SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Karlo Peček

ALGORITIMI ZA GENERIRANJE PROSTIH BROJEVA MANJIH OD N

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U ovom diplomskom radu obradit će se algoritmi za generiranje prostih brojeva manjih od zadanog prirodnog broja N. Ti algoritmi implementirat će se u programskom jeziku Python i analizirat će se njihova složenost. Prvo će se definirati osnovni pojmovi o prostim brojevima, te utvrditi njihova važnost. Nakon toga napravit će se usporedba efikasnosti Eratostenovog, Atkinovog, te Sundaramovog sita, te njihove moguće optimizacije. Konačno, utvrdit će se brzina izvršavanja svakog pojedinog algoritma na računalu.

Poglavlje 1

Prosti brojevi

Za početak uvedimo osnovne pojmove, definicije i teoreme vezane uz proste brojeve.

Skup prirodnih brojeva označen je s $\mathbb{N}$, a s N označavamo gornju granicu niza prostih brojeva.

1.1 Definicija prostih brojeva

Definicija 1.1.1. *Kažemo da je prirodan broj p prost ako vrijedi:*

1. $p > 1$,
2. *p nema prirodnih djelitelja osim 1 i p.*

U nastavku rada, proste brojeve označavat ćemo s p.

Definicija 1.1.2. *Definiramo funkciju $\pi : [2, +\infty\rangle \to \mathbb{N}$ s*

$$\pi(x) = \text{card}\{p : p \leq x\}.$$

Funkcija π označava broj prostih brojeva manjih ili jednakih x.

Definicija 1.1.3. *Mersennovi prosti brojevi su prosti brojevi oblika $2^n - 1$.*

Do sada najveći poznati prost broj je $2^{74\,207\,281} - 1$, sadrži 22 338 618 znamenaka. Pronađen je 2016. godine i to od strane udruženja volontera *Great Internet Mersenne Prime Search* (*GIMPS*), koji se bave potragom Mersennovih prostih brojeva.

1.2 Važnost prostih brojeva

Sljedeći teorem pokazuje važnost prostih brojeva.

Teorem 1.2.1. *Svaki prirodan broj n, osim 1, je produkt prostih brojeva.*

Dokaz. Imamo dva slučaja, ili je n prost ili n ima djelitelje između 1 i n. Pretpostavimo da n nije prost i neka je $m > 1$ najmanji djelitelj od n. Tada je m prost broj. U suprotnom bi vrijedilo

$$\exists l, \quad 1 < l < n, \quad l|m; \qquad l|m \Rightarrow l|n,$$

što je u kontradikciji s definicijom od m, kao najmanjeg djelitelja od n, većeg od 1. Dakle, n je prost ili je djeljiv prostim brojem manjim od n. Neka je to p_1. Tada imamo sljedeći slučaj:

$$n = p_1 n_1, \quad 1 < n_1 < n.$$

Sada je n_1 prost ili je djeljiv s prostim brojem p_2, koji je manji od n_1. Onda vrijedi sljedeće:

$$n = p_1 n_1 = p_1 p_2 n_2, \quad 1 < n_2 < n_1 < n.$$

Ponavljanjem ovog postupka dolazimo do strogo silaznog niza brojeva

$$n, n_1, n_2, \ldots, n_{k-1}, \ldots,$$

koji su svi veći od 1. Prije ili kasnije moramo prihvatiti alternativu da je n_{k-1} prost. Neka je to prost broj p_k. Tada vrijedi:

$$n = p_1 p_2 \ldots p_k,$$

što pokazuje da je n produkt prostih brojeva. □

Primjer 1.2.2. *Na primjer, broj* 666 *je produkt sljedećih prostih brojeva:*

$$666 = 2 \cdot 3 \cdot 3 \cdot 37.$$

Napomena 1.2.3. *Ako je $n = ab$, tada a i b ne mogu istovremeno oba biti veća od $\sqrt{n}$. Dakle, svaki složeni broj n djeljiv je prostim brojem p manjim ili jednakim $\sqrt{n}$.*

Teorem 1.2.4. (***Euklidov drugi teorem***)
Prostih brojeva ima beskonačno mnogo.

Dokaz. Neka je $a = \{2, 3, 5, \ldots, p\}$ niz svih prostih brojeva do proizvoljnog prostog broja p. Definiramo $q = (2 \cdot 3 \cdot 5 \cdots p) + 1$, kao umnožak elemenata niza a uvećan za jedan. Očito, q nije djeljiv niti s jednim prostim brojem iz niza a.

Prema teoremu 1.2.1, q je prost ili je djeljiv s nekim prostim brojem između p i q. U oba slučaja postoji prost broj p_1 koji je veći od p. Time smo dokazali teorem. □

1.3 Kvadratne forme

Atkinov algoritam [2] koristi binarne kvadratne forme pa ćemo ovdje dati njihovu definiciju.

Definicija 1.3.1. *Binarna kvadratna forma je homogeni polinom u dvije varijable, drugog stupnja s cjelobrojnim koeficijentima, tj.*

$$f(x, y) = ax^2 + bxy + cy^2, \quad a, b, c \in \mathbb{Z}.$$

Definicija 1.3.2. *Za broj n reći ćemo da je kvadratno slobodan ako nije djeljiv s niti jednim kvadratom prirodnog broja, osim s 1.*

Na primjer, broj 6 je kvadratno slobodan, jer je djeljiv samo s 1, 2, 3 i 6, dok broj 12 nije kvadratno slobodan, jer je djeljiv s $4 = 2 \cdot 2$.

Teorem 1.3.3. *Neka je n kvadratno slobodan prirodan broj, takav da je $n \in 1+4\mathbb{Z}$. Tada je n prost ako i samo ako jednadžba $4x^2+y^2 = n$ ima neparan broj cjelobrojnih rješenja (x, y), takvih da je $x > 0$, $y > 0$.*

Dokaz koristi činjenicu da je jedinična grupa $\mathbb{Z}[i]^*$ domene glavnih ideala $\mathbb{Z}[i]$ jednaka $\{1, -1, i, -i\}$, gdje je $i = \sqrt{-1}$. Ideja dokaza je naći predstavnike za polugrupu $\mathbb{Z}[i]/\mathbb{Z}[i]^*$ u domeni $\mathbb{Z}[i]$. Za proizvoljan skup K, $\#K$ označava broj elemenata skupa.

Dokaz. Tvrdnja vrijedi za $n = 1$, pa možemo pretpostaviti da je $n > 1$. Definiramo

$$S = \{(x, y) : y > 0,\ 4x^2 + y^2 = n\}$$

te

$$T = \{I : I \text{ je ideal norme } n \text{ u } \mathbb{Z}[i]\}.$$

Za svaki $(x, y) \in S$ definiramo $f(x, y) \in T$ kao ideal generiran izrazom $y + 2xi$.

