# Eratostenovo sito

Sada ćemo se baviti rešavanjem sledećeg problema:

**Problem 1 .** Za dati prirodan broj $n$, naći **sve** proste brojeve u segmentu $[1, n]$.

Ovaj problem je značajan jer je u mnogim zadacima iz teorije brojeva često potrebno na početku izgenerisati sve proste brojeve iz nekog segmenta ili prvih nekoliko prostih brojeva (tj. izvršiti takozvano "preprocesiranje"), a zatim ih koristiti u daljem radu. Već znamo da proverimo da li je dati prirodan broj $n$ prost u složenosti $O(\sqrt{n})$ – ovo možemo iskoristiti i jednom for petljom (od 1 do $n$) pronaći sve proste brojeve u traženom segmentu. Za proveru da li je $i \in [1, n]$ prost, proveravamo najviše $\sqrt{i}$ brojeva pa je ukupna složenost ovog pristupa

$$\sqrt{1} + \sqrt{2} + \cdots \sqrt{n} < \sqrt{n} + \sqrt{n} + \cdots + \sqrt{n} = n\sqrt{n}$$

tj. složenost je $O(n\sqrt{n})$. Procena koju smo napravili nije previše gruba; može se pokazati da je suma $\sqrt{1} + \sqrt{2} + \cdots \sqrt{n}$ približno jednaka $\frac{2}{3}n\sqrt{n}$.

Međutim, ovde prezentujemo nešto brži algoritam baziran na ideji da ne proveravamo svaki broj posebno, već da posmatramo **sve brojeve odjednom**. Ideja je (pametno) "precrtavati" jedinicu i sve složene brojeve iz $[1, n]$; oni brojevi koji ostanu neprecrtani biće (svi) prosti brojevi iz $[1, n]$. Precrtavanje vršimo na osnovu sledeća dva zapažanja

1. Za proizvoljan prirodan broj $d > 1$, možemo odmah precrtati sve brojeve iz $[1, n]$ koji su veći od $d$ i deljivi sa $d$ jer su oni sigurno složeni.
2. Ukoliko ovo uradimo za $d = 2, 3, \ldots n$ precrtaćemo sve složene brojeve i oni brojevi koji ostanu su sigurno prosti!

Demonstrirajmo ovaj postupak za $n = 100$. Na početku precrtamo broj $1$. Zatim zaokružimo prvi neprecrtani broj – to je broj 2. Sada precrtavamo sve brojeve iz $[3, 100]$ koji su deljivi sa 2:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~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~~ | 31 | ~~32~~ | 33 | ~~34~~ | 35 | ~~36~~ | 37 | ~~38~~ | 39 | ~~40~~ |
| 41 | ~~42~~ | 43 | ~~44~~ | 45 | ~~46~~ | 47 | ~~48~~ | 49 | ~~50~~ | 51 | ~~52~~ | 53 | ~~54~~ | 55 | ~~56~~ | 57 | ~~58~~ | 59 | ~~60~~ |
| 61 | ~~62~~ | 63 | ~~64~~ | 65 | ~~66~~ | 67 | ~~68~~ | 69 | ~~70~~ | 71 | ~~72~~ | 73 | ~~74~~ | 75 | ~~76~~ | 77 | ~~78~~ | 79 | ~~80~~ |
| 81 | ~~82~~ | 83 | ~~84~~ | 85 | ~~86~~ | 87 | ~~88~~ | 89 | ~~90~~ | 91 | ~~92~~ | 93 | ~~94~~ | 95 | ~~96~~ | 97 | ~~98~~ | 99 | ~~100~~ |

Opet zaokružimo prvi sledeći neprecrtani broj (broj $3$) i precrtamo sve brojeve iz $[4, 100]$ koji su deljivi sa 3. Primetimo da je moguće da neki brojevi budu precrtani dva puta (u ovom slučaju, to će biti oni brojevi koji su deljivi i sa 2 i sa 3, tj. oni koji su deljivi sa 6).

