UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
FAKULTETA ZA MATEMATIKO IN FIZIKO

Andreja Slosu

Testiranje praštevilskosti

DIPLOMSKO DELO
NA INTERDISCIPLINARNEM UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Borut Robič

Ljubljana, 2013

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00053/2013
Datum: 03.09.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **ANDREJA SLOSU**

Naslov: **TESTIRANJE PRAŠTEVILSKOSTI**
PRIMALITY TESTING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Z razmahom informacijske tehnologije in težnjo po zaščiti informacij se vse bolj zavedamo pomembnosti protokolov na osnovi praštevil. V delu predstavite razvoj algoritmov za testiranje praštevilskosti skozi zgodovino pa vse do danes. Opiše najpomembnejše algoritme, ki se še uporabljajo v praksi, in njihove glavne ideje. Predstavite prvi deterministični algoritem in njegove izboljšave. Dokaze pravilnosti algoritmov opišite na neformalen, bolj poljuden način.

Mentor:

prof. dr. Borut Robič

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Andreja Slosu,
z vpisno številko 63060242,

sem avtorica diplomskega dela z naslovom:
Testiranje praštevilskosti.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom prof. dr. Boruta Robiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 20. 09. 2013 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju, profesorju Borutu Robiču, ki si je vedno vzel čas za nasvet in pomoč pri iskanju literature. Zahvaljujem pa se tudi svoji družini, za vso podporo.

Vsako praštevilo stoji na svojem mestu v neskončnem zaporedju naravnih števil, stisnjeno med dve števili kot vsa druga, a v primerjavi z njimi vendarle samo zase.

Paolo Giordano

Kazalo

Seznam uporabljenih kratic in simbolov

AKS	algoritem Agrawal-Kayal-Saxena
RSA	algoritem Rivest-Shamir-Adleman
gcd(a,b)	greatest common divisor, slovensko največji skupni delitelj
lcm(a,b)	least common multiple, slovensko najmanjši skupni večkratnik
$\mathbb{Z}_n$	množica ostankov po modulu n
$\mathbb{Z}_n^*$	množica vseh obrnljivih elementov iz $\mathbb{Z}_n$
$x \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$	tisti x, ki niso obrnljivi
$\mathbb{Z}_n[x]$	množica polinomov s koeficienti iz $\mathbb{Z}_n$
$\pi(n)$	število praštevil manjših od n
$\mathbb{P}$	množica praštevil
$\phi(n)$	število tujih si števil s številom n
$a \equiv b \pmod{n}$	a je kongruentno b po modulu n
$a \mid b$	a deli b
$\left(\frac{a}{p}\right)$	Legendrov simbol

Povzetek

Z razmahom informacijske tehnologije in prodora te tehnologije na vedno več področij, npr. na razna spletna poslovanja, se porodi veliko vprašanj glede varnosti. Podatki v elektronski obliki prinašajo številne prednosti, ampak so bolj izpostavljeni raznim zlorabam. Zato jih moramo ustrezno zaščititi. Ko opravljamo transakcijo preko interneta, naš brskalnik zakodira številko kreditne kartice. Ti algoritmi za šifriranje pa temeljijo na teoriji praštevil. Npr. RSA asimetrični algoritem za izračun modula n potrebuje dve veliki praštevili p in q. Včasih so se za praštevila zanimali zaradi njihove drugačnosti. Evklid je okoli 300 pr. n. št. zapisal prve definicije o praštevilih. Eratostenovo rešeto je prvi znani algoritem za dokazovanje ali gre za praštevilo ali ne. Leta 1640 je Femat zapisal svoj slavni izrek, na katerem temelji veliko kasnejših algoritmov za dokazovanje praštevilskosti. Izreka mu ni uspelo dokazati, sta pa ga kasneje dokazala Leibniz in Euler.

Diplomska naloga nudi pregled algoritmov za testiranje praštevilskosti od antike pa do danes. Najprej smo naredili pregled naivnih algoritmov, ki so enostavni, ampak pri velikih številih niso več uporabni. Potem so opisani verjetnostni algoritmi, ki temeljijo na naključnih podatkih. Opisana sta Solovay-Strassenov in Miller-Rabinov algoritem in njihova analiza pravilnosti in časovne zahtevnosti. Ravno ta dva algoritma se v praksi najpogosteje uporabljata. Čeprav ti algoritmi ne vrnejo vedno pravilnega odgovora, se zaradi njihove hitrosti največ uporabljajo. Na koncu smo opisali še deterministični algoritem AKS, ki so ga leta 2002 zapisali Agrawal, Kayal in Saxena. Čeprav je bilo kar nekaj poskusov že prej, je njim prvič uspelo dokazati, da je pro-

blem ugotavljanja praštevilskosti v razredu P, tj. problemov, ki so rešljivi v polinomskem času. Ta algoritem je velik dosežek za teoretično računalništvo. Opisali smo osnovno idejo prve različice AKS algoritma in dve njeni izboljšavi avtorjev Lenstra-Pomeranca in Bernsteina. V nadaljevanju pa je natančno opisana šesta različica algoritma iz leta 2004.

Ključne besede: praštevila, testiranje praštevilskosti, algoritem AKS.

Abstract

With the boom in information technology and the penetration of these technologies in an increasing number of areas, such as electronic business, many questions about security arise. Data in electronic form bring many benefits, but they are vulnerable to various abuses. Therefore, data need to be adequately protected. For instance, when we perform a transaction over the Internet, our browser encrypts the credit card number. To do this, the browser usually uses encryption algorithms that are based on the theory of primes. For example: RSA asymmetric algorithm needs two large prime numbers p and q to calculate the module n.