1. Korak: f je injektivna. To je uistinu tako, jer su preostali generatori $-y - 2xi$, $-2x + yi$, te $2x - yi$, od kojih niti jedan ne može biti oblika $y' + 2x'i$, uz uvjet $y' > 0$.

2. Korak: f je surjektivna. Za proizvoljan $I \in T$ odaberemo generator $a + bi$. Tada je $a^2 + b^2 = n$, te je $b \neq 0$, jer je n kvadratno slobodan. Imamo četiri slučaja:

1. a je paran i $b > 0$, onda je $I = f(-a/2, b)$;
2. a je paran i $b < 0$, onda je $I = f(a/2, -b)$;
3. a je neparan i $a > 0$, onda je $I = f(b/2, a)$;
4. a je neparan i $a < 0$, onda je $I = f(-b/2, -a)$.

3. Korak: Ako je n prost tada je $\#T = 2$, iz čega slijedi da je

$$\#\{(x,y) : x > 0,\ y > 0,\ 4x^2 + y^2 = n\} = (\#S)/2 = (\#T)/2 = 1.$$

U suprotnom, n možemo zapisati kao produkt prostih faktora $n = p_1 p_2 \cdots p_r$. Broj ideala norme p_k je paran, pa je $\#T$ djeljiv s 2^r, gdje je $r \geq 2$. Slijedi da je

$$\#\{(x,y) : x > 0,\ y > 0,\ 4x^2 + y^2 = n\} = (\#S)/2 = (\#T)/2$$

djeljiv s 2^{r-1}, pa mora biti paran. □

Teorem 1.3.4. *Neka je n kvadratno slobodan prirodan broj takav da $n \in 1 + 6\mathbb{Z}$. Tada je n prost ako i samo ako jednadžba $3x^2+y^2 = n$ ima neparan broj cjelobrojnih rješenja (x,y), takvih da je $x > 0$, $y > 0$.*

Neka je $\omega = (-1+\sqrt{-3})/2$. Dokaz koristi činjenicu da je jedinična grupa $\mathbb{Z}[\omega]^*$ domene glavnih ideala $\mathbb{Z}[\omega]$ jednaka $\{1, \omega, \omega^2, -1, -\omega, -\omega^2\}$. Ideja dokaza je naći predstavnike za polugrupu $\mathbb{Z}[\omega]/\mathbb{Z}[\omega]^*$ u domeni $\mathbb{Z}[\omega]$.

Dokaz. Pretpostavimo da je $n > 1$. Definiramo

$$S = \{(x,y) : y > 0,\ 3x^2 + y^2 = n\},$$

te

$$T = \{I : I \text{ je ideal norme } n \text{ u } \mathbb{Z}[\omega]\}.$$

Za svaki $(x,y) \in S$ definiramo $f(x,y) \in T$ kao ideal generiran izrazom $x + y + 2x\omega$. Slično kao u prethodnom dokazu, pokaže se da je f bijekcija iz S u T.

Ako je n prost tada je $\#T = 2$, iz čega slijedi da je

$$\#\{(x,y) : x > 0,\ y > 0,\ 3x^2 + y^2 = n\} = (\#S)/2 = (\#T)/2 = 1.$$

U suprotnom, n možemo zapisati kao produkt prostih faktora $n = p_1 p_2 \cdots p_r$. Broj ideala norme p_k je paran, pa je $\#T$ djeljiv s 2^r, uz $r \geq 2$. Slijedi da je

$$\#\{(x,y) : x > 0,\ y > 0,\ 3x^2 + y^2 = n\} = (\#S)/2 = (\#T)/2$$

paran. □

Teorem 1.3.5. *Neka je n kvadratno slobodan prirodan broj takav da $n \in 11 + 12\mathbb{Z}$. Tada je n prost ako i samo ako jednadžba $3x^2-y^2 = n$ ima neparan broj cjelobrojnih rješenja (x,y), takvih da je $x > y > 0$.*

Neka je $\gamma = \sqrt{3}$. Dokaz koristi činjenicu da je jedinična grupa $\mathbb{Z}[\gamma]^*$ domene glavnih ideala $\mathbb{Z}[\gamma]$ jednaka $\{\pm(2+\gamma)^j : j \in \mathbb{Z}\}$. Ideja dokaza je naći predstavnike za polugrupu $\mathbb{Z}[\gamma]/\mathbb{Z}[\gamma]^*$ u domeni $\mathbb{Z}[\gamma]$.

Dokaz. Definiramo

$$S = \{(x, y) : |x| > y > 0,\ 3x^2 - y^2 = n\},$$

te

$$T = \{I : I \text{ je ideal norme } n \text{ u } \mathbb{Z}[\gamma]\}.$$

Za svaki $(x, y) \in S$ definiramo $f(x, y) \in T$ kao ideal generiran izrazom $y + x\gamma$. Kao što smo vidjeli u prethodna dva dokaza, dovoljno je pokazati da je f bijekcija iz S u T.

Definiramo $L = \log(2 + \gamma)$, te homeomorfizam $\mathrm{Log} : \mathbb{Q}[\gamma]^* \to \mathbb{R}^2$ s

$$\mathrm{Log}(a + b\gamma) = (\log|a + b\gamma|,\ \log|a - b\gamma|).$$

Tada je

$$\mathrm{Log}\,\mathbb{Z}[\gamma]^* = (L, -L)\mathbb{Z}.$$

Ako je $|b| > a > 0$, tada za $u = \log|a + b\gamma|$, $v = \log|a - b\gamma|$ vrijedi $|u - v| < L$, te ako vrijedi $|u - v| \leq L$, onda je $|a| \leq |b|$ ili $|a| \geq 3|b|$.

Injektivnost: Za proizvoljne $(x, y) \in S$, te $(x', y') \in S$, odredimo

$$(u, v) = \mathrm{Log}(x + y\gamma)$$

te

$$(u', v') = \mathrm{Log}(x' + y'\gamma).$$

Tada je $|u - v| < L$ i $|u' - v'| < L$, pa vrijedi $|u - v - u' + v'| < 2L$. Pretpostavimo da je $f(x, y) = f(x', y')$. Tada vrijedi

$$(u, v) - (u', v') \in (L, -L)\mathbb{Z},$$

iz čega slijedi da je $(u, v) = (u', v')$. Iz toga slijedi da je $(x', y') \in \{(x, y), (-x, -y)\}$. No, i y i y' su pozitivni pa slijedi da je $(x, y) = (x', y')$. Dakle, f je injektivna.

