Brzi algoritmi za traženje prostih brojeva

Kovač, Dubravka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

BRZI ALGORITMI ZA TRAŽENJE PROSTIH BROJEVA

Diplomski rad

Dubravka Kovač

Osijek, 2017.

SADRŽAJ

1. UVOD

Iako se čini da je pojam prostog broja prilično jednostavan i zdravorazumski lako razumljiv, prosti su brojevi zadavali mnogo muke matematičarima, a i danas još postoji nekoliko jednostavnih pitanja na koja se ne znaju odgovori. Znamo da prirodnih brojeva ima beskonačno mnogo, a tada je jasno da i parnih, kao i neparnih brojeva ima beskonačno mnogo. No, manje je jasan bio odgovor na pitanje ima li i prostih brojeva beskonačno mnogo, tj. može li se dokazati da iza svakog prostog broja postoji bar još jedan prost broj i koji je to broj? Nadalje, znamo da postoji formula za traženje parnih brojeva: $p(n) = 2n$ gdje je $n \in \boldsymbol{N}$, postoji i formula za traženja neparnih brojeva: $q(n) = 2n - 1$ gdje je $n \in \boldsymbol{N}$. Postavlja se i pitanje postoji li određena formula za traženje prostih brojeva? Na početku rada ćemo se upoznati s definicijom i svojstvima prostih brojeva te razdiobi i o gustoći prostih brojeva. Nadalje, proučavamo brze algoritme za traženje prostih brojeva. Da bismo imali osjećaj što je brzi algoritam, krenut ćemo od osnovnog algoritma. Osnovni algoritam na najjedostavniji način provjerava je li broj prost, tj. provjerava je li promatrani broj djeljiv s bilo kojim svojim prethodnikom. Nadogradnjom osnovnog algoritma su se izbjegla nepotrebna provjeravanja djeljivosti. Osim osnovnih algoritama, bit će obrađena i sita i to: Eratostenovo, Sundramovo i Atkinovo sito. Na kraju rada će se testirati svi spomenuti algoritmi. Testiranje će se izvršiti na istom računalu i na istom skupu brojeva, tj. na istom intervalu. Usporedbom vremena potrebnih za izvršavanje pojedinog algoritma, zaključit će se koji je najbrži, odnosno kojem algoritmu je potrebno najmanje vremena za izvršavanje. Radit će se i provjera algoritama jer svi algoritmi moraju dati isti rezultat, ali u različitim vremenima.

2. PROSTI BROJEVI

2.1. Definicije i svojstva prostih brojeva

U ovom radu će se promatrati samo skup prirodnih brojeva. Prirodni brojevi su nastali kao potreba za prebrojavanjem, odnosno uređivanje. Oni su osnova matematike i prvi brojevi do kojih je čovječanstvo došlo tijekom svog razvoja. Oznaka za skup prirodnih brojeva je $\boldsymbol{N}$ koja dolazi od početnog slova latinske riječi *naturalis* što prevedeno na hrvatski jezik znači prirodan. Najmanji prirodan broj je 1, a najveći prirodan broj ne postoji što znači da prirodnih brojeva ima beskonačno mnogo. Svaki prirodni broj u dekadskom sustavu se može napisati pomoću deset znamenaka: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Ove su znamenke nastale u Indiji prije 5. stoljeća, a Arapi su ih u srednjem vijeku prenijeli u Europu zbog čega se nazivaju arapski brojevi. Skup prirodnih brojeva strogo je definirao 1891. godine talijanski matematičar Giuseppe Peano (1858-1932) pomoću sljedećih pet aksioma:

(i) $1 \in \boldsymbol{N}$;

(ii) ako je $n \in \boldsymbol{N}$, onda je $n + 1 \in \boldsymbol{N}$;

(iii) ako je $n + 1 = m + 1$, onda je $n = m$;

(iv) nema prirodnog broja sa svojstvom $1 = n + 1$ (broj 1 nema prethodnika u $\boldsymbol{N}$);

(v) (aksiom matematičke indukcije) ako je A podskup skupa prirodnih brojeva takav da vrijedi

(a) $1 \in A$ (baza indukcije),

(b) iz $n \in A$ slijedi $n + 1 \in A$ (korak indukcije),

onda je $A = \boldsymbol{N}$.

Kod Francuza skup prirodnih brojeva dogovorno počinje s brojem 0. Naime, i Peanova originalna definicija imala je prvi aksiom (i)′ $0 \in \boldsymbol{N}$, a četvrti aksiom je glasio: (iv)′ nema prirodnog broja n takvog da je $0 = n + 1$ (nula nema prethodnika u $\boldsymbol{N}$). Mnogi matematičari i inženjeri radije broje od nula nego od jedan te moraju imati na umu da je tada ukupan broj za jedan veći od zadnjeg broja. Kod nas, kao i u većem dijelu svijeta, skup prirodnih brojeva počinje brojem 1. Skup prirodnih brojeva je $N = \{1, 2, 3, 4, 5, \dots\}$.

Uvođenje računske operacije množenja u skup prirodnih brojeva dovelo je do novih i zanimljivih problema u vezi s prirodnim brojevima. Množenje omogućuje određene podjele

prirodnih brojeva, kao na primjer, podjela prirodnih brojeva na parne brojeve $\{2, 4, 6, 8, \ldots\}$ i neparne brojeve $\{1, 3, 5, 7, \ldots\}$. Još važnija podjela prirodnih brojeva je podjela na proste brojeve i složene brojeve. Kada bismo prirodne brojeve gledali u odnosu na računsku operaciju zbrajanja, tada bismo mogli reći da je svaki prirodni broj, osim broja 1, složen broj jer se može prikazati kao zbroj drugih prirodnih brojeva, kao npr. $2 = 1 + 1$, $3 = 2 + 1 = 1 + 1 + 1$, $4 = 3 + 1 = 2 + 1 + 1 = 1 + 1 + 1 + 1$. Kada prirodne brojeve gledamo u odnosu na operaciju množenja, imamo kompliciraniju situaciju. Znamo da svaki prirodni broj pomnožen brojem 1 kao rezultat daje isti taj broj. Zanemarimo li množenje brojem 1, uviđamo da se ne mogu svi prirodni brojevi prikazati kao umnošci drugih prirodnih brojeva. Brojevi $2, 3, 5, 7, 11, 13$ itd. ne mogu se prikazati kao umnošci drugih prirodnih brojeva, kao što se recimo broj 4 može napisati kao umnožak broja 2 sa samim sobom (tj. $4 = 2 \cdot 2$), broj 6 se može prikazati kao umnožak brojeva 2 i 3 (tj. $6 = 2 \cdot 3$), nadalje $8 = 2 \cdot 4 = 2 \cdot 2 \cdot 2$, slijedi $9 = 3 \cdot 3$ i tako dalje. To je nagnalo još starogrčke matematičare da naprave podjelu prirodnih brojeva na proste i složene. Najjednostavnije rečeno, prosti brojevi su oni prirodni brojevi koji se ne mogu prikazati kao umnošci drugih prirodnih brojeva.

Definiciju prostog broja uveo je grčki matematičar Euklid, koji je živio na prijelazu iz 4. u 3. stoljeće prije Krista, u svom djelu ″Elementi″. Euklidovi ″Elementi″ su jedno od najvažnijih djela u povijesti čovječanstva i najčešće su prevođeno djelo u povijesti poslije Biblije. Djelitelje nekog broja $a \in \boldsymbol{N}$ ovdje promatramo samo u skupu prirodnih brojeva. Prirodni broj $a > 1$ ima uvijek dva djelitelja: broj 1 i broj a. Te djelitelje zovemo trivijalnim djeliteljima.

DEFINICIJA. Za prirodni broj $p > 1$ kažemo da je prost broj (ili prim broj) ako ima samo trivijalne djelitelje (tj. ako su mu jedini djelitelji 1 i p). Za broj $a > 1$ koji nije prost broj (tj. posjeduje netrivijalne djelitelje) kažemo da je složen broj.

Proste brojeve zovemo i prim brojevi ili prabrojevi.

Jednostavnije možemo reći da je broj $n > 1$ prost ako je djeljiv samo s 1 i sa samim sobom. Inače je broj složen. Broj 1 nije ni prost ni složen. Uočavamo da je broj 2 jedini paran prost broj.

Djelitelje od $a \in \boldsymbol{N}$ zovemo još i faktorima broja a. Faktorizaciju prirodnog broja a prikazujemo u obliku $a = bc$, gdje su $b, c \in \boldsymbol{N}$. Ako je djelitelj b prost broj, zovemo ga prostim djeliteljem ili prostim faktorom broja a. Sada ćemo u svrhu dokazivanja osnovnog teorema aritmetike promotriti nekoliko pomoćnih rezultata koje ćemo iskoristiti kasnije.

LEMA 1. Neka je a prirodan broj, $a > 1$ i neka je p najmanji djelitelj od a koji je veći od 1. Tada je p prost broj.

DOKAZ. Pretpostavimo suprotno, tj. da je p složen broj. Znači da vrijedi: $p = qr$, gdje su $q, r > 1$. Tada su i q i r djelitelji broja a koji su manji od p. To je kontradikcija s minimalnošću djelitelja p. Q.E.D.

DEFINICIJA. Neka su a i b prirodni brojevi. Kažemo da a dijeli b ako je b višekratnik od a, tj. postoji $k \in \boldsymbol{N}$ tako da je $b = ka$. U tom slučaju pišemo $a \mid b$ i čitamo $''a$ dijeli b''. Broj a zovemo djeliteljem broja b.