In the past, the primes were interesting because of their uniqueness. Euclid wrote the first definition of primes around 300 BC. The Sieve of Eratosthenes is the first known algorithm for deciding whether or not a number is a prime. In 1640, Fermat wrote his famous theorem to prove the primality of a number. The theorem is the basis of many subsequent algorithms. However, Fermat failed to prove the theorem, so that it was only later proved by Leibniz and Euler.

My bachelor thesis gives an overview of algorithms for primality testing from antique times to the present. First of all, we give an overview of the naive algorithms that are simple but not suitable for large numbers. Then we describe probabilistic algorithms, i.e., algorithms that use random data. Next we describe the Solovay-Strassen and Miller-Rabin's algorithm and analyze their correctness and time complexity. The two algorithms are the most commonly used in practice. Even though these algorithms do not always

return a correct answer, they are often used due to their speed. Next, we describe the deterministic algorithm AKS, discovered in 2002 by Agrawal, Kayal and Saxena. Although there have been several attempts, they are the first who succeeded to prove that the primality testing problem is in the class P of problems solvable in polynomial time. Their algorithm is a major achievement for theoretical computer science. Next, we describe the basic idea of the first version of the AKS algorithm and two improvements due to Lenstra-Pomerance and Bernstein. Finally, we describe in detail the sixth version of this algorithm from 2004.

Key words: prime numbers, primality testing, AKS algorithm.

Poglavje 1

Uvod

Uporaba praštevil je zelo raznolika. Prvotno so jih preučevali, ker se veliko matematičnih problemov nanaša na faktorizacijo števil. Danes so praštevila ključnega pomena na področju interneta. Praštevila se uporabljajo za različne metode šifriranja, da transakcije varno tečejo. Praštevila so pritegnila zanimanje matematikov skozi več stoletij. Spraševali so se, koliko jih je? Ali obstaja formula, s katero jih lahko generiramo?

Praštevila pa niso zanimiva samo z vidika teorije števil, ampak se je njihova uporabnost razširila tudi na druga področja. V kriptografiji so dobila zelo pomembno vlogo z asimetričnim šifriranjem. Pred tem je bilo znano le simetrično šifriranje na podlagi skrivnih ključev. Leta 1976 sta Whitfield Diffie in Martin Hellman odkrila postopek za izmenjavo ključev [28]. Ideja je bila, da naj bi imel vsak uporabnik svoj zasebni ključ in javni ključ. Slednji bi bil javno objavljen in dostopen vsem, ki bi skušali komunicirati z lastnikom javnega ključa. Če je, A želel poslati šifrirano sporočilo B-ju, je to naredil tako, da je zašifriral sporočilo z B-jevim javnim ključem in to poslal B-ju. Ko je B sporočilo prejel, je s svojim zasebnim ključem odšifriral sporočilo. To je bila prva metoda v praksi za prenos tajnih podatkov preko nezaščitenega kanala. Najbolj enostavna različica protokola uporablja modul p, kjer je p praštevilo.

Leta 1977 pa se je pojavil še asimetrični sistem RSA [25, 28] in [32, str. 161-163, 173-177], ki so ga odkrili Ron Rivest, Adi Shamir in Len Adleman. Sistem vsebuje dva ključa, zasebnega in javnega. Javni ključ poznajo vsi in se uporablja za šifriranje. Z zasebnim ključem pa se sporočilo odšifrira. RSA kriptosistem izvaja računanje v $\mathbb{Z}_n$, kjer modul n izračunamo kot produkt dveh različnih velikih praštevil p in q. Če sta p in q dovolj veliki, potem faktorizacija ni možna v nekem razumnem času, saj je problem faktorizacije računsko zahteven. Števili p in q se izbereta naključno. Zato potrebujemo učinkovit algoritem, da preverimo, ali sta p in q res praštevili.

Diffie-Helmanov protokol je namenjen generiranju skrivnega ključa, ki ga pošljemo po nezaščitenem kanalu in ga kasneje uporabimo pri simetričnem šifriranju. RSA se uporablja pri asimetričnem šifriranju. Lahko pa ga uporabljamo še za digitalne podpise. Diffie-Helmanov protokol bi lahko razbili, če bi znali učinkovito izračunati diskretni algoritem, RSA pa, če bi poznali učinkovite algoritme za izračun faktorizacije.

Poglavje 2

Praštevila

Praštevila so naravna števila, ki so večja od 1 in ki so deljiva samo z 1 in sama s sabo. Števila, večja od 1, za katera ne velja zgoraj zapisana lastnost, so sestavljena. Število 2 je edino sodo praštevilo. S problemom kako definirati praštevila in sestavljena števila, so se ukvarjali že od antike naprej. Evklid je v 4. stoletju pr. n. št. v svoji knjigi podal naslednje definicije [12], citiramo po [14, str. 11]:

- *Definicija 1. Enota je to, zaradi česar se vsaka od obstoječih stvari imenuje ena.*
- *Definicija 2. Število je množina, sestavljena iz enot.*
- *Definicija 3. Število je del drugega števila, manjše od večjega, če meri večjega*

Tretjo definicijo razumemo kot $a \mid b$.

- *Definicija 11. Praštevilo je število, ki ga meri le enota.*
- *Definicija 12. Tuja števila so tista, ki imajo za skupno mero le enoto.*
- *Definicija 13. Sestavljeno število je tisto, ki ga meri kakšno število.*

Te definicije razumemo, kot da je število praštevilo, če ga deli le enota, če pa ga poleg enote deli še kako drugo število, je sestavljeno število.