Surjektivnost: Za proizvoljni ideal $I \in T$ odaberemo generator $a+b\gamma$. Odredimo

$$(u, v) = \mathrm{Log}(a + b\gamma).$$

Zatim izaberemo cijeli broj j, tako da je udaljen najviše $1/2$ od $(v - u)/2L$, i stavimo

$$(x + y\gamma) = (a + b\gamma)(2 + \gamma)^j.$$

Tada vrijedi sljedeće

$$\mathrm{Log}(x + y\gamma) = (u + jL, v - jL),$$

te

$$|(u + jL) - (v - jL)| \leq L,$$

iz čega slijedi da je $|y| \leq |x|$ ili $|y| \geq 3|x|$. Međutim, $n = \pm(3x^2 - y^2)$, te $n \in 11 + 12\mathbb{Z}$, što povlači da je $n = 3x^2 - y^2$. Dodatno, vrijedi $3x^2 - y^2 > 0$ pa je $|y| \leq |x|$. Također, znamo da je $|y| \neq 0$ te $|y| \neq |x|$, jer je n kvadratno slobodan. Imamo dva slučaja: ako je $y > 0$ onda je $I = f(x, y)$; ako je $y < 0$ onda je $I = f(-x, -y)$.

Dakle, f je bijekcija iz S u T. Sada znamo da je $\#T = 2$ ako je n prost broj. U suprotnom, $\#T$ je djeljiv s 4 pa je

$$\#\{(x, y) : x > y > 0,\ 3x^2 - y^2 = n\}$$

sigurno paran. □

Poglavlje 2

Nizovi prostih brojeva

Prvi prosti brojevi su:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, \ldots .$$

Lako je konstruirati niz svih prostih brojeva do nekog zadanog broja N i to postupkom koji se naziva *Eratostenovo sito*.

2.1 Metoda sita i gustoća prostih brojeva

Definicija 2.1.1. (***Metoda sita***)
Metoda sita u teoriji brojeva generalizira principe izbacivanja složenih brojeva iz skupa prirodnih brojeva. Problem sita sastoji se od određivanja broja prostih brojeva u konačnom podskupu skupa prirodnih brojeva $\mathbb{N}$.

U nastavku opisat ćemo algoritme Eratostenovog, Sundaramovog i Atkinovog sita.

Sljedeći teoremi daju informaciju o asimptotskoj gustoći prostih brojeva, a trebat će nam za ocjenu složenosti algoritama koji rade na principu sita.

Teorem 2.1.2. (***O prostim brojevima***)
Funkcija π *može se aproksimirati s* $\frac{x}{\ln x}$, *točnije, vrijedi*

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1.$$

Dokaz. Dokaz se može naći u [3]. □

Teorem 2.1.3. (***Mertensov prvi teorem***)

$$\sum_{p \leq x} \frac{\log p}{p} = \log x + O(1)$$

kada $x \to \infty$.

Teorem 2.1.4. (***Mertensov drugi teorem***)

$$\sum_{p \leq x} \frac{1}{p} = \ln \ln x + O(1)$$

kada $x \to \infty$.

Teorem 2.1.5. (***Mertensov treći teorem***)

$$\sum_{p \leq x} \ln \left(1 - \frac{1}{p}\right) = -\ln \ln x - \gamma + o(1)$$

kada $x \to \infty$. *Pritom je* γ *Euler–Mascheronijeva konstanta,*

$$\gamma = \lim_{n \to \infty} \left(-\ln n + \sum_{k=1}^{n} \frac{1}{k} \right).$$

Napomena 2.1.6. *Dokazi Mertensovih teorema mogu se pronaći u [3] i [6].*

2.2 Algoritmi za generiranje nizova

U ovom potpoglavlju iskazujemo 3 najznačajnija algoritma za generiranje prostih brojeva, koji koriste princip sita. Algoritmi generiraju nizove svih prostih brojeva manjih od unaprijed zadanog broja N.

Prvi algoritam koji ćemo iskazati je *Eratostenovo sito*. To je jedan od prvih algoritama u povijesti i prvi algoritam za generiranje svih prostih brojeva manjih od unaprijed izabranog prirodnog broja. Osmislio ga je grčki matematičar, geograf i astronom *Eratosten* u 3. stoljeću prije Krista.

U literaturi se sljedeći algoritam često navodi kao Eratostenov algoritam.

Algoritam 2.2.1. *Za niz prirodnih brojeva od* $2, \ldots, N$*:*

1. *Kreiraj niz koji će sadržavati proste brojeve i u njega stavi broj* 2.
2. *Za svaki sljedeći prirodan broj, počevši od* 3*, provjeri je li djeljiv s nekim od prostih brojeva sadržanih u nizu.*

3. *Ako nije, dodaj ga u niz prostih brojeva. Ako jest, prijeđi na sljdeći broj.*
4. *Ponavljaj postupak dok ne dođeš do kraja.*

Međutim, ovo nije algoritam Eratostenovog sita, jer za svaki broj redom provjerava je li on složen. Zbog toga je jako neefikasan. Algoritam Eratostenovog sita izbacuje sve višekratnike prostih brojeva manjih od N.

Iskažimo sada algoritam Eratostenovog sita.

Algoritam 2.2.2. ***Eratostenovo sito:***

1. *Kreiraj niz prirodnih brojeva* $2, \ldots, N$.
2. *Postavi* $p = 2$, *najmanji prost broj.*
3. *Izbaci iz niza sve višekratnike od* p, *počevši s* $2p$, *sve do* N, *i to u koracima od* p. *Dakle, izbaci* $2p, 3p, \ldots$.
4. *Pronađi prvi sljedeći broj u nizu. Ako takav ne postoji, stani. Inače, postavi ga za* p *i ponovi 3. korak.*

Brojevi koji su preostali u nizu su prosti.

Napomena 2.2.3. *Algoritam možemo dodatno optimizirati na način da u trećem koraku algoritma krenemo od* p^2. *To možemo učiniti, jer smo u prijašnjim prolascima sigurno izbacili sve višekratnike od* p *manje od* p^2. *U implementacijama ćemo iskoristiti tu malu optimizaciju.*

Sljedeći algoritam je indijski matematičar *S. P. Sundaram* otkrio 1934. godine. Nastao je kao poboljšanje Eratostenovog sita, dakle, kao algoritam s manjom teorijskom složenošću.

Njegova prednost u odnosu na *Eratostanovo sito* je u tome što koristi upola kraći početni niz. Također, u Eratostanovom situ za svaki prost broj oblika $2i+1$, $i \in \mathbb{N}$, iz liste izbacujemo njegovih k višekratnika, dok u Sundaramovom situ izbacujemo brojeve $i + j(2i + 1)$, za $1 \leq j \leq \lfloor k/2 \rfloor$, tj. upola manje višekratnika.

Algoritam 2.2.4. ***Sundaramovo sito:***

1. *Kreiraj niz prirodnih brojeva* $1, \ldots, \frac{N}{2} - 1$.
2. *Iz niza izbaci sve brojeve oblika* $i + j + 2ij$, *gdje su* $i, j \in \mathbb{N}$, $1 \leq i \leq j$.
3. *Kreiraj rezultatni niz i u njega stavi broj* 2.
4. *Brojeve koji su preostali u nizu sita pomnoži s* 2 *i dodaj im* 1, *te ih spremi u rezultatni niz.*

Rezultatni niz sadrži sve proste brojeve strogo manje od N.

