // compus[i] = true dacă i a fost tăiat

int main()
{
    int n;
    cin >> n;
    for (int p = 2; (long long)p * p <= n; p++)
        if (!compus[p])                  // p este prim
            for (int j = p * p; j <= n; j += p)
                compus[j] = true;        // tăiem multiplii lui p
    for (int i = 2; i <= n; i++)
        if (!compus[i]) cout << i << ' ';
    return 0;
}
```

DE CE AȘA?

Complexitate. Pentru fiecare prim `p` se fac n/p tăieri; suma n/2 + n/3 + n/5 + ... pentru primii p ≤ n este n · log log n. Practic, ciurul este aproape liniar: pentru n = 10^7 rulează în sub o secundă. **De ce `(long long) p * p`?** La n apropiat de 2^{31}, `p * p` poate depăși `int`.

DE ȘTIUT

Pentru memorie: `bool` ocupă un octet; pentru n = 10^8 folosește `bitset<100000001>` (1 bit pe număr, 12,5 MB). Cu ciurul **impar** (doar numerele impare) memoria se înjumătățește.

3. Descompunerea în factori primi, rapid

Cu împărțiri succesive, descompunerea lui x costă O(√x). Pentru **multe** numere, folosim un ciur care reține pentru fiecare număr **cel mai mic divizor prim** (`cmdp[i]`): apoi descompunerea lui x înseamnă să împărțim repetat la `cmdp[x]` .

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

int cmdp[1000005];                          // cel mai mic divizor prim al lui i

int main()
{
    const int N = 1000000;
    for (int i = 2; i <= N; i++)
        if (cmdp[i] == 0)                   // i este prim
            for (int j = i; j <= N; j += i)
                if (cmdp[j] == 0) cmdp[j] = i;

    int q;
    cin >> q;
    while (q--)
    {
        int x;
        cin >> x;
        while (x > 1)
        {
            int p = cmdp[x], e = 0;
            while (x % p == 0) { x /= p; e++; }
            cout << p << '^' << e << ' ';
        }
        cout << '\n';
    }
    return 0;
}
```

x	pași	rezultat
360	cmdp = 2: 360 → 180 → 90 → 45 (e = 3); cmdp(45) = 3: 45 → 15 → 5 (e = 2); cmdp(5) = 5 (e = 1)	$2^3 \cdot 3^2 \cdot 5^1$
97	cmdp(97) = 97 (prim)	97^1

SFAT

Câți divizori are x? Dacă x = p₁^e₁ · p₂^e₂ · ... atunci numărul divizorilor este $(e_1 + 1)(e_2 + 1)$... De exemplu 360 = $2^3 \cdot 3^2 \cdot 5$ are $4 \cdot 3 \cdot 2 = \mathbf{24}$ divizori. **Suma divizorilor** este produsul (1 + p + ... + p^e) pe fiecare factor.

4. cmmdc și cmmmc

Algoritmul lui Euclid (cunoscut din clasa a IX-a) calculează `cmmdc(a, b)` în O(log min(a, b)). În C++17 ai `__gcd(a, b)` și `std::gcd` (din `<numeric>`).

```cpp
long long cmmdc(long long a, long long b) { return b == 0 ? a : cmmdc(b, a % b); }
long long cmmmc(long long a, long long b) { return a / cmmdc(a, b) * b; }   // întâi împărțim: evităm depășirea
```

> **ATENȚIE — GREȘELI FRECVENTE**
>
> `a * b / cmmdc(a, b)` poate depăși `long long` când a și b sunt mari, chiar dacă rezultatul ar încăpea. Împarte **mai întâi** (`a / cmmdc * b`).

5. Ridicare la putere rapidă

Calculul lui a^b prin b înmulțiri este O(b). Dacă b = 10^9, e imposibil. Observație: `a^b = (a^(b/2))²` când b e par, iar `a^b = a · a^(b−1)` când b e impar. Fiecare pas înjumătățește b: **O(log b)**.