DEFINICIJA. Ako su a, b, d prirodni brojevi takvi da vrijedi $d \mid a$ i $d \mid b$, onda d zovemo zajedničkim djeliteljem od a i b. Ako je broj d najveći zajednički djelitelj onda ga zovemo najvećom zajedničkom mjerom od a i b. Označavamo ga s $\mathrm{Nzm}\,(a, b)$.

LEMA 2. Za svaki prost broj p je ili $\mathrm{Nzm}\,(p, a) = 1$ ili $p \mid a$.

DOKAZ. Jedini djelitelji broja p su 1 ili p. Zato je ili $\mathrm{Nzm}\,(p, a) = 1$ ili $\mathrm{Nzm}\,(p, a) = p$. U drugom slučaju $p \mid a$. Q.E.D.

TEOREM. (EUKLIDOVA LEMA) Ako je p prost broj i dijeli umnožak ab, tada p dijeli a ili p dijeli b (ili oboje).

DOKAZ. Neka je p prost broj i neka $p \mid ab$. Pretpostavimo da p ne dijeli a. Tada trebamo dokazati da p dijeli b. Budući da p dijeli ab, postoji cijeli broj k takav da je $ab = kp$. Broj p ne dijeli a, pa mora biti $(p, a) = 1$, jer je 1 jedini mogući faktor broja p. Zato postoje cijeli brojevi x i y takvi da je $xa + yp = 1$. Stoga imamo $b = b(xa) + b(yp) = xkp + ybp = (xk + yb)p$ iz čega je vidljivo da je b djeljiv s p. Da teorem vrijedi samo za proste brojeve možemo dokazati i sljedećim primjerom: broj 6 dijeli umnožak $3 \cdot 4$, ali 6 ne dijeli 3 i 6 ne dijeli 4.
Time je teorem dokazan. Q.E.D.

KOROLAR. Ako je p prost broj i $p \mid a_1 a_2 \ldots a_n$, tada postoji barem jedan a_i takav da $p \mid a_i$.

Slijedi jedan od važnijih rezultata teorije brojeva, tzv. Osnovni teorem aritmetike, čiji se dokaz također pripisuje starogrčkom matematičaru Euklidu.

TEOREM. (OSNOVNI TEOREM ARITMETIKE) Za svaki prirodni broj $a > 1$ postoji jedinstven rastav na proste djelitelje $a = {p_1}^{\alpha_1} {p_2}^{\alpha_2} \ldots {p_k}^{\alpha_k}$ pri čemu su $p_1 < p_2 < \ldots < p_k$ svi

različiti prosti faktori broja a te $\alpha_1, \alpha_2, \ldots, \alpha_k$ prirodni brojevi. Pritom α_i zovemo kratnošću odgovarajućeg prostog broja p_i u rastavu.

DOKAZ. Teorem se sastoji od dvije tvrdnje: jedna je da se svaki prirodni broj može rastaviti na proste djelitelje, odnosno da se može prikazati kao umnožak prostih faktora, a druga tvrdnja je da je taj rastav jedinstven. Prvo dokažimo spomenutu prvu tvrdnju. Neka je q_1 najmanji prosti faktor broja a, odnosno $a = q_1 a_1$. Ako je $a_1 > 1$ i q_2 najmanji prosti faktor broja a_1, onda je $a_1 = q_2 a_2$, tj. $a = q_1 q_2 a_2$. Nastavljajući dalje vidimo da će zbog $a > a_1 > a_2 > \ldots$ nakon konačno mnogo koraka broj a biti rastavljen na proste faktore: $a = q_1 q_2 \ldots q_n$. Neki od prostih brojeva q_k mogu biti i jednaki. Nakon grupiranja po veličini dobivamo rastav kao u teoremu. Dokažimo sada drugu tvrdnju teorema, tj. dokažimo da je rastav broja a na proste faktore jedinstven (do na njihov poredak). Pretpostavimo da imamo još jedan rastav na m prostih faktora: $a = r_1 r_2 \ldots r_m$, tj.: $q_1 q_2 \ldots q_n = r_1 r_2 \ldots r_m$. Kako q_1 dijeli lijevu stranu, on dijeli i desnu, pa prema prethodnom korolaru on dijeli neki od r-ova, npr. za r_1 vrijedi $q_1 | r_1$. Broj r_1 je prost broj i $q_1 > 1$, pa slijedi da je $q_1 = r_1$. Podijelimo jednakost s q_1 s lijeve strane i s desne strane dobijemo $q_2 \ldots q_n = r_2 \ldots r_m$. Nastavimo na isti način s brojem q_2. Nakon konačno mnogo koraka dobivamo da mora biti $n = m$ i $q_i = r_i$ za svaki i.
Time je teorem dokazan. Q.E.D.

KOROLAR. Djelitelja prirodnog broja $a = {p_1}^{\alpha_1} {p_2}^{\alpha_2} \ldots {p_k}^{\alpha_k}$ rastavljenog na proste faktore $p_1 < p_2 < \ldots < p_k$, ima ukupno $(\alpha_1 + 1)(\alpha_2 + 1) \ldots (\alpha_k + 1)$.

DOKAZ. Svaki djelitelj d od broja a ima oblik $d = {p_1}^{\beta_1} {p_2}^{\beta_2} \ldots {p_k}^{\beta_k}$ gdje je $\beta_i \in \{0, 1, \ldots, \alpha_i\}$. Dakle, broj djelitelja broja a jednak je broju poredanih n-torki $(\beta_1, \beta_2, \ldots, \beta_n)$ gdje svaki β_i može poprimiti bilo koju od $\alpha_i + 1$ vrijednosti. Njihov ukupan broj je $(\alpha_1 + 1)(\alpha_2 + 1) \ldots (\alpha_k + 1)$. Q.E.D.

Nekoliko direktnih posljedice prethodnog korolara su:

- Broj djelitelja od n je dva ako i samo ako je n prost.
- Broj djelitelja od n je tri ako i samo ako je n oblika $n = p^2$, p prost.
- Broj koji ima točno deset djelitelja je broj oblika $n = {p_1}^4 \cdot p_2$, ili pak $n = {p_1}^9$. Najmanji takav broj je $n = 2^4 \cdot 3 = 48$

PRIMJER. Nađimo broj djelitelja broja 60, koristeći prethodni korolar. To ćemo napraviti na način da broj rastavimo na proste djelitelje: $60 = 2 \cdot 30 = 2 \cdot 2 \cdot 15 = 2 \cdot 2 \cdot 3 \cdot 5 = 2^2 \cdot 3 \cdot 5$. Prema prethodnom korolaru, broj 60 ima ukupno $(2+1)(1+1)(1+1) = 12$. Nije velik broj djelitelja pa ih možemo i ispisati, to su brojevi: $1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 10,\ 12,\ 15,\ 20,\ 30,\ 60$. Stari narodi su znali da broj 60 ima razmjerno velik broj različitih djelitelja i zbog toga su koristili bazu 60 za računanje, to je tzv. seksagezimalni sustav, od latinske riječi *seksagezimus* što u prijevodu znači šezdeset. Najstarija poznata egipatska matematička knjiga je ″Ahmesova računica″ (nastala oko 1700. g. pr. Kr.) u kojoj je prikazano računanje u bazi 60. I mi smo naslijedili računanje s brojem 60 : jedan sat ima 60 minuta, minuta ima 60 sekundi, puni kut ima $60° \cdot 6 = 360°$, jednakostraničan trokut ima kutove od $60°$ itd.

Euklid je u svojim, već spomenutim, ″Elementima″ dokazao i da je skup prostih brojeva beskonačan. Na početku je istaknuto da prirodnih brojeva ima beskonačno mnogo, a jasno je, u skladu pojma beskonačnosti, da i parnih, kao i neparnih brojeva također ima beskonačno mnogo. No, manje je jasno ima li i prostih brojeva beskonačno mnogo. Euklid je dokazao da i prostih brojeva ima beskonačno mnogo.

2.2. Gustoća prostih brojeva

Ne postoji pravilo po kojem možemo odrediti kada će se u zadanom nizu pojaviti prosti broj. Na primjer između brojeva 1 i 10 pronalazimo četiri prosta broja, dok između brojeva 80 i 90 postoje samo dva prosta broja, a između 2000 i 2100 postoji jedan prost broj.

Dokazano je da prostih brojeva ima beskonačno mnogo, ali nije otkrivena formula za pronalaženje prostih brojeva pa se počelo empirijski istraživati udio prostih brojeva u skupu svih prirodnih brojeva. Ako P_n označava broj svih prostih brojeva koji nisu veći od n. Udio prostih brojeva u skupu svih prirodnih brojeva koji nisu veći od n pokazuje kvocijent $Q_n = \frac{P_n}{n}$.

Naslućujemo da se povećanjem broja n udio prostih brojeva sve više smajuje. Mnogi su se stari matematičari, počevši od Gaussa, trudili istraživati zakonitost ponašanja kvocijenata Q_n, što im je konačno i uspjelo potkraj 19. stoljeća, kada su dokazali da se za velike n kvocijent Q_n ponaša kao $\frac{1}{\log n}$. U nastavku će to biti objašnjeno kroz teoreme.

TEOREM. Skup svih prostih brojeva je beskonačan.