Primjer 2.2.5. *Pokažimo kako algoritam radi za generiranje prostih brojeva manjih od* 10.

Stavimo $N = 10$.

1. *Kreiramo niz duljine* $N/2 - 1 = 4$.
2. $L = \{1, 2, 3, 4\}$, *jedini element oblika* $i + j + 2ij$ *je* 4, *dakle, njega izbacujemo iz niza.*
3. $R = \{2\}$, $L = \{1, 2, 3\}$.
   Sada punimo rezultatni niz.
4. $1 \cdot 2 + 1 = 3 \to R = \{2, 3\}$,
   $2 \cdot 2 + 1 = 5 \to R = \{2, 3, 5\}$,
   $3 \cdot 2 + 1 = 7 \to R = \{2, 3, 5, 7\}$.

Posljednji algoritam je nastao 2003. godine, a napravili su ga matematičari Arthur O. L. Atkin i Daniel J. Bernstein [2]. Autori su željeli pronaći algoritam za generiranje prostih brojeva sa što manjom vremenskom složenošću. Napravili su algoritam koji je baziran na ireducibilnim kvadratnim formama, a namijenjen je izvođenju na računalu.

Algoritam zanemaruje prva tri prosta broja (2, 3, 5), te gleda ostatke pri dijeljenju indeksa elemenata liste sa 60 (koristeći $60 = 2 \cdot 3 \cdot 5$), i to samo ostatke koji su prosti brojevi ili 1. Oni su sadržani u skupu

$$\{1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59\}.$$

Broj koji ima ostatak u danom skupu je potencijalno prost broj. Za određivanje je li dani broj prost, algoritam koristi kvadratne forme.

Algoritam 2.2.6. ***Atkinovo sito:***

Svi ostatci pri dijeljenju su ostatci modulo 60. *Svi brojevi, uključujući* x *i* y, *su prirodni brojevi. Promjena stanja logičke varijable znači: ako je varijabla bila True, mijenjamo je u stanje False i obratno. Rješenja kvadratnih formi su parovi* (x, y). *Rezultati kvadratnih formi koji imaju neparan broj rješenja su potencijalno prosti brojevi (moraju biti još i kvadratno slobodni, da bi bili prosti). Rezultati koji imaju paran broj rješanja su složeni brojevi.*

Algoritam:

1. *Napravi listu rezultata i stavi u nju* 2, 3, 5.
2. *Napravi listu sita* ***logičkih*** *vrijednosti duljine* N. *Postavi sve elemente na* ***False****.*
3. *Za svaki element liste sita s indeksom* k, *takvim da pri dijeljenju sa* 60 *daje ostatak* r, *tj.* $k \equiv r \pmod{60}$*:*

a) ako je $r \in \{1, 13, 17, 29, 37, 41, 49, 53\}$*, za svaki par* (x, y)*, koji je rješenje jednadžbe* $4x^2 + y^2 = k$*, promijeni stanje logičke varijable s indeksom* k *u listi sita;*

b) ako je $r \in \{7, 19, 31, 43\}$*, za svaki par* (x, y)*, koji je rješenje jednadžbe* $3x^2 + y^2 = k$*, promijeni stanje logičke varijable s indeksom* k *u listi sita;*

c) ako je $r \in \{11, 23, 47, 59\}$*, za svaki par* (x, y)*, koji je rješenje jednadžbe* $3x^2 - y^2 = k$*,* $x > y$*, promijeni stanje logičke varijable s indeksom* k *u listi sita;*

d) ako je r *neki drugi broj, ne radi ništa.*

4. *Kreni s prvim elementom liste sita koji ima vrijednost logičke varijable jednaku* ***True****.*
5. *Dodaj njegov indeks u rezultatnu listu.*
6. *U listi sita postavi sve njegove višekratnike na* ***False****, počevši od njegovog kvadrata.*
7. *Ponavljaj korake* 4.–6.*, sve dok ne dođeš do kraja liste.*

2.3 Asimptotska vremenska složenost

Sada ćemo odrediti asimptotske vremenske složenosti navedenih algoritama.

Za Eratostenovo sito vrijedi da je asimptotska vremenska složenost jednaka $O(N \log \log N)$.

Izvod:

Algoritam iz niza duljine N izbacuje višekratnike *prostih* brojeva N/p puta, za svaki p. Znamo da je broj prostih brojeva manjih od m jednak $\pi(m)$, pri čemu je

$$\pi(m) \approx \frac{m}{\ln m},$$

te za i-ti prost broj p_i vrijedi $p_i \approx i \ln i$. Dakle, vrijedi sljedeće:

$$\sum_{p \le N} \frac{N}{p} = \sum_{i=1}^{\pi(\sqrt{N})} \frac{N}{p_i} \approx \frac{N}{2} + N \sum_{i=2}^{\frac{2\sqrt{N}}{\ln N}} \frac{1}{i \ln i}$$

$$\approx \frac{N}{2} + N \int_{i=2}^{\frac{2\sqrt{N}}{\ln N}} \frac{1}{i \ln i}\, di \approx N \ln \ln N + O(N). \tag{2.1}$$

Pokažimo da je asimptotska vremenska složenost Sundaramovog sita jednaka $O(N \log N)$.

Izvod:

Algoritam iz niza duljine $\frac{N}{2} - 1$ izbacuje elemente oblika $i + j + 2ij$, za $1 \leq i \leq j \leq \frac{N}{2} - 1$. Iz nejednakosti $i + j + 2ij \leq \frac{N}{2} - 1$ dobijemo $i \leq \frac{N-j}{1+2j}$. Ako uzmemo u obzir i drugu nejednakost vidimo da je $j \leq \frac{N}{3}$. Dakle,

$$\sum_{j=1}^{\frac{N}{3}} \frac{N-j}{1+2j} = N \sum_{j=1}^{\frac{N}{3}} \frac{1}{1+2j} - \sum_{j=1}^{\frac{N}{3}} \frac{j}{1+2j} = O(N \log N). \tag{2.2}$$

Ovdje smo iskoristili da je suma iza faktora N reda veličine $O(\ln N)$, a sljedeća suma je reda veličine samo $O(N)$, pa prvi član dominira.

Na kraju, odredimo složenost Atkinovog sita.

Izvod:

Za dovoljno veliki N možemo izračunati proste brojeve do N na sljedeći način. Neka je p_k prost broj reda veličine $\sqrt{\log N}$ i neka je W definiran kao 12 puta produkt svih prostih brojeva od 5 do p_k,

$$W = 12 \cdot (5 \cdot 7 \cdot 11 \cdots p_k); \quad p_k \approx \sqrt{\log N}.$$

Primijetimo da za W vrijedi

$$W \approx \exp \sqrt{\log N},$$

pa je W bitno manji od N.