Varianta iterativă folosește biții lui b: pentru fiecare bit de 1, înmulțim rezultatul cu baza curentă; după fiecare pas, ridicăm baza la pătrat.

Pas	b (binar)	bit	baza	rezultat
început, 3^{13}	1101	–	3	1
1	1101	1	3	1 · 3 = 3
2	110	0	$3^2 = 9$	3
3	11	1	$9^2 = 81$	3 · 81 = 243
4	1	1	$81^2 = 6561$	243 · 6561 = 1 594 323

Într-adevăr $3^{13} = 3^1 \cdot 3^4 \cdot 3^8 = 3 \cdot 81 \cdot 6561 = 1\,594\,323$ (13 = 8 + 4 + 1).

6. Aritmetică modulară

În concurs, rezultatele mari se cer **modulo** un număr: de obicei `MOD = 1 000 000 007` ($10^9 + 7$, prim). Regulile (toate se aplică la fiecare pas, ca numerele să nu crească):

Operația	Cum o calculezi
`(a + b) mod m`	`(a % m + b % m) % m`
`(a − b) mod m`	`((a − b) % m + m) % m` (rezultat nenegativ)
`(a · b) mod m`	`(a % m) * (b % m) % m` (cu `long long`)
a^b `mod m`	ridicare la putere rapidă, cu `% m` la fiecare înmulțire
`(a / b) mod m`	`a · b⁻¹` , unde b^{-1} este inversul modular al lui b

> **ATENȚIE — GREȘELI FRECVENTE**
>
> **Nu există** `(a / b) % m = (a % m) / (b % m)` . Împărțirea modulară necesită **inversul modular**. Iar `%` pe numere negative dă rezultat negativ în C++: adaugă `m` și mai fă un `%` .

Inversul modular (Fermat)

Dacă `m` este **prim** și `b` nu este multiplu de `m` , atunci `b^(m−1) ≡ 1 (mod m)` (mica teoremă a lui Fermat). Deci `b · b^(m−2) ≡ 1` : inversul lui b este `b^(m−2) mod m` . Îl calculezi cu ridicarea la putere rapidă.

Combinări modulo p

C(n, k) = n! / (k! · (n − k)!). Precalculezi factorialele modulo p și inversele lor, apoi fiecare C(n, k) costă O(1).

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

const long long MOD = 1000000007;
const int N = 1000000;
long long fact[N + 1], inv[N + 1];

long long putere(long long a, long long b)           // a^b mod MOD, O(log b)
{
    long long rez = 1;
    a %= MOD;
    while (b > 0)
    {
        if (b & 1) rez = rez * a % MOD;              // bit de 1: înmulțim
        a = a * a % MOD;                             // ridicăm baza la pătrat
        b >>= 1;
    }
    return rez;
}

int main()
{
    fact[0] = 1;
    for (int i = 1; i <= N; i++) fact[i] = fact[i - 1] * i % MOD;
    inv[N] = putere(fact[N], MOD - 2);               // inversul lui N! (Fermat)
    for (int i = N; i >= 1; i--) inv[i - 1] = inv[i] * i % MOD;   // inv[i-1] = inv[i] * i

