Deljivost

Počnimo od vrlo jednostavne definicije pojma deljivosti:

Definicija 1 (Deljivost). Ceo broj a je **deljiv** celim brojem $b \neq 0$, ako postoji ceo broj q takav da je $a = bq$. Tada pišemo $b|a$ i za broj b kažemo da je **delilac** broja a.

Kada govorimo o deljivosti posmatramo i pozitivne i negativne brojeve. Međutim, znak broja ovde ne igra neku bitnu ulogu (ukoliko $b|a$ tada $b|-a$ i $-b|a$) tako da se često koncentrišemo na prirodne brojeve. Nula je deljiva svakim celim brojem (sa druge strane, "nulom se ne deli"). Jasno je i da za svako $a \in Z$, $1|a$ i $a|a$.

Definicija 2 (Prosti brojevi). Ceo broj $p > 1$ je **prost** ako p nema delilac d za koji važi $1 < d < p$. Ceo broj $m > 1$ je **složen** ako nije prost.

Primetimo da se pojmovi prost i složen broj odnose na **prirodne brojeve**. Broj je prost ako nema drugih pozitivnih delioca osim jedinice i sebe samog. Prosti brojevi igraju centralnu ulogu u teoriji brojeva i oni su "gradivni elementi" celih brojeva o čemu svedoči sledeća (intuitivna) teorema.

Teorema 1 (Osnovna Teorema Aritmetike). Svaki prirodan broj $n > 1$ se može na jedinstven način prikazati kao proizvod prostih činilaca (sa tačnošću do na njihov poredak).

Npr. $36 = 2 \cdot 2 \cdot 3 \cdot 3$ i $56 = 2 \cdot 2 \cdot 2 \cdot 7$ su **faktorizacije** (rastavljanja na proste činioce) brojeva 36 i 56. Deo "tačnost do na poredak" govori da se npr. faktorizacije $2 \cdot 2 \cdot 3 \cdot 3$ i $3 \cdot 2 \cdot 3 \cdot 2$ ne razlikuju. Drugim rečima, Teorema 1 tvrdi da za svaki prirodan broj $n > 1$ postoji jednistven prirodan broj k, jedinstveni prosti brojevi $p_1 < p_2 < \cdots < p_k$ i jedinstveni prirodni brojevi $a_1, a_2, \dots, a_k$ tako da je

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}.$$

Prethodna jednačina predstavlja **kanonsku faktorizaciju** broja n. Primetimo da, po definiciji, broj 1 nije ni prost ni složen. Jedan od razloga je upravo prethodna teorema: ukoliko bi 1 bio prost, tada ne bismo imali jedinstvenu faktorizaciju (npr. $56 = 2^3 \cdot 7 = 1 \cdot 2^3 \cdot 7 = 1^2 \cdot 2^3 \cdot 7 = \cdots$). Dakle, da bismo "upoznali" broj, potrebno je odrediti njegove proste delioce. **U ovoj lekciji se, za dati prirodan broj n, bavimo efikasnim rešavanjem sledećih problema:**

1. Odrediti da li je n prost broj.
2. Naći sve delioce broja n (ili njihov broj/sumu).
3. Naći kanonsku faktorizaciju broja n, tj. odrediti nizove $p[]$ i $a[]$.

Ispitivanje da li je n prost broj možemo uraditi vrlo jednostavno direktnom primenom definicije: dovoljno je proveriti da li je n deljiv nekim od brojeva $2, 3, \dots, n-1$. Ovu proveru možemo uraditi redom (npr. **for** petljom): ukoliko naiđemo na broj koji deli n, prekidamo dalju proveru i broj n proglašavamo složenim; u suprotnom, broj n proglašavamo prostim. Međutim, složenost ovog pristupa je $O(n)$ tj. u najgorem slučaju možemo izvršiti otprilike n operacija – npr. ukoliko je n prost broj, tada nećemo naći njegov delilac među brojevima $2, 3, \dots, n-1$ ali ćemo morati da proverimo **svaki od njih**. Ovo je u redu ako je n reda veličine 10^6 (pa čak i 10^8) ali ukoliko radimo sa brojevima reda veličine 10^9 ili čak 10^{15} pomenuti algoritmi su **previše spori**.