Definiramo φ_1 kao broj jediničnih, tj. multiplikativno invertibilnih elemenata modulo W. Ekvivalentno, to je broj brojeva manjih od W, koji su relativno prosti s W,

$$A = \{a \in \mathbb{Z}_W : M(a, W) = 1\}, \quad \varphi_1 = \#A.$$

Tada, prema Mertensovom drugom teoremu 2.1.4, omjer φ_1/W ima asimptotsko ponašanje jednako $O(1/\log\log N)$, za velike N.

Definiramo φ_2 kao broj parova $(x \bmod W, y \bmod W)$ takvih da su $4x^2 + y^2$ ili $3x^2 + y^2$ ili $3x^2 - y^2$ jedinični elementi modulo W. Tada, prema standardnoj generalizaciji Mertensovog drugog teorema, φ_2/W^2 ima asimptotsko ponašanje jednako $O(1/\log\log N)$.

Odaberemo prirodan broj B koji je blizu $W\sqrt{N}$, prirodan broj $L < N/W$, te jedinični element δ modulo W.

Na sljedeći način određujemo proste brojeve iz $\delta + W\mathbb{Z}$, koji su između WL i $WL + WB$:

- Definiramo parove:

  $(a, b) = (4, 1)$, ako je $\delta \in 1 + 4\mathbb{Z}$;
  $(a, b) = (3, 1)$, ako je $\delta \in 7 + 12\mathbb{Z}$;
  $(a, b) = (3, -1)$, ako je $\delta \in 11 + 12\mathbb{Z}$.

- Nađemo sva rješenja ($x \bmod W, y \bmod W$) za $ax^2 + by^2 \in \delta + W\mathbb{Z}$. To možemo izvršiti u $W^{O(1)}$ operacija.
- Za svaki ($x \bmod W, y \bmod W$), prebrojimo sve (x, y) za koje je

  $$WL \leq ax^2 + by^2 < WL + WB$$

  te promijenimo stanja u listi iz 3. koraka algoritma. Broj potrebnih operacija je $O(B/W)$.
- Ostaje 6. korak algoritma – on se izvodi u $O(B)$ operacija.

Ukupan broj operacija za sve δ, za računanje svih prostih brojeva između WL i $WL + WB$ je:

$$W^{O(1)} + O(\varphi_2 B/W) + O(\varphi_1 B) = O(WB/\log\log N).$$

Dakle, za nalaženje svih prostih brojeva manjih od N, složenost je $O(N/\log\log N)$.

Poglavlje 3

Implementacija i testiranje

3.1 Implementacije osnovnih algoritama

Implementacije algoritama napravljene su u programskom jeziku *Python*, verzija 3.5. Sve implementacije učitavaju prirodan broj N s komandne linije.

Eratostenovo sito

Nakon učitavanja granice N, određuje se njezin drugi korijen, koji služi kao granica u petlji. Nakon toga, inicijalizira se niz duljine N, **booleovskih** vrijednosti 0 ili 1, u kojem su svi elementi, osim prva dva, postavljeni na 1. Nadalje, pokreće se funkcija koja mjeri vrijeme izvršavanja algoritma.

Sam algoritam se sastoji od jedne petlje koja ima granice od 2 do $\sqrt{N}$. Unutar petlje prvo provjeravamo jesmo li već ranije "izbacili" element iz liste, točnije, je li već promijenjena vrijednost pripadne varijable na 0. Ako je vrijednost varijable jednaka 1, algoritam ulazi u novu petlju koja izbacuje sve višekratnike indeksa trenutnog elementa u nizu, počevši od njegovog kvadrata.

Nakon što je početna petlja došla do kraja, algoritam prolazi kroz niz i sve indekse elemenata s vrijednošću 1 kopira u rezultatnu listu. Time se rezultatna lista sastoji isključivo od prostih brojeva manjih od učitanog N. Nakon što je rezultatna lista popunjena, zaustavlja se mjerenje vremena izvršenja, te se vrijeme ispiše na ekranu.

Kod:

```
import sys
import os
import time
```

```
input_var=input("Enter N: ")
N=int(input_var)
n=int(N**0.5)+1

array=[0]*2+[1 for i in range(2,N)]

start=time.clock()

for i in range(2,n):
    if array[i]:
        for j in range(i*i,N,i):
            array[j]=0

rez = list(i for i in range (2,N) if array[i])
print(time.clock()-start,"s")

os.system("pause")
```

Sundaramovo sito

Za razliku od implementacije Eratostenovog algoritma, Sundaramov algoritam za granicu ne uzima $\sqrt{N}$, nego $N/2$, te inicijalizira niz booleovskih vrijednosti isključivo na 1.

Nakon pokretanja mjerenja vremena, pokreće se glavna petlja koja ima granice od 1 do $N/2$. Algoritam zatim kopira indeks i trenutnog elementa niza, te "izbacuje" sve elemente s indeksom oblika $i + j + 2ij$, pri čemu se j kreće u granicama od trenutnog indeksa i, do $N/2$, s korakom 1.

Nakon što se "izbace" svi elementi oblika $i+j+2ij$, indeksi svih elemenata niza koji imaju vrijednost 1 se pomnože s 2, pridoda im se 1, te se spreme u rezultatni niz. Na kraju se zaustavi mjerenje vremena, te se vrijeme ispiše na ekranu.

Kod:

```
import sys
import os
import time

input_var=input("Enter N: ")
N=int(input_var)
n=int(N/2)
```

```
array=[1 for i in range(n)]

start=time.clock()

for i in range(1,n):
    j=i
    k=i+j+2*i*j
    while (k<n):
        if array[k]:
            array[k]=0
        j=j+1
        k=i+j+2*i*j

rez = list(i+i+1 for i in range (1,n) if array[i])
print(time.clock()-start,"s")

os.system("pause")
```

Atkinovo sito

Kod:

```
import sys
import os
import time

input_var=input("Enter N: ")#Input of our limit
N=int(input_var)#Converting to integer
n=int(N**0.5)+1#Precalculating square root of limit so we have to do
    it only once

array=[0,0,1,1]+[0 for i in range(4,N)]#Creating our sieving array of
    Boolean values
s=[3,5,7,11]#Prime congruent mod 12, except 2

start=time.clock()

for i in range(1,n):
   for j in range(1,n):
      z=4*i*i+j*j#First quadratic form
```

```
        if (z<N and (z%12==1 or z%12==5)):
           array[z]=1-array[z]#fliping Boolean value

        z=3*i*i+j*j#Second quadratic form
        if (z<N and (z%12==7)):
           array[z]=1-array[z]

        z=3*i*i-j*j#Third quadratic form
        if (i>j and z<N and (z%12==11)):
           array[z]=1-array[z]

for i in s:
       for j in range(i*i,N,i):
              if array[j]:
                     array[j]=0#Excluding multiplicatives of primes
                        in S

rez = list(i for i in range (N) if array[i])#Creating resulting list
print(time.clock()-start,"s")#Printing execution time

os.system("pause")
```

3.2 Optimizacije

U ovom poglavlju prikazat ćemo implementacije Eratostenovog sita koje su efikasnije od početne, standardne implementacije. Dakle, sljedeće implementacije imaju kraće vrijeme izvršavanja.