Enostavna, toda zelo počasna metoda za preverjanje, ali je neko število praštevilo, je algoritem zaporednih deljenj. Ta po vrsti preverja ali je n sestavljeno s kakim od števil med 2 in $\sqrt{n}$. Algoritem je za velike n tako počasen, da postane neuporaben. Zato so skozi zgodovino matematike preučevali praštevila in poskušali razviti bolj učinkovite algoritme za odločanje ali gre za sestavljeno število ali za praštevilo. Tako so odkrili hitre algoritme za dokazovanje praštevilskosti za števila izbranih oblik. Taka so npr. Mersennova števila [14, str. 35-38].[1] Da obstaja neskončno mnogo praštevil je dokazal že Evklid okoli 300 pr. n. št. Ne znamo točno razpoznavati praštevila in sestavljena števila, ampak so dokazali, da obstaja formula o gostoti praštevil, in kako so ta porazdeljena.

2.1 Zgodovina praštevil

Že Stari Egipčani naj bi poznali praštevila, vendar najstarejši znani zapisi segajo v antično Grčijo tam okoli 300 pr. n. št., ko je Evklid v svojih elementih zapisal definicije, ki že spominjajo na kasnejše definicije. Zapisal je tudi, da je števil neskončno mnogo. Eratosten iz Sirene pa je zapisal algoritem za ugotavljanje ali je dano število praštevilo; danes algoritmu rečemo Eratostenovo rešeto. Potem je bilo na področju praštevil zatišje vse do 17. stoletja. Leta 1640 je Fermat podal svoj znameniti izrek, Fermatov mali izrek, vendar brez dokaza. Kasneje sta ga dokazala Leibniz in Euler. Fermat je postavil še domnevo, da so vsa števila oblike $2^{2^n} + 1$ praštevila. Takim številom rečemo Fermatova števila [14, str. 31-35]. Uspelo mu je preveriti, da to velja za $n \leq 4$. Kasneje je Euler pokazal, da je že naslednje število $2^{32} + 1$ sestavljeno. Francoski menih Mersenne pa je obravnaval števila oblike $2^p - 1$

[1] Februarja 2013 so našli tako število s 17,425,170 števkami.

kjer je p praštevilo.

Glede praštevil je Euler veliko odkril. Pokazal je, da je neskončno zaporedje $\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots$ divergentno. Leta 1747 je pokazal, da je vsako popolno število oblike $2^{p-1}(2^p - 1)$, kjer je drugi faktor Mersennovo število.

Na začetku 19. stoletja sta Legendre in Gauss neodvisno eden od drugega domnevala, da obstaja formula za gostoto praštevil. Leta 1859 je Riemann podal domnevo o zeta funkciji, ki je kasneje omogočila dokaz formule za gostoto praštevil [29]. To sta leta 1896 neodvisno dokazala Hadamard in de la Vallée Poussin.

Raziskave na področju dokazovanja, da je neko veliko število praštevilo, še zdaleč niso končane. Matematiki in računalničarji se še vedno močno trudijo, da bi izboljšali algoritme za testiranje praštevilskosti.

2.2 Osnovna definicija praštevil

Če je naravno število n praštevilo, potem ima natanko dva delitelja: 1 in n. Za naravno število, ki ima več kot dva delitelja, rečemo, da je sestavljeno. Pogoj, da je neko število praštevilo, lahko zapišemo tudi drugače. Praštevilo je praštevilo, če je večje od 1 in nobeno število izmed $2, \ldots, n-1$ ne deli n brez ostanka. Množico praštevil običajno označujemo s $\mathbb{P}$.

2.3 Osnovni izrek teorije števil

Osnovni izrek govori o faktoriziranju števila na praštevilske faktorje. Vsako celo število večje od 1 lahko zapišemo kot produkt enega ali več praštevil. Vsako sestavljeno število lahko zapišemo v kanonični obliki na naslednji način:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i},$$

kjer so $p_1 < p_2 < \cdots < p_k$ praštevila in α_i pozitivna cela števila.
Praštevila so tako osnovni gradniki drugih števil. Osnovni izrek je prvi zapisal Gauss,[2] ampak vse kaže na to, da je to dejstvo poznal že Evklid, ki je v svoji Knjigi VII zapisal naslednji trditvi, citiramo po [14, str. 12-13]:

- *Trditev 30: Če z množenjem dveh števil dobimo neko število, potem vsako praštevilo, ki meri produkt, meri tudi eno od prvotnih števil.*

Ta trditev je danes poznana kot Gaussov izrek: Če je p praštevilo in p deli produkt dveh števil a in b, potem p deli a ali pa p deli b.

- *Trditev 32: Vsako število je bodisi praštevilo bodisi ga meri neko praštevilo.*

Ta trditev pa ni več tako daleč od osnovnega izreka teorije števil.

[2] Recherches Arithmetiques, 1801

2.4 Praštevil je neskončno

Če je dano poljubno končno zaporedje števil, v njem niso zajeta vsa praštevila. Ta trditev je znana kot Evklidov izrek.

Dokaz. Opazujemo poljubno končno množico P, ki vsebuje sama praštevila. Ključna ideja dokaza je, da če zmnožimo vsa praštevila iz P in temu prištejemo 1, potem nobeno praštevilo, iz P ne deli dobljenega števila. Naj bo

$$n = \prod_{p \in P} p + 1.$$

Število n je deljivo vsaj z enim praštevilom iz množice $\mathbb{P}$, ali pa je samo praštevilo. Toda nobeno praštevilo iz P ne more deliti n, saj vsako tako deljenje da ostanek 1. Očitno pa n ni v $\mathbb{P}$. Torej je n samo praštevilo. □

Obstaja tudi Eulerjev analitični dokaz [7].

2.5 Gostota praštevil

Vprašajmo se, koliko naključnih števil je potrebno testirati, dokler ne najdemo praštevila? Naj bo $\pi(n)$ aritmetična funkcija, katere vrednost je število praštevil, ki so manjša kot n. Rezultat iz teorije števil kaže, da je $\pi(n) \approx \frac{n}{\ln n}$ [24]. Torej, če p izberemo med 1 in n, je verjetnost, da je p praštevilo, enaka $\frac{\frac{n}{\ln n}}{n} = \frac{1}{\ln n}$. Naključno 512-bitno število je praštevilo z verjetnostjo $\frac{1}{\ln 2^{512}} \approx \frac{1}{355}$; če se osredotočimo le na liha števila, pa je verjetnost večja, in sicer $\frac{2}{355}$.