Jedno vrlo jednostavno zapažanje dovodi do značajno bržih algoritama. Naime, ukoliko je d delilac broja n tada je i $\frac{n}{d}$ takođe delilac broja n. Jasno, ukoliko je $1 < d < n$, tada i $1 < \frac{n}{d} < n$. Međutim, nama je od posebnog interesa činjenica da je bar jedan od brojeva d i $\frac{n}{d}$ manji ili jednak od $\sqrt{n}$. Zaista, ukoliko bi bilo $d > \sqrt{n}$ i $\frac{n}{d} > \sqrt{n}$ tada bi važilo $d \cdot \frac{n}{d} > \sqrt{n} \cdot \sqrt{n}$ odnosno $n > n$ što je nemoguće. Ovo za posledicu ima sledeću

Teorema 2 („$\sqrt{n}$ teorema"). Svaki složen prirodan broj n ima delilac d za koji važi $1 < d \leq \sqrt{n}$.

Zaključujemo da je za proveru da li je n prost (tj. za rešavanje Problema 1) dovoljno ispitati brojeve $2, 3, \dots [\sqrt{n}]$; ukoliko među njima ne pronađemo delilac broja n, znamo da je n prost. Sledeći pseudokod demonstrira ovaj postupak

```
=================================================================
function IsPrime( int n ) : Boolean

        if (n = 1) then
                return false;

        d ← 2;
        while (d * d <= n) do
                if (n mod d = 0) then
                        return false;
                d ← d + 1;
        end while

        return true;
end function
=================================================================
```

Primetimo da u kodu imamo eksplicitnu proveru za broj 1 – ovo takmičari neretko zaboravljaju. Umesto while petlje se (naravno) može koristiti i for petlja ali se preporučuje da se vrednost $\sqrt{n}$ izračuna na početku (umesto "for d = 1 to $\sqrt{n}$") da bi se izbeglo stalno korenovanje. Još jedno prirodno ubrzanje je prvo ispitati da li je $n = 2$ i u suprotnom za delioce proveravati samo neparne brojeve. Ovo je zaista "duplo" ubrzanje ali možemo i nešto bolje na osnovu sledeće

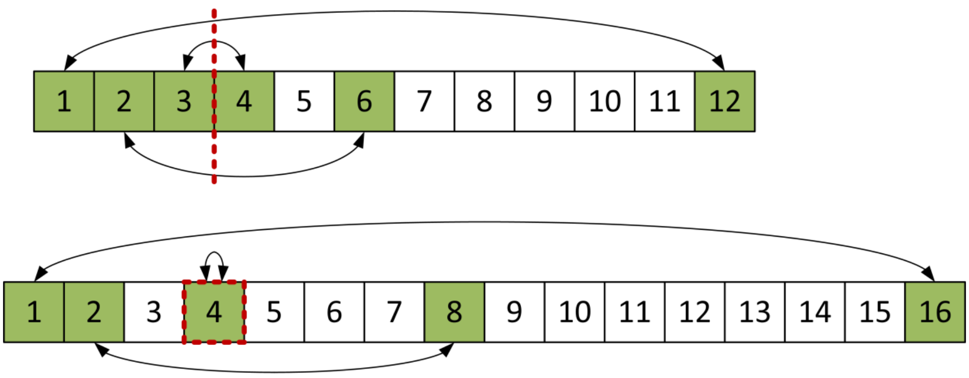
Teorema 3 („$6k \pm 1$ teorema"). Svaki prost broj veći od 3 je oblika $6k + 1$ ili $6k - 1$ za neko $k \in N$.