DOKAZ. Pretpostavimo suprotno, tj. da je skup svih prostih brojeva konačan. Označimo taj skup s $P = \{p_1,\ p_2, \ldots, p_k\}$. Promotrimo broj $a = p_1 p_2 \ldots p_k + 1$. Broj a nije djeljiv ni s jednim od brojeva $p_1,\ p_2, \ldots, p_k$, jer je ostatak pri dijeljenju uvijek jednak 1. To se protivi polaznoj pretpostavci i zaključujemo da je skup svih prostih brojeva beskonačan. Q.E.D.

Postoje stotine i tisuće uzastopnih prirodnih brojeva među kojima nema niti jednog prostog broja. Sljedeći teorem govori o tome.

TEOREM. Za svaki prirodni broj *n* postoji niz od *n* uzastopnih prirodnih brojeva od kojih ni jedan nije prost.

DOKAZ. Neka je *n* bilo koji prirodni broj veći od 1. Promotrimo brojeve: $2 + n!,\ 3 + n!,\ \ldots,\ n + n!$. Ima ih $n - 1$ i slijede jedan za drugim, tj. svaki je naredni za 1 veći od prethodnog broja. Zaključujemo da vrijede jednakosti: $2 + n! = 2 + 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n = 2(1 + 3 \cdot 4 \cdot \ldots \cdot n)$, $3 + n! = 3 + 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n = 3(1 + 2 \cdot 4 \cdot \ldots \cdot n)$ i tako dalje do $n + n! = n + 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n = n[1 + 2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-1)]$. Ti su brojevi složeni jer je prvi djeljiv s 2, drugi s 3 i tako redom do zadnjeg koji je djeljiv s n. Q.E.D.

TEOREM. (ČEBIŠEVLJEV) Između bilo kojeg broja n i broja $2n$, gdje je n veći od 5, nalaze se barem dva prosta broja.

Kada provjeravamo je li određeni broj prost ili je složen nije potrebno tražiti sve djelitelje tog broja. To možemo nazvati i kriterijem za pronalaženje prostog broja, a izreći ćemo ga sljedećim teoremom.

TEOREM. Složen broj n ima barem jedan prost faktor koji je manji ili jednak od $\sqrt{n}$.

DOKAZ. Neka je n složen broj. Tada ga možemo pisati u obliku $n = p_1 a$, gdje je p_1 prost broj, a broj a je složen broj koji se opet može napisati kao produkt prostih brojeva. Pretpostavimo suprotno tvrdnji teorema, tj. pretpostavimo da su svi prosti faktori veći od $\sqrt{n}$. Tada je $p_1 > \sqrt{n}$ i $a > \sqrt{n}$, jer je svaki faktor od broja a veći od broja $\sqrt{n}$. Tada je $p_1 a > \sqrt{n}\sqrt{n} = n$, a to je kontradikcija naše pretpostavke. Q.E.D.

Odavno je poznato da prostih brojeva ima beskonačno mnogo, ali nije otkrivena formula za pronalaženje prostih brojeva pa se počelo empirijski istraživati udio prostih brojeva u skupu svih

prirodnih brojeva. Ako P_n označava broj svih prostih brojeva koji nisu veći od n, onda je: $P_1 = 0$, $P_2 = 1,\ P_3 = P_4 = 2,\ P_5 = P_6 = 3,\ P_7 = P_8 = P_9 = P_{10} = 4,\ P_{11} = P_{12} = 5 \quad P_{13} = P_{14} = P_{15} = P_{16} = 7,\ P_{17} = P_{18} = 8, \ldots$ Na primjer $P_{12} = 5$ znači da ima 5 prostih brojeva koji su manji od 12.

Udio prostih brojeva u skupu svih prirodnih brojeva koji nisu veći od n pokazuje kvocijent $Q_n = \frac{P_n}{n}$. Tako je: $Q_1 = 0,\ Q_2 = \frac{1}{2} = 0.5,\ Q_5 = \frac{3}{5} = 0.6,\ Q_{10} = \frac{4}{10} = 0.4, \ldots$

Također je: $Q_{1000} = 0.168,\ Q_{1000000} = 0.078498$. Naslućujemo da se povećanjem broja n udio prostih brojeva sve više smajuje. Mnogi su se stari matematičari trudili istraživati zakonitost ponašanja kvocijenata Q_n. C. F. Gauss je u dobi od petnaest godina primijetio da se za velike n kvocijent Q_n ponaša kao $\frac{1}{\ln n}$, što je n veći to je aproksimacija bolja. To objašnjavamo i kroz teorem o prostim brojevima.

TEOREM. (O PROSTIM BROJEVIMA) $\frac{P_n}{n} \sim \frac{1}{\ln n}$, odnosno $\lim_{n\to\infty} \frac{\frac{P_n}{n}}{\frac{1}{\ln n}} = 1$.

Strogi dokaz ovog teorema izveli su neovisno svaki za sebe Charles Jean Gustave Nicolas de la Vallée Paussin (1866.-1962.) i Jacques Salomon Hadamard (1865.-1963.) 1896. godine.

3. BRZI ALGORITMI ZA TRAŽENJE PROSTIH BROJEVA

3.1. Osnovni algoritam

Algoritam koji na najosnovniji način traži proste brojeve. Na početku algoritma se upiše gornja granica intervala, tj. prirodni broj $n > 2$ do kojeg se žele tražiti prosti brojevi. Algoritam radi na principu da provjerava djeljivost svakog kandidata, broja a, gdje je $2 < a < n$ u intervalu od broja 2 do broja $\sqrt{a}$, gdje je n broj koji je zadan na početku (gornja granica intervala). Ne primjenjuje ni jedno posebno pravilo djeljivosti brojeva. Npr. prvo provjeri je li broj 3 djeljiv s nekim od svojih prethodnika (osim broja 1 i samim sobom), pošto nije upisuje ga na listu prostih brojeva. Zatim provjerava je li broj 4 djeljiv s nekim od svojih prethodnika čiji je kvadrat manji od broja 4, dobije odgovor da je djeljiv s brojem 2 i zbog toga ga ne stavlja na listu prostih brojeva. Isti postupak se provjerava do željenog broja n. Treba napomenuti da ne traži sve djelitelje promatranog kandidata nego ide do onog djelitelja čiji je kvadrat manji ili jednak od kandidata. Npr. ako za kandidata uzmemo broj 50, algoritam će provjeravati je li broj 50 djeljiv s brojevima 2, 3, 4, 5, 6 i 7. Neće provjeravati je li djelitelj veći od 7 zato što je $7^2 \leq 50$ (dok je sljedeći potencijalni djelitelj, broj 8, a za njega vrijedi $8^2 > 50$). Algoritam naravno ne provjerava je li kandidat djeljiv sa svim mogućim djeliteljima nego se prekida čim nađe prvog djelitelja. Tako će u ovom primjeru algoritam prekinuti s radom kada ustanovi da je 2 djelitelj zadanog kandidata 50, tog kandidata neće uvrstiti na popis nego će nastaviti tražiti nove kandidate u zadanom intervalu.

Algoritam 1. Osnovni algoritam, ulaz je granica intervala *n*, a izlaz je popis prostih brojeva

```
granica ← n
kandidat ← 2
popis_prostih_brojeva ← Ø
sve dok kandidat ≤ granica
    djelitelj ← 2
    prosti_broj ← 1
    sve dok djelitelj² ≤ kandidat
        ako je kandidat % djelitelj = 0
            prosti_broj ← 0
            prekini sve dok
        djelitelj ← djelitelj + 1
```

```
    ako je prosti_broj = 1
        popis_prostih_brojeva dodaj kandidat
    kandidat ← kandidat +1
```

3.2 Nadogradnja osnovnog algoritma

3.2.1. Prva nadogradnja

Ovaj algoritam je modifikacija prethodnog algoritma. Na početku algoritma se upiše gornja granica intervala, tj. prirodni broj $n > 3$ do kojeg se žele tražiti prosti brojevi. Algoritam za djelitelje uzima samo proste brojeve koje pronalazi na popisu prethodno pronađenih prostih brojeva i to dok zadovoljavaju uvjet da je kvadrat djelitelja manji ili jednak od samog kandidata. Na početku algoritma na popisu prostih brojeva je već upisan broj 2.

Algoritam 2. Prva nadogradnja osnovnog algoritma, ulaz je granica intervala *n*, a izlaz je popis prostih brojeva

```
granica ← n
kandidat ← 3
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
sve dok kandidat ≤ granica
    indeks_djelitelja ← 0
    djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    prosti_broj ← 1
    sve dok djelitelj^2 ≤ kandidat
        ako je kandidat % djelitelj = 0
            prosti_broj ← 0
            prekini sve dok
        indeks_djelitelja ← indeks_djelitelja + 1
        djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    ako je prosti_broj = 1
```

```
        popis_prostih_brojeva dodaj kandidat
    kandidat ← kandidat +1
```

3.2.2. Druga nadogradnja

Algoritam se može dodatno ubrzati ako provjeravamo samo neparne brojeve, tj. krene od broja 3 i provjerava svaki drugi broj. Takav algoritam se od prethodnog algoritma razlikuje samo u zadnjoj liniji pseudo koda gdje bi se umjesto povećavanja kandidata za 1 napravilo povećavanje kandidata za 2.