Poglavje 3

Osnove

V tem poglavju navedemo nekaj definicij in lastnosti osnovnih pojmov, ki jih bomo potrebovali v nadaljevanju.

lcm - najmanjši skupni večkratnik
Najmanjši skupni večkratnik dveh števil a in b je najmanjše število, ki je deljivo tako z a kot z b. Označimo ga z $\text{lcm}(a, b)$.

gcd - največji skupni delitelj
Največji skupni delitelj dveh števil a in b je največje število, ki deli a in b brez ostanka. Velja: $\gcd(a, b) = \frac{a \cdot b}{lcm(a,b)}$
Algoritem za izračun gcd je [32][str. 164]:

Algoritem 1 Algoritem gcd

Vhod: števili $a, b \in \mathbb{Z}$

Izhod: $\gcd(a, b)$

1: **while** $b > 0$ **do**
2: $t = b$
3: $b = a \pmod{t}$
4: $a = t$
5: **vrni** a
6: **end while**

Naj bosta (a_i, b_i) vrednosti, shranjeni v spremenljivkah a in b po i-ti iteraciji zanke. Potem je $(a, b) = (a_i, b_i)$, kar implicira, da algoritem vrne (a, b). Števila $b_1, b_2, \ldots$ tvorijo strogo padajoče zaporedje, kar zagotavlja, da se algoritem ustavi. Kakšna pa je časovna zahtevnost algoritma?

Lema 3.1 *Časovna zahtevnost algoritma 1 je* $\mathrm{O}\big(\log a \cdot \log b\big)$ *[19, str. 15-16].*

Grupa $(G, \circ)$ je neprazna množica, z operacijo $\circ$, za katero velja:

- zaprtost: $a \circ b \in G$ za vsak $a, b \in G$
- asociativnost: $(a \circ b) \circ c = a \circ (b \circ c)$ za vse $a, b, c \in G$
- v G obstaja nevtralen element e, za katerega velja: $e \circ a = a \circ e = a$ za vsak $a \in G$
- za vsak element a iz G obstaja element b, za katerega velja: $a \circ b = b \circ a = e$. Element b je inverz elementa a in ga označimo a^{-1}

Abelova grupa $(G, \circ)$ je komutativna, če je $a \circ b = b \circ a$ za poljubna $a, b \in G$.

Polgrupa je množica S z operacijo $\circ$, za katero velja:

- za vsak par $a, b \in S$ velja $a \circ b \in S$ (zaprtost)
- $a, b, c \in S$ velja $(a \circ b) \circ c = a \circ (b \circ c)$ (asociativnost)

Polgrupa v splošnem nima nevtralnega elementa.

Kolobar je množica K z dvema računskima operacijama, za kateri velja zakon distributivnosti. Zaradi enostavnosti rečemo, da sta ti dve operaciji kar $+$ in $\cdot$. Za kolobar velja:

- $(K, +)$ je Abelova grupa
- $(K, \cdot)$ je polgrupa
- distributivnost: za poljubne elemente $a, b, c \in K$ velja:
  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ in $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$

Obseg je množica $(O, +, \cdot)$, v kateri velja:

- $(O, +, \cdot)$ je kolobar
- obstaja nevtralni element 1 za množenje: $a \cdot 1 = 1 \cdot a = a$, za vsak $a \in O$
- vsak element, različen od 0, je obrnljiv: $a \cdot a^{-1} = a^{-1} \cdot a = 1$

Polje je komutativen obseg.

V tej diplomski nalogi se osredotočimo na **praštevilski obseg**, ki ga označimo z $\mathbb{Z}_p$, kjer je p praštevilo.Torej:

- za praštevilo p naj bo $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$

Elemente množice $\mathbb{Z}_p$ lahko seštevamo in množimo po modulu p:

- inverz števila $x \in \mathbb{Z}_p$ je število a, ki zadošča enačbi $x \cdot a \equiv 1 \pmod{p}$
- vsi elementi iz $\mathbb{Z}_p$ razen 0 so obrnljivi
- z $\mathbb{Z}_p^*$ označimo množico vseh obrnljivih elementov množice $\mathbb{Z}_p$, zato je $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$

Struktura $\mathbb{Z}_p^*$

- $\mathbb{Z}_p^*$ je ciklična grupa.

Z drugimi besedami, obstaja $g \in \mathbb{Z}_p^*$, tako da je $\mathbb{Z}_p^* = \{1, g, g^2, \ldots, g^{p-2}\}$. Rečemo, da je g generator $\mathbb{Z}_p^*$. Navedimo še naslednje:

- red generatorja $g \in \mathbb{Z}_p^*$ je najmanjše pozitivno število k tako, da velja $g^k \equiv 1 \pmod{p}$. Red g-ja označimo z $\mathrm{or}_p(g)$.
- velja Langrangev izrek: $\mathrm{or}_p(g) \mid (p-1)$, za vsak $g \in \mathbb{Z}_p^*$

Kongruence[1]

Če imata dve števili a in b lastnost, da je njuna razlika deljiva s številom m, potem rečemo, da je a kongruentno b po modulu m. To zapišemo:

$$a \equiv b \pmod{m}.$$

Če $m \nmid (a-b)$, potem a ni kongruentno b po modulu m in to zapišemo kot:

$$a \not\equiv b \pmod{m}.$$

Polinomi

Polinome s koeficienti iz polja $\mathbb{F}$ zapišemo kot $f(X) = \sum_{i=0}^{d} a_i X^i$, kjer je d največje število, za katero velja $a_d \neq 0$. Številu d rečemo stopnja polinoma[2] in za to uporabljamo oznako $deg(f)$. Množico vseh polinomov s koeficienti iz $\mathbb{F}$ zapišemo kot $\mathbb{F}[X]$.
Velja, da je c **ničla** polinoma natanko takrat, ko $(X-c)$ deli $f(X)$. Število ničel polinoma je omejeno navzgor s stopnjo polinoma.

