Divizibilitate

Ștefan Dăscălescu

1 Motivație

De-a lungul parcursului vostru în domeniul algoritmicii, precum și de multe ori în diferite olimpiade și concursuri de informatică, va trebui să rezolvați multe probleme care se bazează pe un fundament matematic, studiul teoriei din spatele divizibilității numerelor naturale precum și a algoritmilor de aflare a numerelor prime, numărului de divizori, lucrului eficient cu numerele prime devenind toate foarte importante pentru asimilarea în cel mai bun mod posibil a acestui capitol. Totuși, acest document reprezintă doar un punct de plecare în ceea ce privește aplicațiile teoriei numerelor în algoritmică, alte concepte fiind discutate în documentele ulterioare. Aceste noțiuni se vor găsi foarte des în problemele de informatică pentru clasele de gimnaziu și clasa a IX-a.

2 Noțiuni introductive

Definiția 1: Un număr x este numit **divizor** al altui număr y, dacă y se poate scrie ca produsul dintre x și un alt număr întreg t.

Observația 1: Orice număr n se împarte la 1 și la el însuși.

Definiția 2: Definim un divizor comun al unei perechi de numere (a, b) ca fiind un număr c care este un divizor atât al lui a, cât și al lui b.

Definiția 3: Definim cel mai mare divizor comun(c.m.m.d.c.) al unei perechi de numere (a, b) ca fiind cel mai mare număr care este un divizor atât al lui a, cât și al lui b. Vom nota $x = (a, b)$.

Definiția 4: Definim cel mai mic multiplu comun(c.m.m.m.c.) al unei perechi de numere $[a, b]$ ca fiind cel mai mic număr care este un multiplu atât al lui a, cât și al lui b. Vom nota $x = [a, b]$.

Observația 2: $a \cdot b = (a, b) \cdot [a, b]$. Drept concluzie, $(a, b) = \frac{a \cdot b}{[a,b]}$.

Pentru aflarea celui mai mare divizor comun a două numere, există doi algoritmi principali. Primul dintre ei se bazează pe scăderi repetate, la fiecare pas scăzându-se din numărul mai mare, numărul mai mic până când cele două valori devin egale. Deși pentru multe perechi de numere acest algoritm este destul de eficient, atunci când diferența dintre numere este foarte mare, algoritmul va rula în timp cvasi-liniar (de exemplu, pentru numerele 3 și 10^9, un calculator are nevoie de câteva secunde să afle cmmdc-ul folosind acest algoritm).

De aceea vom folosi algoritmul lui Euclid prin împărțiri repetate pentru a ajunge la răspuns. Acest algoritm pleacă de la ideea că o slăbiciune majoră a algoritmului prin scăderi este dată de situația când raportul dintre numărul mai mare și cel mai mic este foarte mare, când practic efectuăm aceeași operație de foarte multe ori. De aceea, în loc de scăderi, la fiecare pas vom afla restul împărțirii numărului mai mare la cel mai mic, înlocuind posibilele operații de scădere cu o singură împărțire, algoritmul devenind mult mai eficient.

De exemplu, să analizăm numerele 40 și 18.

- $a = 40$, $b = 18$. $a\%b = 4$, noile valori fiind $a = 18$, $b = 4$
- $a = 18$, $b = 4$. $a\%b = 2$, noile valori fiind $a = 4$, $b = 2$
- $a = 4$, $b = 2$. $a\%b = 0$, noile valori fiind $a = 2$, $b = 0$
- $a = 2$, $b = 0$, deoarece $b = 0$, iar continuarea algoritmului ne-ar duce la împărțiri la 0, operație ce nu este validă.

Mai jos puteți găsi implementarea în C++ a cmmdc-ului și a cmmmc-ului, program ce află cmmdc și cmmmc pentru t perechi de numere. Complexitatea algoritmului este $O(\log n)$ pentru fiecare test.

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

int gcd(int a, int b)
{
        while(b > 0)
        {
                int c = a % b;
                a = b;
                b = c;
        }
        return a;
}
int main()
{
        int t;
        cin >> t;
        for(int i = 1; i <= t; i++)
        {
                int a, b;
                cin >> a >> b;