Napomena 3.2.1. *Same implementacije su već optimizirane za programski jezik Python. Naime, koristimo množenje, umjesto kvadriranja, te zbrajanje, umjesto množenja s* 2*, jer su te operacije brže.*

Na primjer, kad bismo kod implementacije Atkinovog sita koristili kvadriranje, umjesto množenja, u kvadratnim formama, algoritam bi za $N = 10^8$ *imao vrijeme izvršavanja* 360.270123 s*, umjesto* 153.347160 s.

Također, u algoritmima prvo provjeravamo jesmo li već neki element izbacili u prethodnom prolasku, uvjetom ***if array[i]****, kako ne bismo imali nepotrebne prolaske kroz petlju.*

Generator prostih brojeva, Eratostenovo sito

Ova implementacija generira niz prostih brojeva, počevši od 2.

Generatori se koriste za generiranje nizova elemenata kroz koje moramo proći samo jednom, kao u našem slučaju. Druga prednost generatora je što on ne pamti elemente, čime troši manje memorije za izvršavanje.

Funkcija vraća **generator** indeksa elemenata liste koji su označeni kao prosti brojevi. Bez obzira što, u suštini, obavljamo isti "posao" kao u originalnoj implementaciji **Eratostenovog sita**, vrijeme izvršenja **generatora** je kraće, zbog načina na koji je implementiran u jeziku Python.

Kod:

```
import sys
import os
import time

input_var=input("Enter N: ")
N=int(input_var)
n=int(N**0.5)+1

def generator(N):
   array=[0]*2+[1 for i in range(2,N)]
   for i in range(n):
      if array[i]:
         for j in range(i*i,N,i):
            array[j]=0
   for i in range(N):
      if array[i]:
         yield i

start=time.clock()
rez = list(generator(N))
print(time.clock()-start,"s")

os.system("pause")
```

Neparni prosti brojevi

Ovaj algoritam uzima u obzir samo neparne proste brojeve (jedini parni prost broj je 2, njega doda u rezultatnu listu). Algoritam smanjuje gornju granicu na

$\frac{N-3}{2}+1$, te prolaskom kroz petlju duljine $\frac{\sqrt{N}-3}{2}+1$, izbacuje sve brojeve oblika $2i(i+3)+3$, uvećavane za $2i+3$, sve dok ne dođe do kraja.

Da bi generirao niz prostih brojeva, algoritam sve indekse elemenata množi s 2 i pridodaje im 3.

Kod:

```
import sys
import os
import time

input_var=input("Enter N: ")
N=int(input_var)
n=int(N**0.5)

def odd(N):
    limit=(N-3)//2
    array=[1]*(limit+1)
    for i in range((n-3)//2+1):
        if array[i]:
            j=i+i+3
            k=j*(i+1)+i
            array[k::j]=[0]*((limit-k)//j+1)
    return [2]+[i+i+3 for i, val in enumerate(array) if val]

start=time.clock()
rez = odd(N)
print(time.clock()-start,"s")

os.system("pause")
```

Eratostenovo sito, generator s faktorizacijom 2, 3, 5

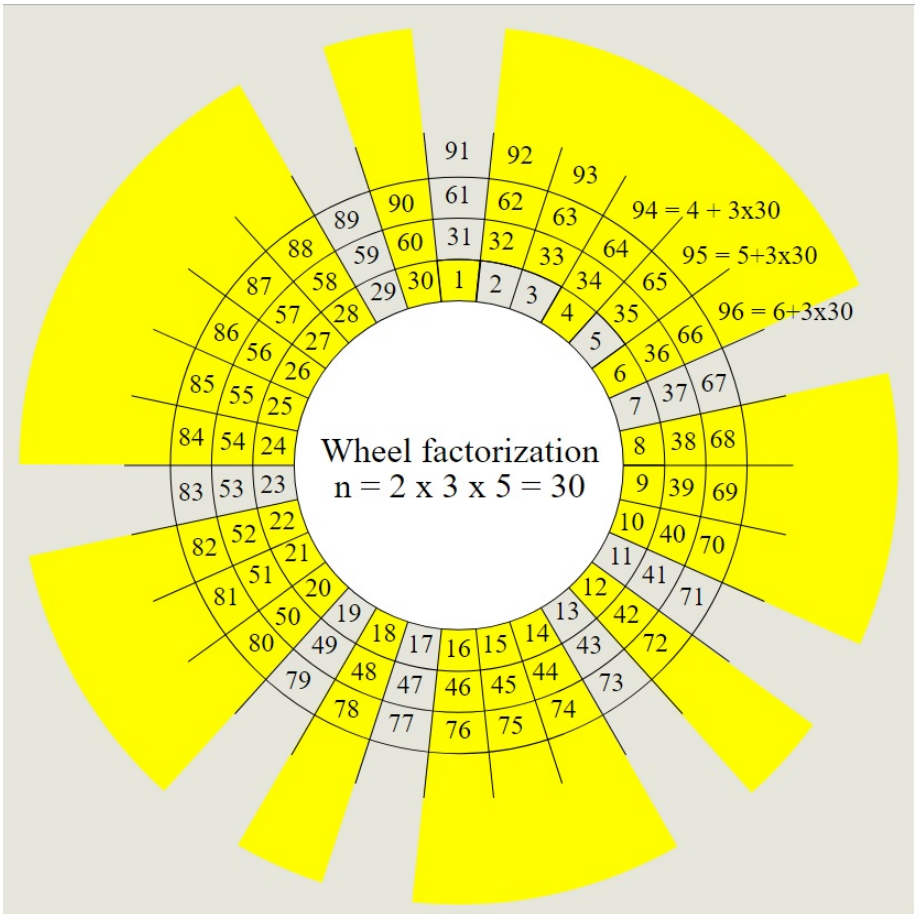
Najbrži algoritam generira niz prostih brojeva koristeći faktorizaciju kotačem, generiranim nizom 2, 3, 5.

Definicija 3.2.2. (***Faktorizacija kotačem***)

Specijalni slučaj faktorizacije brojeva, kod kojeg, umjesto standardnog postupka dijeljenja broja sa svim prostim brojevima manjim ili jednakim njegovom korijenu, dijelimo broj s početnim prostim brojevima $2, 3, 5$*, te sa svim brojevima koji su kon-*

gruentni s $\{1, 7, 11, 13, 17, 19, 23, 29\} \bmod 30$ *i manji od njegovog korijena, naziva se* ***faktorizacija kotačem*** $2, 3, 5$.