[1] Kongruence je uvedel Carl Friedrich Gauss v 18. stoletju.
[2] ničelni polinom je polinom, kjer so vsi koeficienti enaki 0

Nerazcepni polinomi

Polinom je nerazcepen, če ga ne moremo izraziti s produktom dveh ali več polinomov manjše stopnje od stopnje začetnega polinoma.

Zgleda: $(x^2 - 1) = (x - 1)(x + 1)$,
$(x^2 + 1)$ je nerazcepen v $\mathbb{R}$.

Polinom s koeficienti v $\mathbb{F}$ je nerazcepen, če ni konstanta in se ga ne da faktorizirati v dva ali več nekostantnih polinomov s koeficienti v polju $\mathbb{F}$.

Poglavje 4

Naivni algoritmi

4.1 Algoritem zaporednih deljenj

To je najbolj enostaven algoritem za preverjanje ali je neko število n praštevilo ali sestavljeno število. Preveriti moramo le, če obstaja tak m, ki je med 2 in $n-1$, da deli n. Če je n res praštevilo, moramo postopek nadaljevati do konca. V resnici pa ni potrebno preveriti vseh števil med 2 in $n-1$. Zadostuje, da preverjamo do števila $\lfloor\sqrt{n}\rfloor$. Naj bo n enak pq. Potem p in q delita n. Naj bo število p najmanjši delitelj števila n, različen od 1. Potem je p praštevilo. Torej, če je n sestavljeno število, potem je $p < \sqrt{n}$ in $q > \sqrt{n}$. Vsako sestavljeno število ima praštevilski delitelj, manjši od $\sqrt{n}$. Ta ideja se uporablja v algoritmu zaporednih deljenj.

Če poznamo vsa praštevila med 2 in n, lahko algoritem izvajamo le na teh. Tak algoritem je uporaben za $n \leq 10^{12}$, kjer preverjamo praštevila, manjša od 10^6. Torej je deliteljev, ki jih moramo preveriti kvečjemu, $\frac{10^6}{\log 10^6} \approx 75000$. Časovna zahtevnost algoritma 2 je $\mathrm{O}\big(\sqrt{n}\big)$, kar je $\mathrm{O}\big(2^{\frac{\log_2 n}{2}}\big)$. Zadnji zapis smo navedli zato, ker kaže odvisnost od dolžine vhodnega podatka, tj. od $\lceil\log_2 n\rceil$.

Algoritem 2 Algoritem zaporednih deljenj

Vhod: število n

Izhod: p (če obstaja)

```
1: for i = 2 to ⌊√n⌋ do
2:     if n%i == 0 then
3:         vrni false
4:     else
5:         vrni true
6:     end if
7: end for
```

4.2 Eratostenovo rešeto

Če poznamo vsa praštevila med 2 in n, lahko najdemo praštevilski razcep vseh števil do n^2. Zato so tabele praštevil zelo uporabne. Eratostenovo rešeto je preprost algoritem za iskanje vseh praštevil, manjših od nekega vnaprej danega števila n. Najprej izpišemo vsa števila od 2 do n. Če je število n majhno, lahko z zaporednim izločanjem najprej večkratnikov 2, nato večkratnikov 3, večkratnikov 5 in tako naprej, dobimo vsa praštevila, manjša od n. Ta postopek izločanja pripisujemo Eratostenu iz Sirene, ki je živel v tretjem st. pr. n. št., čeprav je proces prvi opisal Pitagorejec Nikomah iz Gerase. Pridobivanje praštevil se imenuje Eratostenovo rešeto, ker če vzamemo vsa liha števila skupaj, jih ta algoritem razdeli v dve skupini na način, ki bi ga dobili z rešetom, torej na eni strani sestavljena števila na drugi strani pa praštevila.

Algoritem 3 Algoritem Eratostenovo rešeto

Vhod: število n

1: **for all** $m = 2$ to $\sqrt{n}$ **do**
2: **if** m neoznačeno **then**
3: dodaj m v seznam praštevil
4: označi vse večkratnike (tj. vsa števila $k \cdot m \leq n$, kjer je $k \geq m$)
5: **else**
6: m je sestavljeno število
7: **end if**
8: **end for**
9: preveri vsa števila med $\sqrt{n}$ in n, vsa najdena neoznačena števila dodaj v seznam praštevil

4.3 Fermatov algoritem

To je algoritem za iskanje faktorizacije števil [14, 21]. Metodo je opisal Fermat leta 1643. Ideja algoritma je razcepiti liho število n kot razliko dveh kvadratov: $n = a^2 - b^2 = (a + b)(a - b)$. Vsako liho število se namreč da zapisati v taki obliki. Zanima nas ali je $a^2 - n$ kvadrat za zaporedna naravna števila a od $\sqrt{n}$ naprej. Naj bo $n = pq$ faktorizacija števila n in $q \geq p$. Potem je $n = (\frac{q+p}{2})^2 - (\frac{q-p}{2})$, kjer je $a = \sqrt{n} + \frac{1}{2}(\sqrt{q} - \sqrt{p})^2 = \frac{q+p}{2}$. Torej je za faktorizacijo sestavljenega števila n potrebnih $\frac{1}{2}(\sqrt{q} - \sqrt{p})^2$ korakov.

