INFORMATICĂ · CLASA A IX-A · ALGORITMI ELEMENTARI · LECȚIA 11

Descompunerea în factori primi și aplicațiile ei

Durată: 60 min Nivel: toate profilurile Înainte: lecțiile ix-10

CE VEI ȘTI DUPĂ LECȚIE

- Descompui un număr în factori primi, eficient (până la √n).
- Afișezi factorii cu exponenții lor.
- Calculezi numărul și suma divizorilor din descompunere.
- Rezolvi probleme cu zerouri la final (factorii 2 și 5) și cu cel mai mare factor prim.

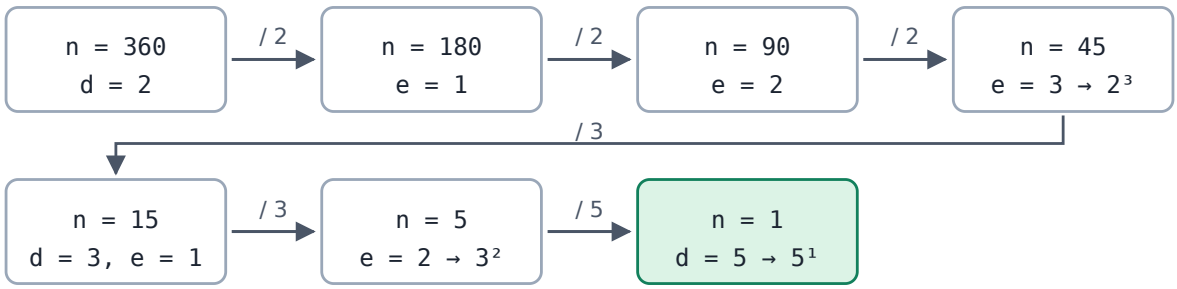
1. Ideea: orice număr e un produs de „cărămizi”

Orice număr natural n ≥ 2 se scrie **în mod unic** ca produs de puteri de numere prime:

$360 = 2^3 \cdot 3^2 \cdot 5^1$

Numerele prime sunt „atomii” numerelor, iar descompunerea e „formula chimică”. Din ea citești imediat: numărul de divizori, cmmdc, cmmmc, zerourile de la final. De aceea merită un algoritm separat.

Algoritmul: încercăm divizorii `d = 2, 3, 4, …`. Cât timp `d` îl divide pe `n`, îl împărțim și numărăm de câte ori.



la fiecare factor: împărțim cât se poate și numărăm exponentul

Descompunerea lui 360

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

int main()
{
    int n, d = 2;
    cin >> n;
    while (n > 1)
    {
        int e = 0;
        while (n % d == 0) { n /= d; e++; }
        if (e > 0) cout << d << ' ' << e << '\n';
        d++;
    }
    return 0;
}
```

Urmărire pentru `n = 360`:

d	n înainte	împărțiri reușite	e	n după	se afișează
2	360	360→180→90→45	3	45	`2 3`
3	45	45→15→5	2	5	`3 2`
4	5	niciuna	0	5	—
5	5	5→1	1	1	`5 1`

Intrare / ieșire:

Intrare	Ieșire (factor exponent)
`360`	`2 3`, `3 2`, `5 1`
`12`	`2 2`, `3 1`
`97`	`97 1` (număr prim)

DE CE AȘA?

De ce `d` este mereu prim când îl divide pe `n`? Toți factorii mai mici au fost deja eliminați complet. Dacă 4 ar divide ce a rămas din n, atunci 2 l-ar fi divizat mai devreme, dar 2 a fost scos până nu a mai putut. La fel pentru 6 (ar fi implicat 2 și 3) și 9. Deci nu trebuie să testăm dacă `d` e prim: se alege singur.

DE ȘTIUT

Pentru `n = 97` algoritmul de mai sus încearcă `d` până la 97: puțin, dar pentru un număr prim mare (de exemplu 999 999 937) ar face aproape un miliard de pași. Vezi secțiunea următoare.

2. Varianta eficientă (până la √n)

Observație: după ce am încercat toți `d` cu `d · d ≤ n`, ce a rămas din n (dacă e > 1) este **un singur factor prim**, deoarece două numere prime ≥ d ar avea produsul > n.

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

int main()
{
    long long n, d = 2;
    cin >> n;
    while (d * d <= n)
    {
        int e = 0;
        while (n % d == 0) { n /= d; e++; }
        if (e > 0) cout << d << ' ' << e << '\n';
        d++;
    }
    if (n > 1) cout << n << ' ' << 1 << '\n';   // ultimul factor prim, cu exponentul 1
    return 0;
}
```