Općenito, ovaj tip faktorizacije naziva se ***faktorizacija kotačem****. Veličinu kotača određuje umnožak početnih prostih brojeva koje smo odabrali. U našem slučaju to je* $n = 2 \cdot 3 \cdot 5 = 30$*. Veličina kotača je proizvoljna. Najmanja veličina je kada za početni prost broj uzmemo samo* 2.



Brojevi koji su kongruentni s $\{1, 7, 11, 13, 17, 19, 23, 29\} \bmod 30$ *čine tzv. žbice kotača — oni su relativno prosti u odnosu na* $2, 3, 5$*, tj. generiraju jedinične elemente u* $\mathbb{Z}_{30}$.

Napomena 3.2.3. *Primijetimo da u kotaču nisu samo prosti brojevi. Ovaj postupak eliminira* $\frac{22}{30} = \frac{11}{15} \approx 73\%$ *brojeva.*

Također, napomenimo da veliki kotači nisu naročito efikasni. Za uklanjanje 90% *brojeva iz niza, trebali bismo koristiti proste brojeve do* 251*, za uklanjanje* 95% *do* $75\,037$*, za* 96% *do* $1\,246\,379$*, te za uklanjanje* 97% *proste brojeve do* $134\,253\,593$.

Kod:

```
import sys
import os
import time

input_var=input("Enter N: ")
N=int(input_var)

def primes235(limit):
    yield 2; yield 3; yield 5
    if limit < 7: return
    modPrms = [7,11,13,17,19,23,29,31]
    gaps = [4,2,4,2,4,6,2,6,4,2,4,2,4,6,2,6] # 2 loops for overflow
    ndxs =
        [0,0,0,0,1,1,2,2,2,2,3,3,4,4,4,4,5,5,5,5,5,5,6,6,7,7,7,7,7,7]
    lmtbf = (limit + 23) // 30 * 8 - 1 # integral number of wheels
        rounded up
    lmtsqrt = (int(limit ** 0.5) - 7)
    lmtsqrt = lmtsqrt // 30 * 8 + ndxs[lmtsqrt % 30] # round down on
        the wheel
    buf = [True] * (lmtbf + 1)
    for i in range(lmtsqrt + 1):
        if buf[i]:
            ci = i & 7; p = 30 * (i >> 3) + modPrms[ci]
            s = p * p - 7; p8 = p << 3
            for j in range(8):
                c = s // 30 * 8 + ndxs[s % 30]
                buf[c::p8] = [False] * ((lmtbf - c) // p8 + 1)
                s += p * gaps[ci]; ci += 1
    for i in range(lmtbf - 6 + (ndxs[(limit - 7) % 30])): # adjust for
        extras
        if buf[i]: yield (30 * (i >> 3) + modPrms[i & 7])

start=time.clock()
rez = list(primes235(N))
print(time.clock()-start,"s")

os.system("pause")

#kod preuzet sa
    https://rosettacode.org/wiki/Sieve_of_Eratosthenes#Python
```

3.3 Asimptotske vremenske složenosti implementacija

U ovom poglavlju više ćemo reći o asimptotskim vremenskim složenostima implementacija algoritama. Pod složenošću mislimo na broj prolazaka kroz petlju, prilikom generiranja liste prostih brojeva, koje pojedini algoritam izvrši. Sama brzina izvođenja ovisi o računalu na kojem se izvodi algoritam.

Eratostenovo sito

Asimptotska vremenska složenost ove implementacije algoritma jednaka je, već ranije spomenutoj, asimptotskoj složenosti Eratostenovog algoritma. Dakle, asimptotska vremenska složenost ovog algoritma je $O(N \log \log N)$. Vrijedi isti račun kao i za asimptotsku vremensku složenost (2.1) ovog algoritma.

Sundaramovo sito

I ova implementacija algoritma ima asimptotsku vremensku složenost jednaku asimptotskoj složenosti Sundaramovog algoritma, dakle, $O(N \log N)$. I ovdje vrijedi isti račun kao kod asimptotske vremenske složenosti (2.2).

Atkinovo sito

Složenost implementacije Atkinovog algoritma je kvadratna, dakle, $O(n^2)$. Naime, prolazimo kroz dvije petlje duljine n i za svaki par i, j provjeravamo jesu li rješenja jedne od tri kvadratne forme, $4x^2 + y^2 = n$, $3x^2 + y^2 = n$, $3x^2 - y^2 = n$, $x > y$, te još za svaki $p \in s = \{3, 5, 7, 11\}$ prolazimo kroz niz i izbacujemo njihove višekratnike, što je još $O(\log n)$. Dakle, vrijedi $O(n^2) + O(\log n) = O(n^2)$.

Generator prostih brojeva

Ova implementacija ima jednaku asimptotsku vremensku složenost kao i početna implementacija Eratostenovog sita, ali njezina prednost je u tome što ne pamti rezultatnu listu nego je generira na izlazu, i time štedi na memoriji. Dakle, asimptotska vremenska složenost ove implementacije je $O(n \log \log n)$.

Napomena 3.3.1. *Iako je jednake asimptotske vremenske složenosti kao i prvotna implementacija, u stvarnosti vidimo malo ubrzanje u izvršavanju, zbog načina na koji je generator implementiran u Pythonu.*

Neparni prosti brojevi

Asimptotska vremenska složenost ovog algoritma je otprilike dvostruko manja od složenosti početnog Eratostenovog algoritma, jer uzima u obzir samo neparne brojeve. Naime, jedini parni prost broj je dva, a svaki niz prirodnih brojeva sadrži pola parnih brojeva. Algoritam u startu n dijeli s dva, točnije generira niz duljine $\frac{n-3}{2}+1$. Duljina petlje smanjena je na $\frac{\sqrt{n}-3}{2}+1$. Iz niza duljine $\frac{n-3}{2}+1$ izbacujemo sve brojeve oblika $2i(i+3)+3$ uvećavane za $2i+3$. Takvih ima $\frac{N-4i^2-8i-3}{4i+6}$. Dakle, imamo približno $\frac{\sqrt{N}}{2}$ prolazaka, te najviše $\frac{N}{6}$ izbacivanja. Odatle slijedi da algoritam ima asimptotsku vremensku složenost $O(\sqrt{N}/2 \log\log N/2)$.

Faktorizacija kotačem 235

Ova implementacija algoritma ima približno jednaku asimptotsku vremensku složenost kao i prethodni algoritam *Neparni prosti brojevi*. Ovaj algoritam je malo brži i zauzima nešto manje memorije, jer radi nad bitovima. Prva petlja je duljine $\sqrt{n}$, dok u drugoj petlji izbacujemo $\frac{n}{p}$ višekratnika prostih brojeva iz skupa $\{7, 11, 13, 17, 19, 23, 29, 31\}$. Na taj način generiramo kotač. Zaključujemo da i ovaj algoritam ima složenost $O(\sqrt{n} \log\log n)$.