                int cmmdc = gcd(a, b);
                long long cmmmc = 1LL * a / cmmdc * b;
                cout << cmmdc << " " << cmmmc << '\n';
        }
        return 0;
}
```

3 Lucrul cu divizorii unui număr

Definiția 5: Un număr $n \geq 2$ este **număr prim** dacă și numai dacă are doar 2 divizori: 1 și n.

Definiția 6: Un număr $n \geq 2$ este **număr compus** dacă și numai dacă nu este prim.

Observația 3: 0 și 1 nu sunt nici numere prime, nici numere compuse.

Observația 4: 2 este singurul număr prim par, celelalte numere prime fiind impare.

Definiția 7: Descompunerea în factori primi se bazează pe **Teorema fundamentală a aritmeticii**: Orice număr natural $n > 1$ se poate scrie în mod unic sub forma $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \cdots \cdot p_k^{e_k}$, unde $p_1 < p_2 < \cdots < p_k$ sunt numere prime, iar $e_1, e_2, \ldots, e_k$ sunt numere naturale nenule.

Observația 4: Se poate observa că numărul maxim de numere prime la care se împarte un număr n este foarte mic(de exemplu, pentru $n \leq 10^9$, sunt cel mult 9 numere prime în reprezentarea ca produs de factori primi.

Pentru a afla divizorii unui număr natural n, cel mai simplu (dar și ineficient) algoritm constă în a verifica pe rând fiecare număr 1 la n și să verificăm dacă n se împarte exact la acel număr. Pentru a optimiza acest algoritm, va trebui să folosim o altă observație importantă.

Observația 5: Dacă n se împarte exact la x, se va împărți exact și la $\frac{n}{x}$. Asta ne duce la ideea să verificăm doar divizorii până la $\sqrt{n}$, observație ce se va dovedi fundamentală în calculele și algoritmii pe care îî vom scrie pentru toate aceste probleme.

Astfel, vom putea afla orice informație legată de divizorii unui număr în $O(\sqrt{n})$, fie că e vorba de numărul de divizori, divizorii primi, descompunerea în factori primi și așa mai departe.

Problema divizibilitate de pe kilonova

Se dă un număr t și t numere naturale. Să se afle pentru fiecare dintre ele răspunsul la una din următoarele întrebări:

- 1 n: Să se afle dacă n este prim sau nu. În caz afirmativ se va afișa 'YES', altfel se va afișa 'NO'.
- 2 n: Să se afle câți divizori are n - de exemplu, dacă $n = 12$, se va afișa 6 (1, 2, 3, 4, 6, 12 sunt divizorii lui 12).
- 3 n: Să se afle numărul divizorilor primi ai lui n - de exemplu, dacă $n = 21$, se va afișa 2.
- 4 n: Să se afișeze descompunerea în factori primi pe care o are un număr, fiecare factor fiind scris pe o linie, în ordine **crescătoare** a numerelor prime - de exemplu, dacă $n = 60$, se vor afișa pe 3 linii separate 2 2, 3 1 și 5 1.

Fiecare tip de întrebare a fost implementat folosind o funcție separată pentru a arăta diferențele ce pot apărea de la un tip de întrebare la alta.

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

void tip1(int n)
{
        bool prim = 1;
        if(n == 1)
                prim = 0;
        for(int i = 2; i * i <= n; i++)
                if(n % i == 0)
                        prim = 0;
        if(prim)
                cout << "YES" << '\n';
```

```
        else
                cout << "NO" << '\n';
}

void tip2(int n)
{
        int nrdiv = 0;
        for(int i = 1; i * i <= n; i++)
                if(n % i == 0)
                {
                        nrdiv++;
                        if(n / i != i)
                                nrdiv++;
                }
        cout << nrdiv << '\n';
}

void tip3(int n)
{
        int nrdivprim = 0;
        for(int i = 2; i * i <= n; i++)
                if(n % i == 0)
                {
                        nrdivprim++;
                        while(n % i == 0)
                                n = n / i;
                }
        if(n > 1)
                nrdivprim++;
        cout << nrdivprim << '\n';
}
```

```
void tip4(int n)
{
        for(int i = 2; i * i <= n; i++)
                if(n % i == 0)
                {
                        cout << i << " ";
                        int cnt = 0;
                        while(n % i == 0)
                        {
                                cnt++;
                                n = n / i;
                        }
                        cout << cnt << '\n';
                }
        if(n > 1)
                cout << n << " " << 1 << '\n';
}

int main()
{
        int t;
        cin >> t;

        for(int i = 1; i <= t; i++)
        {
                int tip, n;
                cin >> tip >> n;

                // vom apela o alta functie pentru fiecare tip
                if(tip == 1)
```

```
                tip1(n);
        if(tip == 2)
                tip2(n);
        if(tip == 3)
                tip3(n);
        if(tip == 4)
                tip4(n);
    }

    return 0;
}
```

4 Probleme și lectură suplimentară

- Probleme cu divizibilitate de pe kilonova
- Number theory — Storing information about multiples/divisors
- Articol de pe usaco guide