Algoritam 3. Druga nadogradnja osnovnog algoritma, ulaz je granica intervala *n*, a izlaz je popis prostih brojeva

```
granica ← n
kandidat ← 3
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
sve dok kandidat ≤ granica
    indeks_djelitelja ← 0
    djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    prosti_broj ← 1
    sve dok djelitelj^2 ≤ kandidat
        ako je kandidat % djelitelj = 0
            prosti_broj ← 0
            prekini sve dok
        indeks_djelitelja ← indeks_djelitelja + 1
        djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    ako je prosti_broj = 1
        popis_prostih_brojeva dodaj kandidat
    kandidat ← kandidat +2
```

3.2.3. Treća nadogradnja

Algoritam se dodatno može nadograditi time da se uz višekratnike broja 2 izbace i višekratnici broja 3. Zbog toga nije potrebno provjeravati svakog kandidata nego se krenuvši od broja 5 provjerava svaki drugi odnosno četvrti kandidat naizmjence.

Algoritam 4. Treća nadogradnja osnovnog algoritma, ulaz je granica intervala *n*, a izlaz je popis prostih brojeva

```
granica ← n
kandidat ← 5
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
popis_prostih_brojeva dodaj 3
pomak ← 2
sve dok kandidat ≤ granica
    indeks_djelitelja ← 0
    djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    prosti_broj ← 1
    sve dok djelitelj² ≤ kandidat
        ako je kandidat % djelitelj = 0
            prosti_broj ← 0
            prekini sve dok
        indeks_djelitelja ← indeks_djelitelja + 1
        djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
    ako je prosti_broj = 1
        popis_prostih_brojeva dodaj kandidat
    kandidat ← kandidat + pomak
    ako je pomak = 2
        pomak ← 4
    inače
        pomak ← 2
```

3.2.4. Četvrta nadogradnja

Razmišljajući o nadogradnjama algoritama, može se doći do zaključka da se uz parne brojeve na vrlo jednostavan način mogu izbaciti i brojevi djeljivi s brojem 5. Stoga, sljedeća implementacija algoritma kreće od provjere zadnje znamenke. Ukoliko je zadnja znamenka potencijalnog kandidata broj 1, 3, 7 ili 9 kandidat ostaje u igri, u suprotnom kandidat nije prost broj. Provjerava se je li kandidat djeljiv s brojem 3, brojem 7 i zatim svakim sljedećim prostim brojem. Na listu prostih brojeva se upišu brojevi 2, 3, 5 i 7 te ostali kandidati koji prođu test djeljivosti.

Algoritam 5. Četvrta nadogradnja osnovnog algoritma, ulaz je granica intervala *n*, a izlaz je popis prostih brojeva

```
granica ← n
kandidat ← 11
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 3
popis_prostih_brojeva dodaj 7
sve dok kandidat ≤ granica
   zadnja_znamenka ← kandidat % 10
   ako je zadnja_znamenka = 1 ili zadnja_znamenka = 3 ili zadnja_znamenka = 7 ili
                                                              zadnja_znamenka
= 9
      indeks_djelitelja ← 0
      djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
      prosti_broj ← 1
      sve dok djelitelj^2 ≤ kandidat
         ako je kandidat % djelitelj = 0
            prosti_broj ← 0
            prekini sve dok
         indeks_djelitelja ← indeks_djelitelja + 1
         djelitelj ← popis_prostih_brojeva na indeksu indeks_djelitelja
      ako je prosti_broj = 1
         popis_prostih_brojeva dodaj kandidat
   kandidat ← kandidat + 1
```

3.3 Sita

3.3.1. Eratostenovo sito

Eratostenovo sito potječe od grčkog matematičara i geografa Eratostena (276.-194.g. prije Krista). Eratosten se obrazovao u Ateni i Aleksandriji, a radio je u Aleksandriji. Poznat je ponajviše po mjerenju opsega Zemlje, a nama je zanimljiv jer je pronašao način za pronalaženje prostih brojeva. Eratosten nije pronašao formulu za traženje prostih brojeva, ali je otkrio postupak kojim se načelno mogu dobiti svi prosti brojevi. U nizu prostih brojeva najprije se izostavi broj 1 koji nije prost broj, zatim se iz preostalih brojeva izbace svi parni brojevi osim broja 2 koji je prost broj. Sljedeći prvi broj među preostalima je prost broj 3, pa se zatim iz preostalih brojeva izbace svi složeni brojevi koji sadrže broj 3 kao faktor, tj. izbacuje se svaki treći broj (ako već nije izbačen) u nizu prorodnih brojeva. Sljedeći najmanji preostali prost broj je broj 5. Stoga se izbacuju svi višekratnici broja 5, tj. izbacuju se svaki peti broj (ako već nije izbačen) u nizu prostih brojeva. Isti postupak nastavljamo i na taj način dobivamo sve veće i veće proste brojeve.

Algoritam 6. Eratostenovo sito

```
granica ← n
trenutni ← 2
popis_prostih_brojeva ← Ø
P ← 0
P0 ← 1
P1 ← 1
sve dok trenutni ≤ granica
    označi ← trenutni + trenutni
    sve dok označi ≤ granica
        Poznači ← 1
        označi ← označi + trenutni
    trenutni ← trenutni + 1
    sve dok  Ptrenutni = 1
        trenutni ← trenutni + 1
i ← 0
sve dok i ≤ granica
    ako je Pi = 0
```

```
        popis_prostih_brojeva dodaj i
    i ← i + 1
```

3.3.2. Optimizirano Eratostenovo sito

Optimizirano Eratostenovo sito je nadogradnja prethodno opisanog Eratostenovog sita. Ne provjeravaju se svi brojevi na popisu (dokle god ih ima), nego se traže višekratnici od najmanjeg djelitelja, broja 2, do djelitelja čiji je kvadrat manji ili jednak od zadnjeg broja na popisu, tzv. granice, onog broja koji je gornja granica intervala unutar kojeg pretražujemo proste brojeve.

Algoritam 7. Eratostenovo sito

```
granica ← n
trenutni ← 2
popis_prostih_brojeva ← Ø
P ← 0 (polje dimenzije n+1)
P0 ← 1
P1 ← 1
pKvadrat ← 0
tKvadrat ← 0
sve dok trenutni^2 ≤ granica
    označi ← trenutni + trenutni
    sve dok označi ≤ granica
        P_označi ← 1
        označi ← označi + trenutni
    tKvadrat ← trenutni^2
    sve dok pKvadrat < tKvadrat
        ako je P_pKvadrat = 0
            popis_prostih_brojeva dodaj pKvadrat
        pKvadrat ← pKvadrat + 1
    trenutni ← trenutni + 1
    sve dok P_trenutni = 1
        trenutni ← trenutni + 1
sve dok pKvadrat ≤ granica
```

```
    ako je P_pKvadrat = 0
        popis_prostih_brojeva dodaj pKvadrat
    pKvadrat ← pKvadrat + 1
```

3.3.3. Sundramovo sito

Prvo se ispišu svi brojevi od 1 do zadanog broja, koji će biti gornja granica intervala u kojem će se pronalaziti prosti brojevi. S popisa se uklanjaju brojevi oblika $i + j(2i + 1)$, gdje su i i j prirodni brojevi za koje vrijedi $1 \leq i \leq j$ te $i + j(2i + 1) \leq n$. Brojevi koji nisu uklonjeni s popisa su prosti brojevi.

Algoritam 8. Sundramovo sito

```
granica ← n
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
P ← 0 (polje dimenzije n+1)
pola ← granica / 2
maxVrijednost ← 0
i ← 1
sve dok i < pola
    djelitelj ← i*2+1
    maxVrijednost ← (pola – i) / djelitelj
    j ← i
    sve dok j ≤ maxVrijednost
        P_i+j*djelitelj ← 1
        j ← j+1
    i ← i+1
i ← 1
sve dok i < pola
    ako je P_i = 0
        popis_prostih_brojeva dodaj i*2+1
    i ← i+1
```

3.3.4. Atkinovo sito

Atkinovo sito su osmislili A.O.L. Atkin i D.J. Bernstejn 2003. godine. U algoritmu se traže ostaci dijeljenja sa 60, time se izbacuju brojevi djeljivi s 2, 3 i 5. Promatra se ostatak s modulo šezdeset na temelju kojega se procjenjuje je li broj prost.

Algoritam 9. Atkinovo sito

```
granica ← n
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
popis_prostih_brojeva dodaj 3
P ← 0 (polje dimenzije n+1)
korijen ← √granica
x ← 1
xKorak ← 3
y ← 1
yKorak ← 3
i ← 1
sve dok i ≤ korijen
   y ← 1
   yKorak ← 3
   j ← 1
   sve dok j ≤ korijen
      v ← 4*x + y
      ako je v ≤ granica i (v % 12 = 1 ili v % 12 = 5)
         Pv ← Pv^c
      v ← v – x
      ako je v ≤ granica i v % 12 = 7
         Pv ← Pv^c
      ako je i > j
         v ← v – 2*y
            ako je v ≤ granica i v % 12 = 11
               Pv ← Pv^c
```

```
        y ← y + yKorak
        yKorak ← yKorak + 2
        j ← j + 1
    x ← x + xKorak
    xKorak ← xKorak + 2
  i ← i + 1
g ← 5
sve dok g ≤ korijen
  ako je P_g = 1
      k ← g^2
      z ← k
      sve dok z ≤ granica
          P_z ← 0
          z ← z + k
  g ← g + 1
i ← 1
sve dok i < granica
  ako je P_i = 1
      popis_prostih_brojeva dodaj i
  i ← i+1
```

3.3.5. Optimizirano Atkinovo sito

Dorađeno originalno Atkinovo sito, radi na istom principu ali optimizirano.