3.4 Vrijeme izvršavanja

U sljedećim tablicama su dana vremena izvršavanja raznih implementacija algoritama. Mjereno je vrijeme potrebno za kreiranje niza prostih brojeva manjih od N. Računalo na kojemu su izvršeni algoritmi opremljeno je dvojezgrenim procesorom Intel i3–2100 frekvencije 3,1 GHz, ima 16 GB DDR3 RAM-a frekvencije 1600 MHz. Vrijeme je mjereno u sekundama i zaokruženo na 6 decimala.

	Eratosten	Sundaram	Atkin
10^3	0.000242 s	0.000521 s	0.001404 s
10^4	0.002454 s	0.006820 s	0.013797 s
10^5	0.031549 s	0.090583 s	0.137024 s
10^6	0.336314 s	1.137708 s	1.510210 s
10^7	3.699307 s	12.887591 s	14.923805 s
10^8	38.225142 s	133.454970 s	153.347160 s

Napomena 3.4.1. *U tablici vidimo da, iako Eratostenov algoritam ima lošiju asimptotsku složenost, za velike N implementacija je višestruko efikasnija u odnosu*

na preostale dvije. Sundaramov algoritam je višestruko sporiji zbog while petlje koja je neefikasna (izračunavanje indeksa k).

Vremena optimiziranih algoritama

	Generator	Neparni	Faktorizacija 235
10^3	0.000232 s	0.000095 s	0.000163 s
10^4	0.002168 s	0.000565 s	0.000729 s
10^5	0.023262 s	0.005715 s	0.004896 s
10^6	0.292390 s	0.057394 s	0.041603 s
10^7	3.222195 s	0.641171 s	0.443251 s
10^8	34.957853 s	7.297996 s	4.740383 s

Napomena 3.4.2. *Kao što vidimo u tablici, faktorizacija uvelike ubrzava generiranje niza prostih brojeva za veliki N. Štoviše, to je najbrži algoritam.*

Bibliografija

[1] Melissa E. O'Neill. *The Genuine Sieve of Eratosthenes*. Harvey Mudd College, Claremont, CA, SAD.

[2] A. O. L. Atkin i D. J. Bernstein. *Prime Sieves using binary quadratic forms*. Mathematics of Computation, vol. 73, sv. 246, str. 1023–1030, 2003.

[3] G. H. Hardy i E. M. Wright. *An introduction to the Theory of Numbers*. Oxford University Press, Ely House, London W.1, Fourth Edition, 1960.

[4] Paul Pritchard. *Improved Incremental Prime Number Sieves*. Griffith University, School of Computing and Informationa Technology, Queensland, Australija.

[5] `https://rosettacode.org/wiki/Sieve_of_Eratosthenes#Python`

[6] `https://terrytao.wordpress.com/2013/12/11/mertens-theorems/`

Sažetak

U radu je obrađeno generiranje prostih brojeva manjih od N korištenjm tri algoritma: Eratostenovog, Sundaramovog i Atkinovog sita. U prvom poglavlju navedena je definicija prostih brojeva, te obrađena njihova važnost. Obrađene su kvadratne forme radi boljeg razumijevanja rada Atkinovog sita. U drugom poglavlju navedena je definicija niza prostih brojeva, te su iskazani teoremi koji dokazuju tvrdnje o asimptotskoj gustoći prostih brojeva u skupu prirodnih brojeva. Opisani su algoritmi sita, te je određena asimptotska vremenska složenost za svaki od njih. U trećem poglavlju izvršena je implementacija u programskom jeziku Python i izmjereno vrijeme izvršenja na računalu. Izvršena je optimizacija Eratostenovog algoritma, te je ponovo određena asimptotska vremenska složenost svake pojedine optimizacije. Na kraju je izvršeno usporedno mjerenje vremena izvršenja na računalu, iz čega je donesen zaključak.

Iz dobivenih rezultata vidimo da se najbrže na računalu izvršava Eratostenovo sito, a od njegovih optimizacija Faktorizacija kotačem 235. Iako Sundaramovo i Atkinovo sito imaju manju asimptotsku vremensku složenost u odnosu na Eratostenovo, samo izvršavanje u programskom jeziku Python je dulje zbog izračunavanja kvadratnih formi.

Summary

This Master's degree thesis deals with generating prime numbers smaller than N. It deals with three algorithms, Eratosten's sieve, Sundaram's sieve, and Atkin's sieve. The thesis is composed of three chapters. Chapter One is introductory and defines basic terminology used in the thesis. The chapter is subdivided into three parts. First Part gives the definition of prime numbers. Second Part tells about importance of prime numbers and finally Third Part explains quadratic forms. Quadratic forms were explained for better understanding of Atkin's sieve.

Chapter Two deals with prime number series and algorithms for generating them. Chapter consists of three parts. Part One gives a definition of a series of prime numbers, and theorems that demonstrate asymptotic density of prime numbers in a set of natural numbers. Part two describes sieving algorithms and in Part Three asymptotic time complexity is determined for each one of them. Chapter Three gives implementation of the algorithms in programming language Python and determines the time taken to execute them on the computer. Part Two of the Third Chapter describes optimizations of the Eratosten's algorithm. In Part Three the asymptotic time complexity of each optimization is determined. In Part Four we draw conclusions from tables of execution times on the computer.

From the results obtained we see that the Eratosten's sieve has the fastest execution time on a PC and that Factorization Wheel 235 is the fastest optimization. Although Sundaram's and Atkin's site have better asymptotic time complexity than Eratosten's, because of the implementations in programming language Python their execution time is longer due to quadratic form calculations.

Životopis

Rođen sam u Zagrebu 17. ožujka 1988. godine, gdje sam pohađao osnovnu školu Dobriše Cesarića koju sam završio 2002. godine. Nakon osnovne škole upisao sam IX. gimnaziju u Zagrebu, koju sam završio 2006. godine. Nakon toga upisao sam preddiplomski sveučilišni studij Matematika na Prirodoslovno–matematičkom fakultetu. Preddiplomski studij završio sam 2013. godine, te sam upisao diplomski sveučilišni studij Računarstvo i matematika, također, na Prirodoslovno–matematičkom fakultetu. Kroz studij sam naučio programirati u više programskih te skriptnih jezika: Assembly, C, C++, C#, Python, PHP, JavaScript, XML, HTML, CSS. Na Case Study Competition studentskom natječaju 2011. godine bio sam član pobjedničkog dvojca u natječaju za Splitsku banku s radom naziva Projekt X – Što ponuditi studentima. Kroz studij sam radio razne studentske poslove, od kojih bih izdvojio demonstrator u Računalnom praktikumu. Također, bavim se izradom i održavanjem web stranica.