Algoritem 4 Fermatova faktorizacija števila n

Vhod: število n

Izhod: faktorizacija n oz. vrne n je praštevilo

1: $a = \lceil \sqrt{n} \rceil$
2: $b^2 = a^2 - n$
3: **while** b^2 ni kvadrat **do**
4: $\quad a = a + 1$
5: $\quad b^2 = a^2 - n$
6: **end while**
7: **vrni** $a - \sqrt{b^2}$

Če je n praštevilo, se algoritem ustavi, saj velja $(\frac{n+1}{2}) - (\frac{n-1}{2}) = n$. Leta 1891 je Lucas dokazal, da je to edini način zapisa praštevila n kot razlike dveh kvadratov.

Poglavje 5

Verjetnostni algoritmi

Najbolj pogosti algoritmi za testiranje praštevilskosti so verjetnostni algoritmi. V teh algoritmih se poleg števila n, ki ga testiramo, pojavi še število a, ki je izbrano naključno. Ponavadi verjetnostni algoritmi nikoli ne proglasijo praštevila za sestavljeno število, toda možno je, da sestavljeno število proglasijo za praštevilo. Verjetnost napake se manjša, če ponavljamo test pri neodvisno izbranih vrednosti a in test vztraja pri odgovoru. Osnovna struktura verjetnostnih algoritmov za testiranje praštevil je naslednja:

1. Naključno izberi število a.
2. Preveri nekatere enakosti (odvisno od algoritma), ki vključujejo a in n. Če enakost ne drži, potem je n sestavljeno število, število a pa je priča za sestavljenost števila n, in algoritem se ustavi.
3. Če enakost drži, ponavljaj od 1. koraka naprej, dokler ni dosežena potrebna gotovost.

Najbolj enostaven verjetnostni algoritem za preverjanje praštevilskosti je Fermatov algoritem. Algoritem temelji na Fermatovem malem izreku. Citiramo[1]:

[1]Fermat, Oeuvres, t. II, Letress 'a Frenicle du 18-10-1640, str. 206

V nadaljevanju se mi zdi pomembno, da vam povem osnovo, na kateri sem postavil dokaze vsega, kar zadeva geometrijskega zaporedja. Vsako praštevilo zagotovo meri eno od potenc -1 v kateremkoli zaporedju, in eksponent tega zaporedja je večkratnik praštevila -1, potem, ko najdemo prvo potenco, ki temu ustreza, bodo ustrezale tudi vse tiste, katerih eksponenti so večkratniki eksponenta praštevila[...] In ta trditev velja v splošnem za vsa zaporedja in vsa praštevila, dokaz tega bi vam poslal, če se ne bi bal, da je predolg ...

Iz zgornjega besedila izhaja Mali Fermatov izrek, ki ga je kasneje dokazal Euler.

Izrek 5.1 (Mali Fermatov izrek) *Naj bo p praštevilo in $1 \leq a < p$. Potem velja $a^{p-1} \equiv 1 \pmod{p}$.*

Če hočemo preveriti ali je p praštevilo, potem izberemo naključno število a iz intervala $[1, p)$. Če enakost $a^{p-1} \equiv 1 \pmod{p}$ ne velja, potem je p sestavljeno število in a je priča za sestavljenost števila p. Če pa enakost velja, rečemo, da je p verjetno praštevilo. Gotovost odgovora je odvisna od tega na koliko različnih vrednostih a smo preverili enakost $a^{p-1} \equiv 1 \pmod{p}$. Lahko se zgodi, da pri vsaki vrednosti a enakost $a^{p-1} \equiv 1 \pmod{p}$ velja. Če vemo, da je n sestavljeno število in vseeno zgornja enakost velja, potem vsakemu takemu a rečemo, da je Fermatov lažnivec.

Algoritem 5 Algoritem po Fermatovem izreku

Vhod: število n, ki ga testiramo; k parameter, kolikokrat poženemo algoritem

Izhod: faktorizacija števila n; če se ne da faktorizirati, potem odgovor, da je število n praštevilo

1: ponovi k-krat:
2: izberi a naključno iz intervala $[1, n-1]$
3: **if** $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
4: **vrni** "n je sestavljeno število" (in zaključi)
5: **end if**
6: **vrni** "n verjetno praštevilo" (in zaključi)

Časovna zahtevnost algoritma 5. Predpostavimo, da uporabljamo hitre algoritme za potenciranje po modulu, npr. **algoritem Kvadriraj in množi** [32, str. 177]. Ta algoritem predpostavlja, da je eksponent c zapisan v binarni obliki na naslednji način:

$$c = \sum_{i=0}^{l-1} c_i 2^i, \tag{5.1}$$

kjer je $c_i = 0$ ali 1 in l število bitov v binarnem zapisu števila c.

Algoritem 6 Algoritem Kvadriraj in množi

Vhod: x, c, n

Izhod: z

1: $z = 1$
2: **for** $i = l - 1$ down to 0 **do**
3: $z = z^2 \bmod n$
4: **if** $c_i = 1$ **then**
5: $z = z \cdot x \bmod n$
6: **end if**
7: **end for**
8: **return** z

Če uporabljamo tak algoritem za potenciranje po modulu, je časovna zahtevnost Algoritma 5 reda $O\big(k \cdot \log^2 n \cdot \log\log n \cdot \log\log\log n\big)$, kjer je k število ponovitev algoritma, n pa število, ki ga testiramo.

5.1 Carmichaelova števila

V teoriji števil je Carmichaelovo število sestavljeno število $n > 0$, ki zadošča kongruenci $b^n \equiv b \pmod{n}$, za vsak b, kjer $1 < b < n - 1$ [27]. Ta števila so dobila ime po Robertu Carmichaelu. Praštevilo p zadošča kongruenci $b^p \equiv b \pmod{p}$ že po Fermatovem malem izreku. Carmichaelova števila so po tej lastnosti podobna praštevilom. Zato jim pravijo tudi Fermatova psevdopraštevila. Carmichaelova števila so pomembna, ker prestanejo Fermatov test za praštevilskost, čeprav sama niso praštevila. Ravno zaradi njih se ne moremo zanesti na ta test za dokazovanje praštevilskosti, (lahko pa ga uporabimo za dokazovanje sestavljenosti). Zato so testi, ki temeljijo na Fermatovem malem izreku, bolj tvegani kot testi, ki temeljijo na kaki drugi lastnosti.