n	ce face bucla	rest la final	se afișează
360	d = 2: 3 împărțiri; d = 3: 2 împărțiri; n = 5; `4·4 > 5` stop	5	`2 3`, `3 2`, apoi `5 1`
49	d = 2…6 nu; d = 7: 49→7→1, n = 1	1	`7 2`

DE CE AȘA?

De ce testăm `n > 1` după buclă? Bucla se oprește când `d · d > n`. Dacă a rămas un `n > 1`, nu are niciun divizor ≤ √n, deci e prim. Pentru 360, după ce scoatem 2 și 3, rămâne 5 care e prim, dar buclă nu l-a mai testat pentru că `4 · 4 > 5`. Așa apare ultimul factor.

LA BAC

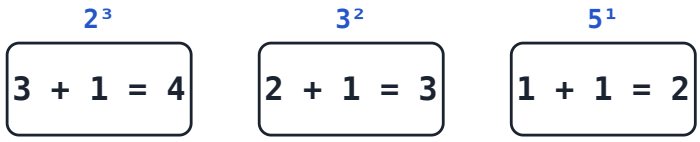
„Cel mai mare factor prim" al lui n este fie ultimul `d` găsit, fie restul `n > 1` de la final. Este o întrebare clasică de BAC.

3. Aplicații

Numărul de divizori

Dacă $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$, atunci numărul de divizori este

$(e_1 + 1) \cdot (e_2 + 1) \cdot \ldots \cdot (e_k + 1)$



ł · 3 · 2 = 24 divizori: la fiecare factor alegi exponentul 0, 1, ..., e

Formula numărului de divizori

Exemplu: 360 = $2^3 \cdot 3^2 \cdot 5$ → (3+1)(2+1)(1+1) = **24** divizori.

DE CE AȘA?

De ce se adună 1 la fiecare exponent? Un divizor ia din fiecare factor prim un exponent de la 0 până la e inclusiv: sunt e + 1 alegeri, independente între factori, deci se înmulțesc. De exemplu, divizorii lui 12 = $2^2 \cdot 3$ sunt 2^a · 3^b cu a ∈ {0, 1, 2}, b ∈ {0, 1}: 3 · 2 = 6 divizori.

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

int main()
{
    long long n, d = 2;
    int nrDiv = 1;
    cin >> n;
    while (d * d <= n)
    {
        int e = 0;
        while (n % d == 0) { n /= d; e++; }
        nrDiv *= (e + 1);
        d++;
    }
    if (n > 1) nrDiv *= 2;
    cout << nrDiv;
    return 0;
}
```

Zerourile de la sfârșitul unui produs

Un zero la final apare dintr-o pereche `2 · 5`. Numărul de zerouri = `min(nr. factori 2, nr. factori 5)`. Pentru `n!` factorii 5 sunt mai puțini decât factorii 2, deci:

```cpp
int z = 0;
for (long long p = 5; p <= n; p *= 5)
    z += n / p;          // numere divizibile cu 5, 25, 125, ...
```

Urmărire pentru `n = 100`: `100/5 = 20`, `100/25 = 4`, `125 > 100` stop → **24** de zerouri.

DE CE AȘA?

De ce se adună `n/5`, `n/25`, `n/125`? Fiecare multiplu de 5 aduce un factor 5; multiplii de 25 aduc **încă unul** (au 5^2), cei de 125 încă unul, și așa mai departe. Adunând, fiecare număr e numărat de câte ori îl conține pe 5.

cmmdc și cmmmc prin factori

- cmmdc: fiecare factor comun la exponentul **minim**;
- cmmmc: fiecare factor la exponentul **maxim**.

Exemplu: $360 = 2^3 \cdot 3^2 \cdot 5$ și $324 = 2^2 \cdot 3^4$: cmmdc $= 2^2 \cdot 3^2 = 36$; cmmmc $= 2^3 \cdot 3^4 \cdot 5 = 3240$.

(În practică cmmdc se calculează cu Euclid, dar aceasta e explicația din spate.)

4. Greșeli frecvente

Greșeala	Efect	Corect
uiți `if (n > 1)` după buclă	pierzi ultimul factor prim	tratează restul
`int` pentru `d * d` la n mare	depășire	`long long`
`d` crește și când l-ai putut împărți din nou	factori sărite	`while (n % d == 0)`, apoi `d++`
afișezi `d` fără să verifici `e > 0`	afișezi și factori cu exponent 0	`if (e > 0)`
`nrDiv *= e` în loc de `(e + 1)`	număr de divizori greșit	`(e + 1)`

5. La BAC

> **LA BAC**
>
> Descompunerea în factori primi apare ca **idee** în grile (numărul divizorilor, zerourile lui n!, „care este cel mai mare factor prim") și ca **algoritm** la Subiectul III (probleme de tip „afișați factorii primi ai lui n", „numărul divizorilor"). Pentru `n` până la 10^9 trebuie obligatoriu varianta cu `d * d <= n`.

6. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Descompune pe hârtie: 84, 1001, 1024, 97.

2. Câți divizori are 1 000 000? ($10^6 = 2^6 \cdot 5^6$)

3. Câte zerouri are la final 100!? Dar 30!?

4. Ce se afișează pentru `n = 12` în algoritmul din secțiunea 1?

5. Care este cel mai mare factor prim al lui $2^5 \cdot 3 \cdot 7$?

6. Calculează cmmdc și cmmmc pentru $360 = 2^3 \cdot 3^2 \cdot 5$ și $324 = 2^2 \cdot 3^4$, scriind descompunerile.

7. Pentru ce valori ale lui `n` între 1 și 30 are `n` exact 4 divizori? (Indiciu: p^3 sau $p \cdot q$.)

8. Scrie un program care afișează **suma** factorilor primi distincți ai lui n.

7. Mai departe (pentru cine vrea mai mult)

Ciurul cu factori. Dacă ai de descompus multe numere (de exemplu 10^6 numere până la 10^7), nu repeți algoritmul de fiecare dată. Construiești un ciur în care reții pentru fiecare număr **cel mai mic factor prim** (`cmf[i]`); apoi descompui orice număr în `O(log n)` : `n = n / cmf[n]` până la 1. Este o tehnică standard de concurs.

Funcția lui Euler φ(n). Numără câte numere între 1 și n sunt prime cu n. Din descompunere: $\varphi(n) = n \cdot \prod (1 - 1/p)$ pentru fiecare factor prim distinct p. Pentru 360: $360 \cdot (1/2) \cdot (2/3) \cdot (4/5) = 96$. Este folosită direct în criptografia RSA.

Aritmetica fundamentală. Unicitatea descompunerii pare evidentă, dar nu e valabilă în orice sistem de numere (există mulțimi de numere în care o descompunere nu e unică). În matematica serioasă, studiul acestor sisteme a dus la teoria numerelor moderne.

Factorizare și securitate. Pentru un număr cu sute de cifre, orice algoritm cunoscut durează mai mult decât vârsta universului. Aceasta este exact siguranța cumpărăturilor online.

Rezumat

- Împarte repetat la `d = 2, 3, …` cât timp se poate; numără exponentul.
- Oprește-te la `d · d ≤ n` ; restul `n > 1` e un factor prim.
- Număr divizori = $\prod (e_i + 1)$.
- Zerouri la final de n! = suma `n/5 + n/25 + n/125 + …` .