Algoritam 10. Optimirano Atkinovo sito

```
granica ← n
popis_prostih_brojeva ← Ø
popis_prostih_brojeva dodaj 2
popis_prostih_brojeva dodaj 3
P ← 0 (polje dimenzije n+1)
korijen ← √granica
```

xKorak ← *3*

$g \leftarrow \sqrt{(granica - 1)/3}$

i ← *0*

sve dok *i* < *12* * *g*

xKorak ← *xKorak* + *i*

yGranica ← $12 * \sqrt{granica - xKorak} - 36$

x ←*xKorak* + *16*

j ← *-12*

sve dok *j* < *yGranica* + *1*

x ← *x* + *j*

$\boldsymbol{P_x} \leftarrow \boldsymbol{P_x^c}$

j ← *j* + *72*

x ←*xKorak* + *4*

j ← *12*

sve dok *j* < *yGranica* + *1*

x ← *x* + *j*

$\boldsymbol{P_x} \leftarrow \boldsymbol{P_x^c}$

j ← *j* + *72*

i ← *i* + *24*

xKorak ← *0*

$g \leftarrow 8 * \sqrt{(granica - 1)/4} + 4$

i ← *4*

sve dok *i* < *g*

xKorak ← *xKorak* + *i*

x ←*xKorak* + *1*

ako je *xKorak* % *3* ≠ *0*

gg ← $4 * \sqrt{granica - xKorak} - 3$

j ← *0*

sve dok *j* < *gg*

x ← *x* + *j*

$\boldsymbol{P_x} \leftarrow \boldsymbol{P_x^c}$

j ← *j* + *8*

inače

$yGranica \leftarrow 12 * \sqrt{granica - xKorak} - 36$

$x \leftarrow xKorak + 25$

$j \leftarrow -24$

sve dok $j < yGranica + 1$

$x \leftarrow x + j$

$P_x \leftarrow P_x^c$

$j \leftarrow j + 72$

$x \leftarrow xKorak + 2$

$j \leftarrow 24$

sve dok $j < yGranica + 1$

$x \leftarrow x + j$

$P_x \leftarrow P_x^c$

$j \leftarrow j + 72$

$i \leftarrow i + 8$

$xKorak \leftarrow 1$

$g \leftarrow \sqrt{granica/2} + 1$

$i \leftarrow 3$

sve dok $i < 12 * g$

$xKorak \leftarrow xKorak + 4 * i - 4$

$x \leftarrow 3 * xKorak$

$s \leftarrow 4$

ako je $x > granica$

$min\_y \leftarrow \sqrt{(x - granica)/4} * 4$

$x \leftarrow x - min\_y^2$

$s \leftarrow 4 * min\_y + 4$

$j \leftarrow s$

sve dok $j < 4 * i$

$x \leftarrow x - j$

ako je $x \leq granica$ ***i*** $x \% 12 = 11$

$P_x \leftarrow P_x^c$

$j \leftarrow j + 8$

```
    i ← i + 2

xKorak ← 0
i ← 2
sve dok i < 12 * g
    xKorak ← xKorak +4 *  i - 4
    x ←3 * xKorak
   ako je x > granica
        min_y ← √((x − granica)/4) ∗ 4 − 1
        x ← x - min_y^2
        s ← 4* min_y + 4
    inače
        s ← 0
        x ← x - 1
    j ← s
    sve dok j < 4*i
        x ← x – j
        ako je x ≤ granica i x % 12 = 11
            P_x ← P_x^c
        j ← j + 8
    i ← i + 2

i ← 5
sve dok i < korijen + 1
    ako je P_i = 1
        k ← i^2
        z ← k
        sve dok z < granica
            P_z ← 0
            z ← z + k
    i ← i + 2
i ← 5
sve dok i < granica
```

```
ako je Pi = 1
    popis_prostih_brojeva dodaj i
i ← i+2
```

3.4. Usporedba algoritama

Provedeno je ispitivanje pronalaska prostih brojeva do prirodnog broja 10 000 000. Na tom intervalu ima ukupno 664 579 prostih brojeva. Eksperiment je proveden 100 puta za svaki implementirani algoritam zbog veće stabilnosti rezultata. Prosječni rezultati vremena su prikazani u tablici 1.

Tablica 1. Prikaz vremena potrebnog za izvođenje pojedinog algoritma

Naziv algoritma	Vrijeme potrebno za izvođenje algoritma $[ms]$
Osnovni algoritam	14785.3000
Prva nadogradnja osnovnog algoritma	2754.4800
Druga nadogradnja osnovnog algoritma	2689.3400
Treća nadogradnja osnovnog algoritma	2590.4900
Četvrta nadogradnja osnovnog algoritma	2605.6100
Eratostenovo sito	136.5180
Optimizirano Eratostenovo sito	93.2058
Sundramovo sito	65.6986
Atkinovo sito	101.2530
Optimizirano Atkinovo sito	51.6565

Sva mjerenja su napravljena na PC računalu s Intel Core i7-4710HQ procesorom i 8GB RAM memorije te Windows 8.1 x64 operacijskim sustavom.

Na temelju vremena potrebnog za izvođenje pojedinog algoritma, koja promatramo u prethodnoj tablici, zaključujemo da je Eratostenovo sito čak 108 puta brže od osnovnog algoritma. Osnovni

algoritam provjera djeljivost promatranog broja sa svim prethodnicima za što je potrebno puno vremena pa je i razumljivo da je najsporiji. Već je prva nadogradnja osnovnog algoritma brža više od 5 puta od osnovnog algoritma. U prvoj nadogradnji se provjerava djeljivost promatranog broja s prethodno pronađenim prostim brojevima. Nadolazeće nadogradnje ne pokazuju posebno poboljšanje. Druga nadogradnja je za 65.14 *ms* brža od prve nadogradnje jer na početku iz provjere izbacuje parne brojeve, tj. traži proste brojeve među neparnim brojevima. Treća nadogradnja je za 98.85 *ms* brža od druge jer na početku izvođenja algoritma iz provjere izbacuje višekratnike boja 2 i višekratnike broje 3, tj. krenuvši od broja 5 provjerava svakog drugog odnosno četvrtog kandidata naizmjence. Četvrta nadogradnja, osim višekratnika brojeva 2 i 3, izbacuje iz provjere i višekratnike broja 5 (provjerava završava li promatrani broj znamenkom 0 ili 5). No, za to joj treba više vremena, točno 15.12 *ms* više nego trećoj nadogradnji. Sita se pokazuju kao najbolje rješenje za traženje prostih brojeva. Eratostenovo sito je najprimitivnije od promatranih sita, a ono treba skoro 19 puta manje vremena za svoje izvođenje od najboljeg osnovnog algoritma, tj. treće nadogradnje osnovnog algoritma. Optimizirano Eratostenovo sito je skoro 1.5 puta brže od početnog Eratostenovog sita. Sundramovom situ treba približno 1.4 puta manje vremena od optimiziranog Eratostenovog sita. Atkinovo sito je brže od Eratostenovog sita, ali je sporije od optimiziranog Eratostenovog sita i od Sundramovog sita. U tablici je na zadnjem mjestu najbolje sito, tj. sito kojem treba najmanje vremena za pronalazak prostih brojeva, a to je optimizirano Atkinovo sito. Atkinovo sito je sito koje traži ostatke dijeljenja s brojem 60, time izbacuje višekratnike brojeva 2, 3 i 5 te na temelju ostataka procjenjuje je li broj prost ili nije. Optimizirano Atkinovo sito odradi zadani posao za 51.6565 *ms*, što je čak 286 puta brže od osnovnog algoritma, približno 50 puta brže od najbržeg osnovnog algoritma, a 1.3 puta brže od sljedećeg najboljeg Sundramovog sita.

4. ZAKLJUČAK

Prosti brojevi su kroz povijest imali veliko značenje, tražilo se pravilo po kojem bi se mogli pronaći prosti brojevi. Danas postoji i projekt koji nudi novčane nagrade za pronalazak najvećeg prostog broja. Primjena prostih brojeva je mnogobrojna, a možda je najznačajnija primjena u kriptografiji u kojoj se koriste veliki prosti brojevi za koje se ne može na jednostavan način provjeriti njihova prostost. U ovom radu su ispitane brzine danih algoritama: osnovnog algoritma i četiri njegove nadogradnje te Eratostenovo, Sundramovo i Atkinovo sito uz optimizaciju Eratostenovog i Atkinovog sita. Testiranje se vršilo na intervalu do broja 10 000 000, tj. pitanje je bilo: koliko ima prostih brojeva u intervalu $\langle 1, 10\ 000\ 000 \rangle$? U navedenom intervalu ima 664 579 prostih brojeva. Svaki spomenuti algoritam je testiran na način da je provjeren broj prostih brojeva koje je našao u zadanom intervalu i bilježeno je vrijeme potrebno za rješavanje zadatka. Eksperiment je proveden 100 puta za svaki implementirani algoritam, a prosječna vremena izvođenja svakog algoritma prikazana su tablično. Zaključujemo da najviše vremena treba osnovnom algoritmu jer ima jako puno provjera. Mnoge nadogradnje osnovnog algoritma trebaju skoro šest puta manje vremena. Sita su se pokazala najboljima. Naime, najsporijem situ treba skoro 19 puta manje vremena od najbržeg osnovnog algoritma, dok najbržem situ treba čak 50 puta manje vremena od najbržeg osnovnog algoritma. Zaključujemo da u algoritmu treba paziti na pravila koja se žele zadovoljiti, ali i na ograničenja računala, kako bismo odabrali najbolji spoj koji ćemo iskombinirati i time dobiti algoritam koji zahtjeva najmanje vremena, a pri tome daje točne rezultate.