Ovo je prilično očigledno – prirodni brojevi daju ostatke $0, 1, 2, 3, 4, 5 = -1$ pri deljenju sa 6. Ako je broj veći od 3 i daje ostatak $0, 2$ ili 4 onda je on paran dok ostatak 3 implicira da je deljiv sa 3. Sa druge strane, nisu svi brojevi oblika $6k \pm 1$ prosti (npr. $6 \cdot 4 + 1$ i $6 \cdot 6 - 1$). Kako Teorema 3 može ubrzati naš algoritam? Za ispitivanje da li je n prost, dovoljno je proveravati da li je deljiv prostim brojevima iz segmenta $[2, \sqrt{n}]$; zaista, ako $d|n$ za neko $d \in [2, \sqrt{n}]$ tada svaki prost delilac broja d deli n. Naravno, određivanje svih prostih brojeva iz $[2, \sqrt{n}]$ je "teže" od ispitivanja da li je samo jedan broj (n) prost i zato ispitujemo samo potencijalne kandidate za proste brojeve – brojeve oblika $6k \pm 1$ kao i 2 i 3. Na taj način, među 6 uzastopnih prirodnih brojeva ispitaćemo samo 2 što je otprilike 3 puta brže od prethodnog algoritma. Implementacija ovog pristupa je data u sledećem pseudokodu.

```
=================================================================
    function IsPrime2( int n ) : Boolean

        if (n = 1) then return false;
        if (n = 2) or (n = 3) then return true;

        k ← 1;
        while ((6k – 1) * (6k – 1) <= n) do
            if (n mod (6k – 1) = 0) or (n mod (6k + 1) = 0) then
                return false;
            k ← k + 1;
        end while

        return true;
    end function
=================================================================
```

Složenost prethodna dva algoritma je $O(\sqrt{n})$, pri čemu prvi ispituje otprilike $\sqrt{n}$ a drugi $\frac{\sqrt{n}}{3}$ brojeva.

Vratimo se sada Problemu 2: određivanje **svih** delioca broja n. Ranije smo zaključili da svi delioci broja n dolaze u paru $(d, \frac{n}{d})$. Preciznije, ukoliko su $1 = d_1 < d_2 < d_3 < \cdots < d_m = n$ svi delioci broja n, tada za svako $i \in \{1, 2, \ldots m\}$ važi $d_i = \frac{n}{d_{m+1-i}}$ tj. najvećem deliocu "odgovara" najmanji i tako redom. Slučaj $d = \frac{n}{d}$ je ekvivalentan sa $n = d^2$; drugim rečima, ukoliko je n potpun kvadrat (i samo tada) jedan delilac (njegov koren) nema svog para.



Slika 1: Odgovarajući parovi delilaca su povezani strelicom. Za $n = 12$ to su $(1, 12), (2, 6)$ i $(3, 4)$ a za $n = 16$ to su $(1, 16), (2, 8)$ i "usamljeni" delilac 4.

Kako je u svakom paru manji delilac manji ili jednak od $\sqrt{n}$, dovoljno je ispitati samo brojeve iz segmenta $[1, \sqrt{n}]$ i za svaki pronađeni delilac d ispisati i $\frac{n}{d}$. Treba posebno voditi računa kada je n potpun kvadrat da ne bismo ispisali njegov koren dva puta – videti naredni pseudokod.

```
=================================================================
    function AllDivisors( int n )

        m ← 0;
        i ← 1;
        while (i * i < n) do
            if (n mod i = 0) then
                d[m + 1] ← i;
```

```
07                      d[m + 2] ← n / i;
08                      m ← m + 2;
09                end if
10                i ← i + 1;
11          end while

12          if (i * i = n) then
13                m ← m + 1;
14                d[m] ← i;
15          end if
16    end function
=================================================================
```

Prethodni algoritam vraća ukupan broj delilaca m i niz samih delilaca $d[]$. U liniji 04 imamo strogu nejednakost da bismo kasnije (linije 12-15) posebno ispitivali koren broja n ako je potrebno. Kao i u prethodnim algoritmima, ispitujemo otprilike $\sqrt{n}$ brojeva, tj. složenost je $O(\sqrt{n})$. Treba napomenuti da niz d nije sortiran jer redom ubacujemo parove (npr. za $n = 12$, $d = (1, 12, 2, 6, 3, 4)$). Ukoliko želimo sortirani niz, najlakši način je da koristimo dva niza: $d1$ u koji ćemo ubacivati delioce iz linije 06 i eventualno linije 14 i niz $d2$ u koji ćemo ubacivati delioce iz linije 07; na kraju niz $d2$ treba obrnuti i dodati na kraj niza $d1$.