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~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~~ | 31 | ~~32~~ | ~~33~~ | ~~34~~ | 35 | ~~36~~ | 37 | ~~38~~ | ~~39~~ | ~~40~~ |
| 41 | ~~42~~ | 43 | ~~44~~ | ~~45~~ | ~~46~~ | 47 | ~~48~~ | 49 | ~~50~~ | ~~51~~ | ~~52~~ | 53 | ~~54~~ | 55 | ~~56~~ | ~~57~~ | ~~58~~ | 59 | ~~60~~ |
| 61 | ~~62~~ | ~~63~~ | ~~64~~ | 65 | ~~66~~ | 67 | ~~68~~ | ~~69~~ | ~~70~~ | 71 | ~~72~~ | 73 | ~~74~~ | ~~75~~ | ~~76~~ | 77 | ~~78~~ | 79 | ~~80~~ |
| ~~81~~ | ~~82~~ | 83 | ~~84~~ | 85 | ~~86~~ | ~~87~~ | ~~88~~ | 89 | ~~90~~ | 91 | ~~92~~ | ~~93~~ | ~~94~~ | 95 | ~~96~~ | 97 | ~~98~~ | ~~99~~ | ~~100~~ |

Sledeći neprecrtani broj je $5$ – zaokružujemo ga i precrtavamo sve brojeve deljive sa $5$ iz $[6, 100]$.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~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~~ | 31 | ~~32~~ | ~~33~~ | ~~34~~ | ~~35~~ | ~~36~~ | 37 | ~~38~~ | ~~39~~ | ~~40~~ |
| 41 | ~~42~~ | 43 | ~~44~~ | ~~45~~ | ~~46~~ | 47 | ~~48~~ | 49 | ~~50~~ | ~~51~~ | ~~52~~ | 53 | ~~54~~ | ~~55~~ | ~~56~~ | ~~57~~ | ~~58~~ | 59 | ~~60~~ |
| 61 | ~~62~~ | ~~63~~ | ~~64~~ | ~~65~~ | ~~66~~ | 67 | ~~68~~ | ~~69~~ | ~~70~~ | 71 | ~~72~~ | 73 | ~~74~~ | ~~75~~ | ~~76~~ | 77 | ~~78~~ | 79 | ~~80~~ |
| ~~81~~ | ~~82~~ | 83 | ~~84~~ | ~~85~~ | ~~86~~ | ~~87~~ | ~~88~~ | 89 | ~~90~~ | 91 | ~~92~~ | ~~93~~ | ~~94~~ | ~~95~~ | ~~96~~ | 97 | ~~98~~ | ~~99~~ | ~~100~~ |

Ovo zatim ponavljamo (sledeći broj je 7) sve dok ima brojeva koji nisu precrtani i koji nisu zaokruženi. Na kraju dobijamo sledeću situaciju:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~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~~ | 31 | ~~32~~ | ~~33~~ | ~~34~~ | ~~35~~ | ~~36~~ | 37 | ~~38~~ | ~~39~~ | ~~40~~ |
| 41 | ~~42~~ | 43 | ~~44~~ | ~~45~~ | ~~46~~ | 47 | ~~48~~ | ~~49~~ | ~~50~~ | ~~51~~ | ~~52~~ | 53 | ~~54~~ | ~~55~~ | ~~56~~ | ~~57~~ | ~~58~~ | 59 | ~~60~~ |
| 61 | ~~62~~ | ~~63~~ | ~~64~~ | ~~65~~ | ~~66~~ | 67 | ~~68~~ | ~~69~~ | ~~70~~ | 71 | ~~72~~ | 73 | ~~74~~ | ~~75~~ | ~~76~~ | ~~77~~ | ~~78~~ | 79 | ~~80~~ |
| ~~81~~ | ~~82~~ | 83 | ~~84~~ | ~~85~~ | ~~86~~ | ~~87~~ | ~~88~~ | 89 | ~~90~~ | ~~91~~ | ~~92~~ | ~~93~~ | ~~94~~ | ~~95~~ | ~~96~~ | 97 | ~~98~~ | ~~99~~ | ~~100~~ |

Svi prosti brojevi iz $[1, 100]$ su zaokruženi a ostali su precrtani. Ovo je i bilo za očekivati: zaokruženi brojevi su upravo oni koji nisu bili precrtani prethodnim brojevima, tj. oni koji nemaju delioce veće od 1 – prosti brojevi. Sa druge strane, primetimo da je dovoljno koristiti samo proste brojeve za dalje precrtavanje a ne sve brojeve od 2 do $n$. Zaista, za proizvoljan broj $x$, ako $d|x$ tada i bilo koji prost činilac broja $d$ deli $x$ pa će $x$ biti precrtan od strane tog prostog činioca i pre nego što dođemo do broja $d$. Pomenuti algoritam je poznat kao **Eratostenovo sito**.