    int q;
    cin >> q;
    while (q--)
    {
        int n, k;
        cin >> n >> k;
        cout << fact[n] * inv[k] % MOD * inv[n - k] % MOD << '\n';
    }
    return 0;
}
```

DE CE AȘA?

De ce `inv[i − 1] = inv[i] · i`? Pentru că 1/(i−1)! = i / i!: dacă ai inversul lui `i!`, înmulțit cu `i` obții inversul lui `(i−1)!`. Astfel, calculezi **un singur** invers cu `putere` (O(log MOD)) și restul în O(1) fiecare, față de N puteri (O(N log MOD)). **De ce `% MOD` după fiecare înmulțire?** Două numere sub $10^9 + 7$ au produsul sub $\approx 10^{18} < 9{,}2 \cdot 10^{18}$: încape în `long long`, dar un al treilea factor nu mai încape: de aceea înmulțirile se fac pe rând.

7. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
ciur cu `j` pornind de la `2 * p`	corect, dar mai lent	de la `p * p`
`p * p` în `int`	depășire pentru n mare	`(long long) p * p`
`a * b % MOD` cu `a`, `b` în `int`	depășire înainte de `%`	`long long`
`a * b * c % MOD`	depășire	`a * b % MOD * c % MOD`
diferență modulară negativă	rezultat negativ	`(x % MOD + MOD) % MOD`
împărțire modulară cu `/`	rezultat greșit	multiplici cu inversul
Fermat cu `m` neprim	inversul nu e corect	folosește Euclid extins
factorial până la `n` și `n ≥ MOD`	`n!` ≡ 0	pentru n < MOD este în regulă
`pow` din `<cmath>` cu întregi mari	pierdere de precizie (double)	`putere` proprie

8. La olimpiadă

Multe probleme cer „afișați restul împărțirii la $10^9 + 7$": atunci, aritmetica modulară este obligatorie. Ciurul apare ca precalculare (prime, funcția φ, numărul de divizori). Combinările modulo apar în probleme de numărare: drumuri într-o grilă, aranjamente, DP cu numărare. Pentru **numere foarte mari** (10^{18}), testul Miller–Rabin înlocuiește ciurul.

9. Exersează pe hârtie

1. Ce numere taie ciurul pentru n = 50? Care sunt singurii primi `p` pentru care se taie ceva (cei cu p·p ≤ 50)?

2. Scrie lista primelor până la 50.

3. Descompune 360, 1001 și 4096 în factori primi.

4. Câți divizori are 360? Dar 1001?

5. Calculează 2^{11} cu algoritmul rapid (binar 1011), cu tabel ca în lecție.

6. Calculează 7^{10} mod 13 (folosește pătrate repetate modulo 13).

7. Calculează inversul lui 3 modulo 7 (găsește x cu 3x ≡ 1 mod 7).

8. Calculează (5 − 9) mod 7, fără rezultat negativ.

9. Calculează C(6, 2) mod 7 cu formula factorialelor și inversul modular (sau direct).

10. Câte operații face ciurul pentru n = 10^7 (aproximativ)? Dar testul prin √n pentru fiecare număr?

10. Mini-grile

1. Ciurul lui Eratostene are complexitatea: a) $O(n^2)$ b) $O(n \log \log n)$ c) $O(n\sqrt{n})$ d) $O(\log n)$

2. Ridicarea la putere rapidă calculează a^b în: a) O(b) b) O(log b) c) O(√b) d) O(1)

3. Inversul modular al lui b modulo `p` prim este: a) `b^p` b) `b^(p−2) mod p` c) `p − b` d) `1 / b`

4. Pentru `(a − b) mod m` fără rezultat negativ: a) `(a − b) % m` b) `((a − b) % m + m) % m` c) `(a % m) − (b % m)` d) `abs(a − b) % m`

11. Mai departe (pentru cine vrea mai mult)

- **Funcția φ a lui Euler** (numărul de numere mai mici și prime cu n) cu ciurul.
- **Euclid extins:** inversul modular pentru `m` oarecare (nu neapărat prim).
- **Teorema chineză a resturilor.**
- **Ciur segmentat:** primele dintr-un interval `[L, R]` cu R ≈ 10^{12}.
- **Miller–Rabin:** test de primalitate pentru numere până la 10^{18}.