Literatura

[1] D. Žubrinić, Diskretna matematika, Element, Zagreb, 2001.

[2] N. Elezović, Diskontna matematika, Element, Zagreb, 2017.

[3] Ž. Pauše, Matematika i zdrav razum, Školska knjiga, Zagreb, 2007.

[4] R. Scitovski, K. Sabo, Prosti brojevi, Osječka matematička škola, br. 3, str. 13-20, 2003.

[5] The Prime Pages, http://primes.utm.edu/

Sažetak

U ovom radu su opisani i analizirani brzi algoritmi za pronalazak prostih brojeva. Definirani su prosti brojevi i objašnjena njihova matematička svojstva. Rad opisuje osnovni algoritam pronalaska prostih brojeva zasnovan na osnovnim pravilima djeljivosti. Nadalje, navedene su četiri njegove nadogradnje koje uvode dodatna pravila i mehanizme u implementaciji, čime se ubrzava rad osnovnog algoritma. Također su opisana i tri sita: Eratostenovo, Sundramovo i Atkinovo te nadogradnje za Eratostenovo i Atkinovo sito. Svi algoritmi su implementirani u C++ programskom jeziku te su analizirane njihove performanse. Analiza je pokazala da je optimirano Atkinovo sito najbrže.

Abstract

This thesis describes and analyses fast algorithms for prime number detection. Prime numbers are defined and their mathematical properties described. Basic algorithm based on fundamental division principle is described. Furthermore, four augmentations are also described, they introduce additional rules and mechanisms which results in increased performance. Three sieves, Eratosten´s, Sundram´s, Atkin´s are also described as well as enhancements to Eratosten´s and Atkin´s sieves. All algorithms were implemented in C++ programming language and their performance analyzed. Enhanced Atkin´s sieve has shown to be the fastest.

Životopis

Dubravka Kovač rođena je 22.09.1981. godine u Sremskoj Mitrovici. Osnovnu školu upisuje u Šidu. Nakon preseljenja, uzrokovanog ratnim okolnostima, osnovnu školu i osnovnu glazbenu školu Dore Pejačević završava u Našicama. Potom upisuje Prirodoslovno-matematičku gimnaziju Isidora Kršnjavoga u Našicama. Po završetku srednje škole, upisuje Sveučilišni dodiplomski studij na Sveučilištu J. J. Strossmayera, Odjel za matematiku, smjer matematika i fizika, koji završava i stječe akademski naziv profesor matematike i fizike. 2006. godine na Sveučilištu J. J. Strossmayera, Elektrotehnički fakultet, upisuje Sveučilišni preddiplomski studij elektrotehnike, nakon prve godine prebacuje se na Sveučilišni preddiplomski studij računarstva. 2010. godine završava sveučilišni preddiplomski studij računarstva te na istom fakultetu upisuje Sveučilišni diplomski studij računarstva, smjer Procesno računarstvo. Od 2014. godine je zaposlena u srednjoj školi kao profesorica matematike i fizike.