Da bismo "precrtavanje" pretvorili u kod, koristićemo logički niz prime[] dužine $n$. Ukoliko je prime[$i$] = true, tada je broj $i$ prost (neprecrtan) a u suprotnom je precrtan. Na početku ceo niz uzima vrednost true (nijedan broj nije precrtan) a u toku algoritma vršimo precrtavanje proizvoljnog broja $i$ jednostavnom dodelom “prime[i] = false“.

```
=================================================================
01     function Eratosten( int n )

02            Svakom element niza prime[] dodeliti vrednost true;
03            prime[1] ← false;

04            for i ← 2 to n do
05                   if (prime[i] = true) then
06                          for j ← 2 to [n/i] do
07                                 prime[i * j] = false;
08                          end for
09                   end if
10            end for

11     end function
=================================================================
```

Posle poziva funkcije Eratosten, broj $i$ je prost akko je prime[i] = true. U linije 05 se pitamo da li je broj $i$ neprecrtan; ukoliko jeste, precrtavamo sve brojeve veće od $i$ koji su deljivi sa $i$ (linije 06-08). Složenost algoritma je ukupan broj precrtavanja koji izvršimo: za dati broj $i$ precrtavamo brojeve $2i, 3i, \ldots \left[\frac{n}{i}\right] i$, tj. otprilike $\frac{n}{i}$ brojeva. Ukupan broj precrtavanja je $\frac{n}{1} + \frac{n}{2} + \cdots + \frac{n}{n} = n(1 + \frac{1}{2} + \cdots \frac{1}{n})$ što je približno jednako $n \ln n$ (ovo nije trivijalno pokazati). Međutim, mi precrtavamo samo prostim brojevima; ukoliko je $p_i$ – $i$-ti prost broj i $p_m$ najveći prost broj $\leq n$, složenost algoritma je

$$n\left(\frac{1}{p_1} + \frac{1}{p_2} + \cdots \frac{1}{p_m}\right) = n \ln \ln n.$$

Primetimo da su u prethodnom pseudokodu dve linije (04 i 06) obojene crveno – na ovim mestima se algoritam može malo ubrzati. Naime, već znamo da je svaki prirodan broj $n > 1$ ili prost ili ima prost delilac ne veći od $\sqrt{n}$. Prema tome, ukoliko neki broj iz $[1, n]$ nije bio precrtan brojevima manjim ili jednakim od $\sqrt{n}$, nikad neće ni biti, tj. sigurno je prost. Dakle, u liniji 04 možemo promeriti granicu for petlje u $[\sqrt{n}]$ i malo uštedeti. Sa druge strane, ukoliko trenutno precrtavamo prostim brojem $i$, nema potrebe da krećemo ispočetka, tj. da precrtavamo brojeve $2i, 3i, \ldots (i-1)i$. Zaista, svi brojevi oblika $j \cdot i$ za $2 \leq j < i$ su već precrtani – precrtao ih je neki prost delilac broja $j < i$. Dakle u liniji 06 možemo krenuti od $i^2$ umesto od $2i$ što je još jedno ubrzanje; videti sledeći pseudokod.

```
=================================================================
01     function Eratosten2( int n ) // Bolja varijanta

02            Svakom element niza prime[] dodeliti vrednost true;
03            prime[1] ← false;

04            for i ← 2 to [√n] do
05                   if (prime[i] = true) then
06                          for j ← i to [n/i] do
07                                 prime[i * j] = false;
08                          end for
09                   end if
```

```
10          end for

11      end function
=================================================================
```

Eratostenovo sito se može vrlo lepo iskoristiti i za druge stvari. Jedna od njih je i **faktorizacija broja**. Za dati broj $n$ ovo već znamo da uradimo u složenosti $O(\sqrt{n})$. Ali šta ako npr. želimo da faktorišemo puno brojeva (npr. imamo puno upita) ili baš sve brojeve iz segmenta $[1, n]$? Umesto poznatog algoritma, izvršićemo jedno “preprocesiranje” na početku a zatim ćemo dati broj faktorisati mnogo brže od $O(\sqrt{n})$. Glavna ideja je sledeća: znamo da tokom Eratostenovog sita broj može biti precrtan više puta; međutim, **prvi put** će dati (složeni) broj $x$ biti precrtan od strane **njegovog najmanjeg prostog delioca** jer jednostavno “idemo redom” (for petlja iz linije 04). Ovo možemo iskoristiti da za svaki prirodan broj $i$ **izračunamo najmanji prost broj koji ga deli** – označimo tu vrednost sa divisor[i].

```
=================================================================
01      function EratostenExtended( int n )

02          for i ← 1 to n do
03              divisor[i] ← i;

04          for i ← 2 to ⌊√n⌋ do
05              if (divisor[i] = i) then
06                  for j ← i to ⌊n/i⌋ do
07                      divisor[i * j] = min(divisor(i * j, i);
08                  end for
09              end if
10          end for

11      end function
=================================================================
```

