Prelucrarea divizorilor

1. Noțiune
2. Prezentare generală
3. Algoritm eficient
4. Numere prime
5. Cmmdc
6. Descompunerea in factori primi

Algoritmi pentru prelucrarea divizorilor

1. Noțiune

- Spunem că numărul natural nenul **n** este divizor al numărului natural **m**, dacă valoarea expresiei de mai jos este adevărată.

```
m % n = 0
```

- următoarea secvență afișează un mesaj corespunzător

```
dacă m % n = 0 atunci
 scrie “divizor”
altfel
 scrie “nu este divizor”
```

Algoritmi pentru prelucrarea divizorilor

2. Prezentare generală

P1 Se dă un număr natural **n**. Afișați divizorii săi.

#376 Se dă un număr natural **n**. Să se determine suma divizorilor săi.

#3666 Se dă **n**, număr natural nenul. Să se testeze dacă n are număr impa r de divizori.

#408 Se dă un număr natural **n**. Să se determine numărul de divizori ai ogl inditului lui **n**.

- aceste probleme au în comun faptul că trebuie determinați toți divizorii unui număr **n.**

- cum se poate realiza acest lucru?

Algoritmi pentru prelucrarea divizorilor

Algoritm neeficient

- o primă metodă de determinare a divizorilor constă în a observa că toți divizorii lui **n** sunt între **1** și **n**, inclusiv.
- putem parcurge numerele din acest interval și verifica dacă sunt într-adevăr divizori ai lui n, caz în care sunt luați în considerare.

```
citește n
pentru d ← 1, n executa
    dacă n % d = 0 atunci
        //prelucreaza d
```

Algoritmi pentru prelucrarea divizorilor

Algoritm eficient

- divizorii oricărui număr **n** sunt în pereche: dacă **d** este divizor al lui **n**, atunci și **k = n/d** este divizor al lui **n**.
- de exemplu, pentru n = 100 avem următorii divizori:
  - d = 1, k = 100
  - d = 2, k = 50
  - d = 4, k = 25
  - d = 5, k = 20
  - d = 10, k = 10 (atenție, caz particular)
- pentru a determina divizorii lui **n** este suficient să parcurgem numerele de la **1** la $\sqrt{\mathbf{n}}$.

Algoritmi pentru prelucrarea divizorilor

- un caz special îl constituie **pătratele perfecte**. În cazul lor trebuie evitată analizarea de două ori a lui $\sqrt{n}$, care este divizor al lui n.

- algoritmul devine:

```
citește n
pentru d ← 1, √n executa
 dacă n % d = 0 atunci
  //prelucreaza d
  k ← n / d //perecehea
  dacă k ≠ d atunci//evitam prelucrarea radacinii patrate
   //prelucreaza k
```

Algoritmi pentru prelucrarea divizorilor

- în C/C++ evităm utilizarea funcției **sqrt**, pentru calculul radicalului (**d <= sqrt(n)**).
- utilizăm o formă echivalentă: **d * d <= n**, mai eficientă!

Algoritmi pentru prelucrarea divizorilor

P1 Se dă un număr natural **n**. Afișați divizorii săi.

```
#include <iostream>
using namespace std;
int main(){
    int n;
    cin >> n;
    for(int d = 1; d * d <= n; d++)
        if(n % d == 0){
            cout << d << " ";
            int k = n / d;
            if(k != d)  cout << k << " ";
        }
    return 0;
}
```

Algoritmi pentru prelucrarea divizorilor

#376 Se dă un număr natural **n**. Să se determine suma divizorilor săi.

```
#include <iostream>
using namespace std;
int n, d, p;
long long s;
int main(){
    cin >> n;
    s = 0;
    for(d = 1; d * d <= n; d++){
        if(n % d == 0){
            p = n / d;///calculez perechea
            s = s + d;///prelucrez divizorul
            if(p != d)///daca perechea e diferita de d
                s = s + p;///prelucrez perechea
        }
    }
    cout << s;///afisez suma
    return 0;
}
```

Algoritmi pentru prelucrarea divizorilor

#3666 Se dă **n**, număr natural nenul. Să se testeze dacă n are număr impar de divizori.

```
citește n
cnt ← 0
pentru d ← 1, √n executa
    dacă n % d = 0 atunci
        cnt ← cnt + 1//prelucreaza d
        k ← n / d //perecehea
        dacă k ≠ d atunci//evitam prelucrarea radacinii patrate
            cnt ← cnt + 1//prelucreaza k
```

Algoritmi pentru prelucrarea divizorilor

```
dacă cnt % 2 = 1 atunci
  scrie "da"
altfel
  scrie "nu"
```

#408 Se dă un număr natural **n**. Să se determine numărul de divizori ai ogl inditului lui **n**.

```
citește n
inv ← 0 //initial inv este nul
cât timp n ≠ 0 execută
   uc ← n % 10
   inv ← inv * 10 + uc
   n ← n / 10
cnt ← 0
```

Algoritmi pentru prelucrarea divizorilor

```
pentru d ← 1, √inv executa
 dacă inv % d = 0 atunci
  cnt ← cnt + 1//prelucreaza d
  k ← inv / d //perecehea
   dacă k ≠ d atunci//evitam prelucrarea radacinii patrate
    cnt ← cnt + 1//prelucreaza k
scrie cnt
```

Algoritmi pentru prelucrarea divizorilor

3. Numere prime

- un număr prim este un număr natural, **mai mare decât 1**, care are exact doi divizori: numărul 1 și numărul în sine.

Pentru a stabili dacă un număr este prim:

- fie numărăm divizorii săi. Dacă sunt 2 divizori, p este prim.
- fie căutăm divizorii proprii. Dacă nu găsim, numărul este prim.

Algoritmi pentru prelucrarea divizorilor

```
citeste n
prim ← 1 //presupunem ca numărul este prim
daca n < 2 atunci
  prim ← 0 //0 si 1 nu sunt prime
 altfel
  daca n>2 si n%2=0 atunci
   prim ← 0//numerele pare mai mari ca 2 nu sunt prime
  altfel
   d ← 3//evitam divizorii impari
   cat timp d*d ≤ n si prim = 1 executa
    dacă n % d = 0 atunci
     prim ← 0
    d ← d + 2
```

Algoritmi pentru prelucrarea divizorilor

```
dacă prim = 1 atunci
  scrie “da”
altfel
  scrie “nu”
```