Prilog

Ovdje je prikazan izvorni kod korištenog programa za testiranje performansi implementiranih algoritama za pronalazak prostih brojeva.

```
// Prosti_brojevi.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"

#include <Windows.h>
#include <ctime>

#include <vector>
#include <set>
#include <iostream>
#include <math.h>

typedef unsigned long long velikiInt; //minimalno 64-bitni broj (pozitivni)

//Najobicnija, brutalna metoda pronalaska prostih brojeva //Obicna metoda
std::vector<velikiInt> Obicni(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
{
	std::vector<velikiInt> prosti_brojevi;
	velikiInt kandidat = 2;
	//vremenske varijable
	LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

	//Trazimo proste brojeve od 0 (2) do zadane granice
	velikiInt djeljitelj;
	bool prost_broj;
	while (kandidat <= gornja_granica)
	{
		djeljitelj = 2;
		prost_broj = true;
		//Za svaki kandidat provjeravamo djeljivost od 0 do djeljitelj^2 == kandidat
		while (djeljitelj * djeljitelj <= kandidat)
		{
			//Ako je djeljiv ostatak ce biti 0
			if (kandidat % djeljitelj == 0)
			{
				prost_broj = false;
				break;
			}
```

```
                djeljitelj++;
            }
            //ako je detektiran prost broj stavi ga na popis
            if (prost_broj)
                prosti_brojevi.push_back(kandidat);
            kandidat++;
        }

        //Prikazi vrijeme ako je zahtjevano
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
        QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
        vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
        if (prikazi_vrijeme)
            std::cout << "Obicni algoritam - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme << std::endl;

        //Vrati popis prostih brojeva
        return prosti_brojevi;
}

//Ovdje provjeravamo djeljitelje s popisa prethodno pronadenih prostih brojeva
std::vector<velikiInt> Obicni_nadogradnja_1(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
{
        std::vector<velikiInt> prosti_brojevi;
        velikiInt kandidat = 3;
        prosti_brojevi.push_back(2);
        //vremenske varijable
        LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

        //Trazimo proste brojeve od 0 (3) do zadane granice
        velikiInt djeljitelj;
        bool prost_broj;
        int indeks_djeljitelja;
        while (kandidat <= gornja_granica)
        {

            djeljitelj = prosti_brojevi.at(0);
            indeks_djeljitelja = 0;
            prost_broj = true;
            //Za svaki kandidat provjeravamo djeljitelje s popisa
            while (djeljitelj * djeljitelj <= kandidat)
            {
```

```
                        //Ako je djeljiv ostatak ce biti 0
                        if (kandidat % djeljitelj == 0)
                        {
                                prost_broj = false;
                                break;
                        }
                        //iduci djeljitelj na popisu
                        indeks_djeljitelja++;
                        if (indeks_djeljitelja >= prosti_brojevi.size())        //Ako je indeks veci nego je dopusteno, prekini
                                break;
                        djeljitelj = prosti_brojevi.at(indeks_djeljitelja);
                }
                //ako je detektiran prost broj stavi ga na popis prostih brojeva i djeljitelja
                if (prost_broj)
                        prosti_brojevi.push_back(kandidat);
                kandidat++;
        }

        //Prikazi vrijeme ako je zahtjevano
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
        QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
        vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
        if (prikazi_vrijeme)
                std::cout << "Obicni nadogradnja 1 - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme << std::endl;

        //Vrati popis prostih brojeva
        return prosti_brojevi;
}

//Isto kao i nadogradnja 1 samo ovdje gledamo samo neparne brojeve
std::vector<velikiInt> Obicni_nadogradnja_2(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
{
        std::vector<velikiInt> prosti_brojevi;
        prosti_brojevi.push_back(2); //prvi prosti broj poslije 1
        velikiInt kandidat = 3;         //krecemo od 3
        //vremenske varijable
        LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

        //Trazimo proste brojeve od 0 (2) do zadane granice
        velikiInt djeljitelj;
```

```
	bool prost_broj;
	int indeks_djeljitelja;
	while (kandidat <= gornja_granica)
	{
		djeljitelj = prosti_brojevi.at(0);
		indeks_djeljitelja = 0;
		prost_broj = true;
		//Za svaki kandidat provjeravamo djeljitelje s popisa
		while (djeljitelj * djeljitelj <= kandidat)
		{
			//Ako je djeljiv ostatak ce biti 0
			if (kandidat % djeljitelj == 0)
			{
				prost_broj = false;
				break;
			}
			//iduci djeljitelj na popisu
			indeks_djeljitelja++;
			if (indeks_djeljitelja >= prosti_brojevi.size())	//Ako je indeks veci nego je dopusteno, prekini
				break;
			djeljitelj = prosti_brojevi.at(indeks_djeljitelja);
		}
		//ako je detektiran prost broj stavi ga na popis prostih brojeva i djeljitelja
		if (prost_broj)
			prosti_brojevi.push_back(kandidat);
		kandidat += 2; //Svaki drugi broj povcevsi od 3 (neparni brojevi)
	}

	//Prikazi vrijeme ako je zahtjevano
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
	QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
	vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
	if (prikazi_vrijeme)
		std::cout << "Obicni nadogradnja 2 - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme << std::endl;

	//Vrati popis prostih brojeva
	return prosti_brojevi;
}

//Slicno kao i nadogradnja 2 samo ovdje elminirammo visekratnike broja 2 i 3
std::vector<velikiInt> Obicni_nadogradnja_3(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
```

```
{
	std::vector<velikiInt> prosti_brojevi;
	prosti_brojevi.push_back(2); //prvi prosti broj
	prosti_brojevi.push_back(3); //drugi prosti broj
	velikiInt kandidat = 5; //krecemo od 5
	//vremenske varijable
	LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

	//Trazimo proste brojeve od 5 do zadane granice
	velikiInt djeljitelj;
	bool prost_broj;
	int indeks_djeljitelja;
	int dodaj = 2; //na pocetku dodajemo 2
	while (kandidat <= gornja_granica)
	{
		djeljitelj = prosti_brojevi.at(0);
		indeks_djeljitelja = 0;
		prost_broj = true;
		//Za svaki kandidat provjeravamo djeljitelje s popisa
		while (djeljitelj * djeljitelj <= kandidat)
		{
			//Ako je djeljiv ostatak ce biti 0
			if (kandidat % djeljitelj == 0)
			{
				prost_broj = false;
				break;
			}
			//iduci djeljitelj na popisu
			indeks_djeljitelja++;
			if (indeks_djeljitelja >= prosti_brojevi.size())	//Ako je indeks veci nego je dopusteno, prekini
				break;
			djeljitelj = prosti_brojevi.at(indeks_djeljitelja);
		}
		//ako je detektiran prost broj stavi ga na popis prostih brojeva i djeljitelja
		if (prost_broj)
			prosti_brojevi.push_back(kandidat);
		kandidat += dodaj; //
		if (dodaj == 2)//alterniramo izmedu dodavanja 2 i 4
			dodaj = 4;
		else
			dodaj = 2;
	}
```

```
	//Prikazi vrijeme ako je zahtjevano
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
	QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
	vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
	if (prikazi_vrijeme)
		std::cout << "Obicni nadogradnja 3 - " << "Vrijeme pronalaska prostih brojeva: "
<< vrijeme << std::endl;

	//Vrati popis prostih brojeva
	return prosti_brojevi;
}

//Ovdje provjeravamo zadnje znamenke
std::vector<velikiInt> Obicni_nadogradnja_4(velikiInt gornja_granica, double &vrijeme, bool
prikazi_vrijeme = false)
{
	std::vector<velikiInt> prosti_brojevi;
	velikiInt kandidat = 11;
	prosti_brojevi.push_back(3);
	prosti_brojevi.push_back(7);
	//vremenske varijable
	LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

	//Trazimo proste brojeve od 11 do zadane granice
	velikiInt djeljitelj;
	bool prost_broj;
	int indeks_djeljitelja;
	int rez;
	while (kandidat <= gornja_granica)
	{
		//provjeravamo zadnje znamenke
		rez = kandidat % 10;
		if (!((rez == 1) || (rez == 3) || (rez == 7) || (rez == 9)))
		{
			kandidat++;
			continue;
		}

		djeljitelj = prosti_brojevi.at(0);
		indeks_djeljitelja = 0;
		prost_broj = true;
		//Za svaki kandidat provjeravamo djeljitelje s popisa
```

```
			while (djeljitelj * djeljitelj <= kandidat)
			{
				//Ako je djeljiv ostatak ce biti 0
				if (kandidat % djeljitelj == 0)
				{
					prost_broj = false;
					break;
				}
				//iduci djeljitelj na popisu
				indeks_djeljitelja++;
				if (indeks_djeljitelja >= prosti_brojevi.size())	//Ako je indeks veci nego je dopusteno, prekini
					break;
				djeljitelj = prosti_brojevi.at(indeks_djeljitelja);
			}
			//ako je detektiran prost broj stavi ga na popis prostih brojeva i djeljitelja
			if (prost_broj)
				prosti_brojevi.push_back(kandidat);
			kandidat++;
		}

		//Prikazi vrijeme ako je zahtjevano
		QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
		QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
		vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
		if (prikazi_vrijeme)
			std::cout << "Obicni nadogradnja 4 - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme << std::endl;

		prosti_brojevi.push_back(2);
		prosti_brojevi.push_back(5);

		//Vrati popis prostih brojeva
		return prosti_brojevi;
}

//Eratostenovo sito - original
std::vector<velikiInt> Eratosten_original(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
{
		std::vector<velikiInt> prosti_brojevi;
		//vremenske varijable
		LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
		QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);
```

```
	//Polje koje cemo korisiti za oznacavanje brojeva koji nisu prosti
	unsigned char* polje = new unsigned char[gornja_granica + 1];
	memset(polje, 0, (gornja_granica + 1) * sizeof(unsigned char));
	polje[0] = 1;
	polje[1] = 1;

	//Oznacavamo sve brojeve koji nisu prosti pocevsi od broja 2 do zadabe granice
	velikiInt trenutni = 2;
	velikiInt oznaci;
	while (trenutni <= gornja_granica)
	{
		//Ozvacavamo visekratnike trenutnog
		oznaci = trenutni + trenutni;
		while (oznaci <= gornja_granica)
		{
			polje[oznaci] = 1;
			oznaci += trenutni;
		}

		//Postavljamo trenutnog na slijedeci prosti broj (trazimo broj koji nije obiljezen)
		trenutni++;
		while (polje[trenutni])
			trenutni++;
	}

	//zapisjuemo preostale proste brojeve koje nismo zapisali u glavnoj petlji
	for (int i = 0; i < gornja_granica; i++)
	{
		if (!polje[i])
			prosti_brojevi.push_back(i);
	}

	//Prikazi vrijeme ako je zahtjevano
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
	QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
	vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
	if (prikazi_vrijeme)
		std::cout << "Eratosten original - " << "Vrijeme pronalaska prostih brojeva: " <<
vrijeme << std::endl;

	//Obrisi polje oznaka
	delete[] polje;
```

```
	//Vrati popis prostih brojeva
	return prosti_brojevi;
}

//Optimirano Eratostenovo sito
std::vector<velikiInt> Eratosten_optimiziran(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme = false)
{
	std::vector<velikiInt> prosti_brojevi;
	//vremenske varijable
	LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

	//Polje koje cemo korisiti za oznacavanje brojeva koji nisu prosti
	unsigned char* polje = new unsigned char[gornja_granica + 1];
	memset(polje, 0, (gornja_granica + 1) * sizeof(unsigned char));
	polje[0] = 1;
	polje[1] = 1;

	//Oznacavamo sve brojeve koji nisu prosti pocevsi od broja 2 do zadabe granice
	velikiInt trenutni = 2;
	velikiInt pKvadrat = 0;
	velikiInt tKvadrat = 0;
	velikiInt oznaci;
	while (trenutni * trenutni <= gornja_granica)
	{
		//Ozvacavamo visekratnike trenutnog
		oznaci = trenutni + trenutni;
		while (oznaci <= gornja_granica)
		{
			polje[oznaci] = 1;
			oznaci += trenutni;
		}

		//zapisujemo do sada određene proste brojeve
		tKvadrat = trenutni * trenutni;
		for (; pKvadrat < tKvadrat; pKvadrat++)
		{
			if (!polje[pKvadrat])
				prosti_brojevi.push_back(pKvadrat);
		}

		//Postavljamo trenutnog na slijedeci prosti broj (trazimo broj koji nije obiljezen)
		trenutni++;
```

```
            while (polje[trenutni])
                trenutni++;
        }

        //zapisjuemo preostale proste brojeve koje nismo zapisali u glavnoj petlji
        for (; pKvadrat <= gornja_granica; pKvadrat++)
        {
            if (!polje[pKvadrat])
                prosti_brojevi.push_back(pKvadrat);
        }

        //Prikazi vrijeme ako je zahtjevano
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
        QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
        vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
        if (prikazi_vrijeme)
            std::cout << "Eratosten optimiziran - " << "Vrijeme pronalaska prostih brojeva: "
<< vrijeme << std::endl;

        //Obrisi polje oznaka
        delete[] polje;

        //Vrati popis prostih brojeva
        return prosti_brojevi;
}

//Sundramovo sito
std::vector<velikiInt> Sundram(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme
= false)
{
        std::vector<velikiInt> prosti_brojevi;
        prosti_brojevi.push_back(2); //Sundramovo sito nece uhvatiti broj 2

        //vremenske varijable
        LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

        //Polje koje cemo korisiti za oznacavanje brojeva koji nisu prosti
        unsigned char* polje = new unsigned char[gornja_granica + 1];
        memset(polje, 0, (gornja_granica + 1) * sizeof(unsigned char));

        int pola = gornja_granica / 2;
        int maksVrijednost = 0;
        int djeljitelj = 0;
```

```
	for (int i = 1; i < pola; i++)
	{
		djeljitelj = (i * 2) + 1;
		maksVrijednost = (pola - i) / djeljitelj;
		for (int j = i; j <= maksVrijednost; j++)
			polje[i + j * djeljitelj] = 1;
	}

	for (int i = 1; i < pola; i++)
	{
		if (!polje[i])
			prosti_brojevi.push_back((i * 2) + 1);
	}

	//Prikazi vrijeme ako je zahtjevano
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
	QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
	vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
	if (prikazi_vrijeme)
		std::cout << "Sundram - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme <<
std::endl;

	//Vrati popis prostih brojeva
	return prosti_brojevi;
}

//Atkinovo sito
std::vector<velikiInt> Atkin(velikiInt gornja_granica, double &vrijeme, bool prikazi_vrijeme =
false)
{
	std::vector<velikiInt> prosti_brojevi;
	prosti_brojevi.push_back(2); //Atkinovo sit nece uhvatiti 2 i 3
	prosti_brojevi.push_back(3);

	//vremenske varijable
	LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
	QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

	//Polje koje cemo korisiti za oznacavanje brojeva koji nisu prosti
	unsigned char* polje = new unsigned char[gornja_granica + 1];
	memset(polje, 0, (gornja_granica + 1) * sizeof(unsigned char));

	velikiInt korjen = sqrt(gornja_granica);
```

```
int x = 1, xKorak = 3;
int y = 1, yKorak = 3;
int v = 0;

for (int i = 1; i <= korjen; i++)
{
	y = 1;
	yKorak = 3;
	for (int j = 1; j <= korjen; j++)
	{
		v = (x * 4) + y;

		if ((v <= gornja_granica) && (v % 12 == 1 || v % 12 == 5))
			polje[v] = !polje[v];

		v -= x;
		if ((v <= gornja_granica) && (v % 12 == 7))
			polje[v] = !polje[v];

		if (i > j)
		{
			v -= y * 2;
			if ((v <= gornja_granica) && (v % 12 == 11))
				polje[v] = !polje[v];
		}
		y += yKorak;
		yKorak += 2;
	}
	x += xKorak;
	xKorak += 2;
}

for (int g = 5; g <= korjen; g++)
{
	if (polje[g])
	{
		int k = g * g;
		for (int z = k; z <= gornja_granica; z += k)
			polje[z] = 0;
	}
}

for (int i = 1; i < gornja_granica; i++)
{
```

```
            if (polje[i])
                prosti_brojevi.push_back(i);
        }

        //Prikazi vrijeme ako je zahtjevano
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
        QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
        vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
        if (prikazi_vrijeme)
            std::cout << "Atkin - " << "Vrijeme pronalaska prostih brojeva: " << vrijeme <<
std::endl;

        //Vrati popis prostih brojeva
        return prosti_brojevi;
}

//Optimirano Atkinovo sito
std::vector<velikiInt> Atkin_optimiran(velikiInt gornja_granica, double &vrijeme, bool
prikazi_vrijeme = false)
{
        std::vector<velikiInt> prosti_brojevi;
        prosti_brojevi.push_back(2);
        prosti_brojevi.push_back(3);
        //vremenske varijable
        LARGE_INTEGER d_ctr1, d_ctr2, d_freq;
        QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr1);

        unsigned char* polje = new unsigned char[gornja_granica + 1];
        memset(polje, 0, (gornja_granica + 1) * sizeof(unsigned char));

        int korjen = sqrt(gornja_granica);

        int xKorak = 3;
        int yGranica = 0;
        int x = 0;

        //prolazak 3x^2 + y^2
        int g = sqrt((gornja_granica - 1) / 3);
        for (int i = 0; i < 12 * g; i += 24)
        {
            xKorak += i;
            yGranica = 12 * (int)sqrt(gornja_granica - xKorak) - 36;
            x = xKorak + 16;
            for (int j = -12; j < yGranica + 1; j += 72)
```

```
		{
			x += j;
			polje[x] = !polje[x];
		}

		x = xKorak + 4;

		for (int j = 12; j < yGranica + 1; j += 72)
		{
			x += j;
			polje[x] = !polje[x];
		}
}

//prolazak 4x^2 + y^2
xKorak = 0;
g = 8 * (int)sqrt((gornja_granica - 1) / 4) + 4;
for (int i = 4; i < g; i += 8)
{
		xKorak += i;
		x = xKorak + 1;

		if (xKorak % 3 != 0)
		{
			int gg = 4 * (int)sqrt(gornja_granica - xKorak) - 3;
			for (int j = 0; j < gg; j += 8)
			{
				x += j;
				polje[x] = !polje[x];
			}

		}
		else
		{
			yGranica = 12 * (int)sqrt(gornja_granica - xKorak) - 36;
			x = xKorak + 25;
			for (int j = -24; j < yGranica + 1; j += 72)
			{
				x += j;
				polje[x] = !polje[x];
			}

			x = xKorak + 1;
```

```
                for (int j = 24; j < yGranica + 1; j += 72)
                {
                        x += j;
                        polje[x] = !polje[x];
                }
        }
}

//prolazak 3x^2 - y^2
xKorak = 1;
g = (int)sqrt(gornja_granica / 2) + 1;
for (int i = 3; i < g; i += 2)
{
        xKorak += 4 * i - 4;
        x = 3 * xKorak;
        int s = 4;
        if (x > gornja_granica)
        {
                int min_y = ((int)sqrt(x - gornja_granica) / 4) * 4;
                x -= min_y * min_y;
                s = 4 * min_y + 4;
        }

        for (int j = s; j < 4 * i; j += 8)
        {
                x -= j;
                if (x <= gornja_granica && x % 12 == 11)
                        polje[x] = !polje[x];
        }

}

xKorak = 0;
for (int i = 2; i < g; i += 2)
{
        xKorak += 4 * i - 4;
        x = 3 * xKorak;
        int s = 0;
        if (x > gornja_granica)
        {
                int min_y = (((int)sqrt(x - gornja_granica) / 4) * 4) - 1;
                x -= min_y * min_y;
                s = 4 * min_y + 4;
        }
```

```
			else
			{
				x -= 1;
				s = 0;
			}
			for (int j = s; j < 4 * i; j += 8)
			{
				x -= j;
				if (x <= gornja_granica && x % 12 == 11)
					polje[x] = !polje[x];
			}
		}

		//Eliminiramo kvadrate
		for (int i = 5; i < korjen + 1; i += 2)
		{
			if (polje[i])
			{
				int k = i * i;
				for (int z = k; z < gornja_granica; z += k)
					polje[z] = 0;
			}
		}

		for (int i = 5; i < gornja_granica; i += 2)
		{
			if (polje[i])
				prosti_brojevi.push_back(i);
		}

		//Prikazi vrijeme ako je zahtjevano
		QueryPerformanceCounter((LARGE_INTEGER *)&d_ctr2);
		QueryPerformanceFrequency((LARGE_INTEGER *)&d_freq);
		vrijeme = (d_ctr2.QuadPart - d_ctr1.QuadPart) * 1000.0 / d_freq.QuadPart;
		if (prikazi_vrijeme)
			std::cout << "Atkin optimiran - " << "Vrijeme pronalaska prostih brojeva: " <<
vrijeme << std::endl;

		//Vrati popis prostih brojeva
		return prosti_brojevi;
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
double vrijeme = 0;
double sumObicni = 0;
double sumObicniN1 = 0;
double sumObicniN2 = 0;
double sumObicniN3 = 0;
double sumObicniN4 = 0;
double sumErastoten = 0;
double sumErastotenOpti = 0;
double sumSundram = 0;
double sumAtkin = 0;
double sumAtkinOpti = 0;
int brCiklusa = 100;
for (int i = 0; i < brCiklusa; i++)
{
	std::vector<velikiInt> prosti_brojevi_obicni;
	prosti_brojevi_obicni = Obicni(10000000, vrijeme, true);
	sumObicni += vrijeme;
	std::vector<velikiInt> prosti_brojevi_obicni_N1;
	prosti_brojevi_obicni_N1 = Obicni_nadogradnja_1(10000000, vrijeme, true);
	sumObicniN1 += vrijeme;
	std::vector<velikiInt> prosti_brojevi_obicni_N2;
	prosti_brojevi_obicni_N2 = Obicni_nadogradnja_2(10000000, vrijeme, true);
	sumObicniN2 += vrijeme;
	std::vector<velikiInt> prosti_brojevi_obicni_N3;
	prosti_brojevi_obicni_N3 = Obicni_nadogradnja_3(10000000, vrijeme, true);
	sumObicniN3 += vrijeme;
	std::vector<velikiInt> prosti_brojevi_obicni_N4;
	prosti_brojevi_obicni_N1 = Obicni_nadogradnja_4(10000000, vrijeme, true);
	sumObicniN4 += vrijeme;
	std::vector<velikiInt> prosti_brojevi_Eratosten;
	prosti_brojevi_Eratosten = Eratosten_original(10000000, vrijeme, true);
	sumErastoten += vrijeme;
	std::vector<velikiInt> prosti_brojevi_Eratosten_opti;
	prosti_brojevi_Eratosten_opti = Eratosten_optimiziran(10000000, vrijeme, true);
	sumErastotenOpti += vrijeme;
	std::vector<velikiInt> prosti_brojevi_Sundram;
	prosti_brojevi_Sundram = Sundram(10000000, vrijeme, true);
	sumSundram += vrijeme;
	std::vector<velikiInt> prosti_brojevi_Atkin;
	prosti_brojevi_Atkin = Atkin(10000000, vrijeme, true);
	sumAtkin += vrijeme;
	std::vector<velikiInt> prosti_brojevi_Atkin_opti;
	prosti_brojevi_Atkin_opti = Atkin_optimiran(10000000, vrijeme, true);
	sumAtkinOpti += vrijeme;
```

```
        }
        std::cout << "Obicni: " << sumObicni / brCiklusa << std::endl;
        std::cout << "Obicni N1: " << sumObicniN1 / brCiklusa << std::endl;
        std::cout << "Obicni N2: " << sumObicniN2 / brCiklusa << std::endl;
        std::cout << "Obicni N3: " << sumObicniN3 / brCiklusa << std::endl;
        std::cout << "Obicni N4: " << sumObicniN4 / brCiklusa << std::endl;
        std::cout << "Eratosten: " << sumErastoten / brCiklusa << std::endl;
        std::cout << "Eratosten opti: " << sumErastotenOpti / brCiklusa << std::endl;
        std::cout << "Sundram: " << sumSundram / brCiklusa << std::endl;
        std::cout << "Atkin: " << sumAtkin / brCiklusa << std::endl;
        std::cout << "Atkin opti: " << sumAtkinOpti / brCiklusa << std::endl;
        return 0;
}
```