- cmmdc: exponenți minimi; cmmmc: exponenți maximi.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] #62 **factorizare** — ușoară
  Factor și exponent pentru fiecare factor prim.
  https://www.pbinfo.ro/probleme/62/factorizare

- [ ] #1319 **descompunere-factori** — ușoară
  https://www.pbinfo.ro/probleme/1319/descompunere-factori

- [ ] #63 **factorizare1** — ușoară
  Factorul cu exponentul maxim.
  https://www.pbinfo.ro/probleme/63/factorizare1

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #435 **factoriprimi** — ușoară
  Două numere au exact aceiași factori primi?
  https://www.pbinfo.ro/probleme/435/factoriprimi

- [ ] #439 **factzero** — ușoară
  Zerourile de la finalul lui n!: câți factori de 5?
  https://www.pbinfo.ro/probleme/439/factzero
- [ ] #443 **divizoriprimi** — ușoară
  https://www.pbinfo.ro/probleme/443/divizoriprimi
- [ ] #2821 **factoriprimi1** — ușoară
  https://www.pbinfo.ro/probleme/2821/factoriprimi1
- [ ] #1568 **mediediv** — ușoară
  https://www.pbinfo.ro/probleme/1568/mediediv

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #2306 **numere22** — ușoară
  https://www.pbinfo.ro/probleme/2306/numere22
- [ ] #1345 **kprim** — ușoară
  https://www.pbinfo.ro/probleme/1345/kprim

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #1264 **Exercițiul #1264**
  secvență pe numărul n (Bac 2021)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1264
- [ ] #814 **Exercițiul #814**
  secvență cu while și i = 2 (Bac 2017)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=814
- [ ] #1620 **Exercițiul #1620**
  care secvențe afișează numărul tuturor divizorilor
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1620
- [ ] #1619 **Exercițiul #1619**
  câte cifre de 0 are la sfârșit n!
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1619
- [ ] #821 **Exercițiul #821**
  x, y, d naturale strict pozitive (Bac 2016)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=821

Citește și exersează în plus

- Descompunerea în factori primi https://www.pbinfo.ro/articole/75/descompunerea-in-factori-primi
- Aplicații ale descompunerii în factori primi https://www.pbinfo.ro/articole/67/aplicatii-ale-descompunerii-in-factori-primi
- Algopedia — descompunerea în factori primi — explicații și probleme în română https://algopedia.ro/
- Python Tutor pentru C++ — urmărește cum scade n și cum crește exponentul e https://pythontutor.com/cpp.html
- CSES — Counting Divisors — aceeași idee, dar pentru mulți n: ce faci când un singur n nu mai e suficient? https://cses.fi/problemset/task/1713

Răspunsuri

Exersează pe hârtie

1. 84 = $2^2 \cdot 3 \cdot 7$; 1001 = $7 \cdot 11 \cdot 13$; 1024 = 2^{10}; 97 este prim (97^1).

2. $(6 + 1)(6 + 1) = \mathbf{49}$.

3. `100!`: 20 + 4 = **24** de zerouri. `30!`: 30/5 = 6, 30/25 = 1 → **7** zerouri.

4. `2 2` și `3 1` ($12 = 2^2 \cdot 3$), fiecare pe rând nou.

5. `7`.

6. cmmdc = $2^2 \cdot 3^2 = \mathbf{36}$; cmmmc = $2^3 \cdot 3^4 \cdot 5 = 8 \cdot 81 \cdot 5 = \mathbf{3240}$. Verificare: $36 \cdot 3240 = 116\,640 = 360 \cdot 324$.

7. Numerele cu exact 4 divizori sunt de forma $p \cdot q$ ($p \neq q$ primi) sau p^3: 6, 8, 10, 14, 15, 21, 22, 26, 27. ($8 = 2^3$ și $27 = 3^3$ sunt de forma p^3.)

8.

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

int main()
{
    long long n, d = 2, s = 0;
    cin >> n;
    while (d * d <= n)
    {
        if (n % d == 0)
        {
            s += d;
            while (n % d == 0) n /= d;
        }
        d++;
    }
    if (n > 1) s += n;
    cout << s;
    return 0;
}
```