Vidimo da je jedina razlika između klasičnog Eratostenovg sita i računanja najmanjeg prostog delioca za svaki broj iz $[1, n]$ samo u početnim vrednostima i u liniji 07. Naime, broj $i > 1$ je trenutno neprecrtan ako je divisor[$i$] = $i$ i u liniji 05 se vrši ista provera kao i pre. Takođe, kada prvi put precrtamo broj $x$ (npr. brojem $i$), jednostavno stavimo divisor[$x$] = $i$ i više ga nikad nećemo precrtavati – ovo obezveđuje funkcija min iz linije 07. Niz divisor daje mnogo više informacija nego niz prime; primetimo i da je broj $i > 1$ prost akko $divisor[i] = i$. Dakle, za prirodan broj $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, divisor[$n$] je upravo $p_1$ i, ukoliko ga želimo faktorisati, delimo ga ovim brojem sve dok je to moguće. Kada nam ostane broj $n' = p_2^{a_2} \cdots p_k^{a_k}$, posmatramo divisor[$n'$] = $p_2$ i ponavljamo postupak. Završavamo kada nam ostane 1.

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

02          k ← 0;
03          while (n > 0) do
04              k ← k + 1;
05              p[k] = divisor[n];
06              a[k] ← 0;
07              while (n mod p[k] = 0) do
08                  n ← n div p[k];
09                  a[k] ← a[k] + 1;
```

```
                 end while
           end while

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

Složenost ovog algoritma je $O(a_1 + a_2 + \cdots + a_k)$ što je mnogo manje od $\sqrt{n}$.

Za kraj, pomenimo da algoritam Eratostenovog sita ima i jednu očiglednu manu: za ispitivanje brojeva iz segmenta $[1, n]$ potreban je niz dužine $n$, tj. $O(n)$ memorije! Prema tome, ovaj algoritam treba primenjivati za $n \leq 10^6$(ponekad čak i za $n \leq 10^7$) i kada se zahteva nalaženje svih prostih brojeva ili mnogo upita tog tipa za segment $[1, n]$. Kada radimo sa brojevima reda veličine $10^{12}$ ili imamo malo upita "da li je broj prost" ili "faktorisati broj", treba se držati poznatih $O(\sqrt{n})$ algoritama.

**Napomena 1.** Niz $prime[]$ na kraju Eratostenovog sita omogućuje brzo odgovaranje na upite tipa "da li je broj $i$ prost" (if prime[i] = true …); međutim, ukoliko nam je potrebno da (često) iteriramo po svim prostim brojevima iz datog segmenta $[1, n]$, praktičnije je imati niz $p[]$ gde je $p[i]$ – $i$-ti prost broj. Ovaj niz lako možemo dobiti na osnovu niza $prime[]$ jednom for petljom i ubacivanjem brojeva $i$ za koje je $prime[i] = true$ u niz $p[]$.

**Napomena 2.** Označimo sa $\pi(n)$ broj prostih brojeva manjih ili jednakih od $n$, a sa $p_n$ – $n$-ti prost broj. Korisno je znati proceniti ove vrednosti (npr. da bismo unapred zadali veličinu niza) – u tome nam pomaže **Teorema o prostim brojevima** koja tvrdi da je $\pi(n)$ asimptotski jednako $\frac{n}{\ln n}$ a $p_n$ asimptotski jednako $n \ln n$. U sledećoj tabeli su date neke vrednosti za $\pi(n)$ i $p_n$.

| $\boldsymbol{n}$ | $\boldsymbol{\pi(n)}$ | $\boldsymbol{p_n}$ |
|---|---|---|
| 100 | 25 | 541 |
| 1.000 | 168 | 7.919 |
| 10.000 | 1.229 | 104.729 |
| 100.000 | 9.592 | 1.299.709 |
| 1.000.000 | 78.498 | 15.485.863 |
| 10.000.000 | 664.579 | 179.424.673 |
| 100.000.000 | 5.761.455 | 2.038,074.743 |
| 1.000.000.000 | 50.847.534 | 22.801.763.489 |

Ovi podaci se mogu koristiti i za procenu efikasnosti algoritma. Npr. ukoliko imamo zadatak da ispišemo sve brojeve koji su jednaki zbiru dva prosta broja manja od $n \leq 100.000$, pristup "fiksiraj svaka dva broja manja od $n$, proveri da li su prost i ispiši zbir", koji radi u $O(n^2)$ pod pretpostavkom da koristimo niz $primes[]$, je previše spor. Međutim, ukoliko fiksiramo svaka dva prosta broja niza $p[]$, iz tabele vidimo da najviše imamo oko $9.592^2$ operacija što bi trebalo da radi dovoljno brzo!