Ostaje nam Problem 3, tj. određivanje kanonske faktorizacije broja n. Kao što je za očekivati i ovo je moguće uraditi u složenosti $O(\sqrt{n})$. Neka je $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$. Ideja je jednostavna: ispitivati redom brojeve $2, 3, \ldots$ (najviše do $[\sqrt{n}]$) i prvi (najmanji) koji deli n je njegov najmanji prost činilac (p_1). Zatim, dok god je moguće, delimo broj n brojem p_1 (tako određujemo a_1). Ostaje nam broj $n' = p_2^{a_2} \cdots p_k^{a_k}$ nad kojim ponavljamo prethodni postupak pri čemu ne krećemo sa traženjem broja p_2 ispočetka $(2, 3, \ldots)$ već od $p_1 + 1, p_1 + 2, \ldots$ (najviše do $[\sqrt{n'}]$) jer je $p_2 > p_1$. Ovo ponavljamo dok nam brojač (potencijalni delilac) ne bude veći od korena trenutno posmatranog broja (označimo ga sa x). Moguće su dve situacije:

- $x = 1$: U tom slučaju je pronađena kanonska faktorizacija broja n.
- $x > 1$: Kako x nema delilac $\leq \sqrt{x}$, prema Teoremi 2, on je prost. To je jedino moguće ako je $x = p_k^1$ i dodavanjem ovog prostog činioca kompletiramo faktorizaciju broja n.

```
=================================================================
function Factorization( int n )

      k ← 0;
      d ← 2;
      while (d * d <= n) do
            if (n mod d = 0) then
                  k ← k + 1;
                  p[k] ← d;
                  a[k] ← 0;
                  while (n mod d = 0) do
                        n ← n div d;
                        a[k] ← a[k] + 1;
                  end while
            end if
            d ← d + 1;
```

```
15          end while

16          if (n > 1) then
17                k ← k + 1;
18                p[k] ← n;
19                a[k] ← 1;
20          end if

21    end function
==============================================================
```

Ova ideja je prezenovana u prethodnom pseudokodu (broj k i nizovi p i a su oni iz faktorizacije). Glavna petlja je while petlja iz linija 04-15, dok je d brojac. Broj n **se menja** tokom algoritma (linija 10). Ukoliko $d|n$, on je najmanji delilac trenutnog broja n i u linijama 06-12 ga pamtimo i određujemo mu eksponent. Na kraju vršimo dodatnu proveru (linije 16-20) da li je ostao još jedan činilac.

Ovim smo rešili sva tri problema.

Napomena 1. Najbitnija stvar u ovoj lekciji je zapažanje da delioci prirodnog broja idu "u paru" što za posledicu ima Teoremu 2 (između ostalog). Ona omogućava rešavanje osnovnih problema (Problemi 1-3) u složenosti $O(\sqrt{n})$ što je **ogromno poboljšanje** u odnosu na trivijalne $O(n)$ algoritme: na ovaj način možemo raditi i sa brojevima reda veličine 10^{15} (na malo bržim računarima) dok linearni algoritmi prekoračuju vremenska ograničenja već za brojeve reda 10^9. Osim toga, prikazani algoritmi su jednostavni i često se koriste u rešavanju raznih potproblema zadataka iz teorije brojeva.

Napomena 2. U teoriji brojeva, standardna oznaka za broj (pozitivnih) delilaca broja n je $\tau(n)$ dok je oznaka za zbir svih (pozitivnih) delilaca broja n - $\sigma(n)$. Na osnovu zapažanja da delioci broja n idu u paru, nije teško dokazati da je $\tau(n) < 2\sqrt{n}$; ova procena je korisna ukoliko treba unapred deklarisati dužinu niza za sve delioce broja n. Kako su svi delioci ne veći od n, važi i $\sigma(n) < 2n\sqrt{n}$.

Napomena 3. Ako je $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ kanonska faktorizacija broja n, tada važi $n \geq 2^{a_1} 2^{a_2} \cdots 2^{a_k} \geq 2^k$ pa je $k \leq [\log n]$, tj. prirodan broj n ima najviše $[\log n]$ različitih prostih činilaca (logaritam sa osnovom 2) a uglavnom mnogo manje! Kanonska faktorizacija daje kompletnu informaciju i o deliocima broja n: svaki delilac broja n je oblika $p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}$ gde je $0 \leq b_i \leq a_i$, za svako $i = \overline{1, k}$. Preporučuje se čitaocu da, koristeći poslednju osobinu, dokaže sledeće jednakosti

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

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