Za testiranje praštevilskosti se v praksi najpogosteje uporabljajo verjetnostni **Monte Carlo algoritmi** s polinomsko časovno zahtevnostjo. Med temi algoritmi najdemo Solovay-Strassenov in Miller-Rabinov algoritem. To sta hitra algoritma.

Definicija 5.2 *Monte Carlo algoritem, ki je naklonjen odgovoru* **DA**, *je naključni odločitveni algoritem pri katerem je odgovor* **DA**, *vedno pravilen. Odgovor* **NE** *je lahko nepravilen. Rečemo, da ima tak algoritem verjetnost napake enako ϵ, če za poljuben vhod vrne napačen odgovor* **NE** *z verjetnostjo kvečjemu ϵ. Ta verjetnost se izračuna čez vse naključne izbire algoritma na izbranem vhodu.*

Najprej bomo opisali Solovay-Strassenov algoritem [32, str. 182-186]. Algoritem je Monte Carlo algoritem, naklonjen odgovoru **DA**, za testiranje sestavljenosti števil. Verjetnost napake je kvečjemu $\frac{1}{2}$. Če algoritem odgovori

DA, potem je n sestavljeno število. Algoritem proglasi sestavljeno število za praštevilo z verjetnostjo največ 2^{-k}, kjer je k število ponovitev algoritma.

Preden opišemo Solovay-Strassenov algoritem, je potrebno definirati še kvadratične ostanke po modulu n [32, str. 179], Legendrove simbole [32, str. 181] in Jacobijeve simbole [32, str. 181].

5.2 Kvadratični ostanki

Definicija 5.3 *Naj bo p liho praštevilo in a celo število. Pravimo, da je a kvadratični ostanek po modulu p, če je $a \not\equiv 0 \pmod{p}$ in ima kongruenca $y^2 \equiv a \pmod{p}$ rešitev $y \in \mathbb{Z}_p^*$.*

Naj bo p liho praštevilo in naj bo a kvadratični ostanek po modulu p. Potem obstaja $y \in \mathbb{Z}_p^*$, da je $y^2 \equiv a \pmod{p}$. Seveda je tudi $(-y)^2 \equiv a \pmod{p}$, saj je p lih. Naj bo $x^2 - a \equiv 0 \pmod{p}$. To kongruenco lahko zapišemo kot $(x-y)(x+y) \equiv 0 \pmod{p}$), kar pa je ekvivalentno $p \mid (x-y)(x+y)$, ker je p praštevilo ali $p \mid (x-y)$ ali $p \mid (x+y)$. Drugače zapisano, $x \equiv \pm y \pmod{p}$, torej obstajata dve rešitvi kongruence $x^2 - a \equiv 0 \pmod{p}$. Da določimo, če je neko število a kvadratični ostanek po modulu p, potrebujemo naslednji izrek.

Izrek 5.4 (Eulerjev izrek) *Naj bo p liho praštevilo. Število a je kvadratični ostanek po modulu p, če velja:*

$$a^{(p-1)/2} \equiv 1 \pmod{p}. \tag{5.2}$$

Kvadratične ostanke lahko v polinomskem času izračunamo s pomočjo algoritma Kvadriraj in množi. Časovna zahtevnost je $\mathrm{O}\big(\log^3 p\big)$.

5.3 Legendrovi simboli

Legendrove simbole bomo vpeljali zato, da bomo kasneje lahko definirali Jacobijeve simbole, slednje pa pri analizi Sollovay-Strassenovega algoritma.

Definicija 5.5 *Naj bo p liho praštevilo. Za vsako celo število a naj bo Legendrov simbol* $\left(\frac{a}{p}\right)$ *definiran takole:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{če je } a \equiv 0 \pmod{p}; \\ 1, & \text{če je } a \text{ kvadratični ostanek;} \\ -1, & \text{če } a \text{ ni kvadratični ostanek.} \end{cases} \tag{5.3}$$

Že prej smo videli, da je $a^{(p-1)/2} \equiv 1 \pmod{p}$, če je a kvadratični ostanek. Če je a večkratnik p, potem je očitno $a^{(p-1)/2} \equiv 0 \pmod{p}$. Če pa a ni kvadratični ostanek, potem je $a^{(p-1)/2} \equiv -1 \pmod{p}$, zato ker velja $(a^{(p-1)/2})^2 \equiv a^{(p-1)} \equiv 1 \pmod{p}$ in $a^{(p-1)/2} \not\equiv 1 \pmod{p}$. Zgornje definicije vodijo do učinkovitega izračuna Legendrovih simbolov.

5.4 Jacobijevi simboli

Jacobijevi simboli se od Legendrovih razlikujejo po tem, da je n poljubno liho število.

Definicija 5.6 *Naj bo n liho pozitivno število. in naj bo po osnovnem izreku*

$$n = \prod_{i=1}^{k} p_i^{e_i}. \tag{5.4}$$

Naj bo a celo število. Jacobijev simbol $\left(\frac{a}{n}\right)$ *je definiran kot:*

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}. \tag{5.5}$$

Predpostavimo, da je $n > 1$ lih. Če je n praštevilo, potem je tako kot prej $\left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}$ za vsak a. Če pa n ni praštevilo, potem prejšnja enakost lahko drži ali pa ne.