- **Principiul includerii și excluderii** pentru probleme de numărare.

Rezumat

- **Ciur:** taie multiplii fiecărui prim p începând cu p·p; O(n log log n).
- **cmdp:** descompunere în factori primi în O(log x) după precalculare.
- **Putere rapidă:** `a^b` în O(log b), cu `% MOD` la fiecare înmulțire.
- **Modulo:** diferență `+ MOD`, produs în `long long`, împărțire prin invers.
- **Invers modular (MOD prim):** `b^(MOD−2)`; combinări cu factoriale și inverse, O(1) după precalculare.

Probleme de lucru (concurs)

Probleme de pe infoarena, CSES, Kilonova sau pbinfo, în ordinea dificultății. Trimite soluțiile pe platformă și bifează ce ai terminat.

- [ ] `infoarena` **ciur** — ușoară
  numărul numerelor prime până la n: ciurul din lecție
  https://www.infoarena.ro/problema/ciur
- [ ] `CSES` **exponentiation** — ușoară
  a^b mod 10^9+7 pentru multe perechi (CSES 1095, Exponentiation)
  https://cses.fi/problemset/task/1095
- [ ] `CSES` **Counting Divisors** — medie
  numărul de divizori cu ciurul sau prin √n
  https://cses.fi/problemset/task/1713
- [ ] `CSES` **Binomial Coefficients** — medie
  combinări modulo 10^9+7 cu factoriale și invers
  https://cses.fi/problemset/task/1079

Temă (concurs)

Mai grele: încearcă-le după ce le-ai terminat pe cele de mai sus.

- [ ] CSES **Prime Multiples** — dificilă
  principiul includerii și excluderii
  https://cses.fi/problemset/task/2185

- [ ] CSES **Exponentiation II** — dificilă
  a^(b^c) mod p: teorema lui Fermat pentru exponent
  https://cses.fi/problemset/task/1712

Citește și exersează în plus

- CP-Algorithms — Sieve of Eratosthenes — ciurul și variantele lui (engleză) https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html
- CP-Algorithms — Binary Exponentiation — ridicare la putere rapidă (engleză) https://cp-algorithms.com/algebra/binary-exp.html
- CP-Algorithms — Modular Inverse — invers modular prin Fermat și prin Euclid extins (engleză) https://cp-algorithms.com/algebra/module-inverse.html

Răspunsuri

Exersează pe hârtie

1. Primii cu `p·p ≤ 50`: 2, 3, 5, 7. Se taie: multiplii lui 2 de la 4; ai lui 3 de la 9; ai lui 5 de la 25; ai lui 7 de la 49.

2. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 (**15** numere).

3. $360 = 2^3 \cdot 3^2 \cdot 5$; $1001 = 7 \cdot 11 \cdot 13$; $4096 = 2^{12}$.

4. 360: $4 \cdot 3 \cdot 2 =$ **24**; 1001 = $7 \cdot 11 \cdot 13$: $2 \cdot 2 \cdot 2 =$ **8**.

5. $11 = 1011_2$. Pas 1: bit 1, rezultat 2, baza 4. Pas 2: bit 1, rezultat $2 \cdot 4 = 8$, baza 16. Pas 3: bit 0, rezultat 8, baza 256. Pas 4: bit 1, rezultat $8 \cdot 256 =$ **2048**.

6. $7^2 = 49 \equiv 10$; $7^4 \equiv 10^2 = 100 \equiv 9$; $7^8 \equiv 81 \equiv 3$. $10 = 8 + 2$: $7^{10} \equiv 3 \cdot 10 = 30 \equiv$ **4** (mod 13).

7. $3 \cdot 5 = 15 \equiv 1 \pmod 7$: inversul este **5**.

8. $5 - 9 = -4$; $(-4 \,\%\, 7 + 7) \,\%\, 7 = (-4 + 7) \,\%\, 7 =$ **3**.

9. C(6, 2) = 15, iar 15 mod 7 = **1**. (Cu formula: 6! / (2! · 4!) = 720 / 48 = 15.)

10. Ciur: $\approx 10^7 \cdot \ln \ln 10^7 \approx 3 \cdot 10^7$ operații. Testul prin √n: $10^7 \cdot \sqrt{(10^7)} \approx 3 \cdot 10^{10}$.

Mini-grile

1. b) O(n log log n). **2.** b) O(log b). **3.** b) `b^(p−2) mod p`. **4.** b) `((a − b) % m + m) % m`.