Lastnosti Jacobijevih simbolov

1. Če je n pozitivno liho število in $m_1 \equiv m_2 \pmod{n}$, potem je

$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right). \tag{5.6}$$

2. Če je $m_1 \perp m_2, (\text{tj. } \gcd(m_1, m_2) = 1)$, potem velja

$$\left(\frac{m_1 \cdot m_2}{n}\right) = \left(\frac{m_1}{n}\right) \cdot \left(\frac{m_2}{n}\right), \tag{5.7}$$

   in če je $m = 2^k \cdot m_1$ ter m_1 lih, potem je

$$\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \cdot \left(\frac{m_1}{n}\right). \tag{5.8}$$

3. Velja

$$\left(\frac{2}{n}\right) = \begin{cases} 1, & \text{če je } n \equiv \pm 1 \pmod{8}; \\ -1, & \text{če je } n \equiv \pm 3 \pmod{8}. \end{cases} \tag{5.9}$$

4. Če sta m, n liha, potem velja

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{m}{n}\right), & \text{če je } m, n \equiv 3 \pmod{4}\ ; \\ \left(\frac{n}{m}\right), & \text{sicer.} \end{cases} \tag{5.10}$$

5.5 Solovay-Strassenov algoritem

Algoritem 7 Algoritem Sollovay-Strassen

Vhod: število n

izberi naključno število a iz intervala $[1, n-1]$
$x = \left(\frac{a}{n}\right)$
if $x = 0$ **then**
    return "n je sestavljeno število" (in zaključi)
end if
$y = a^{(n-1)/2} \pmod{n}$
if $x \equiv y \bmod n$ **then**
    return "n je praštevilo" (in zaključi)
else
    return "n je sestavljeno število" (in zaključi)
end if

To je Monte Carlo algoritem, ki je naklonjen odgovoru **DA**, ko je n sestavljeno število. Verjetnost napake je največ $\frac{1}{2}$. Ali ima algoritem polinomsko časovno zahtevnost? Vemo, kako izračunati $a^{(n-1)/2} \pmod{n}$ v času $\mathrm{O}\big(\log^3 n\big)$. Ker poznamo lastnosti Jacobijevih simbolov, ni potrebna faktorizacija pri izračunu $\left(\frac{a}{n}\right)$. To lahko naredimo v polinomskem času. Vse, kar je treba storiti, je modularna redukcija in faktorizacija z osnovo 2. Če je število v binarni obliki, potem je faktorizacija z osnovo 2 le dodajanje ustreznega števila ničel na konec binarnega niza. Torej je časovna zahtevnost odvisna od števila modularnih redukcij. Potrebnih je največ $\mathrm{O}\big(\log n\big)$ modularnih redukcij, za vsako pa porabimo $\mathrm{O}\big(\log^2 n\big)$ časa. Torej je časovna zahtevnost $\mathrm{O}\big(\log^3 n\big)$, kar je polinomsko mnogo glede na $\log n$.

5.5.1 Verjetnost napake algoritma

Da pokažemo, da je verjetnost napake Solovay-Strassenovega algoritma kvečjemu $\frac{1}{2}$, moramo pokazati, da je $G(n) = \{a \in \mathbb{Z}_n^* : \left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}\}$ podgrupa grupe $\mathbb{Z}_n^*$, in da velja $\mid G(n) \mid \leq \frac{n-1}{2}$, če $G(n) \neq \mathbb{Z}_n^*$.

Dokaz. Če je $G(n) \neq \mathbb{Z}_n^*$, po Langrangevem izreku velja

$$\mid G(n) \mid \leq \frac{\mid \mathbb{Z}_n^* \mid}{2} \leq \frac{n-1}{2}.$$

Naj bosta $a, b \in G(n)$, za katera velja

$$\left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod{n}, \text{in}$$
$$\left(\frac{b}{n}\right) = b^{(n-1)/2} \pmod{n}.$$

Zanima nas, ali je produkt $a \cdot b$ tudi v $G(n)$.

Iz lastnosti Jacobijevih simbolov sledi

$$\left(\frac{a \cdot b}{n}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{b}{n}\right) \equiv a^{(n-1)/2} \cdot b^{(n-1)/2} \pmod{n} \equiv (a \cdot b)^{(n-1)/2} \pmod{n},$$

$$\text{zato je } a \cdot b \in G(n),$$

saj je $G(n)$ podmnožica multiplikativne končne grupe in je zaprta za operacijo množenja. Torej je $G(n)$ podgrupa grupe $\mathbb{Z}_n^*$.

Pokažimo še, da obstaja vsaj en element v $\mathbb{Z}_n^*$, ki ne pripada $G(n)$.
Naj bo $n = p^k \cdot q$, kjer je p praštevilo, q liho število in $k \geq 2$. Seveda je $\gcd(p, q) = 1$. Naj bo $a = 1 + p^{k-1} \cdot q$. Potem je $\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)^k \cdot \left(\frac{a}{q}\right) = 1$.

Če upoštevamo binomski izrek, dobimo:

$$a^{(n-1)/2} = \sum_{i=0}^{(n-1)/2} \binom{(n-1)/2}{2} \cdot (p^{k-1} \cdot q)^i \equiv 1 + \frac{n-1}{2} \cdot p^{k-1} \cdot q \pmod{n}$$

(ker je $k \geq 2$, so vsi ostali členi enaki 0 po modulu n). Če je

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n},$$
$$\text{sledi } \frac{n-1}{2} \cdot p^{k-1 \cdot q} \equiv 0 \pmod{n},$$

$$\text{in od tod } p^q \cdot q \mid \frac{n-1}{2} \cdot p^{k-1} \cdot q \Longrightarrow p \mid \frac{n-1}{2} \Longrightarrow n \equiv 1 \pmod{p}.$$

Toda to je v protislovju z dejstvom, da je $n \equiv 0 \pmod{p}$. Torej $a \in \mathbb{Z}_n^*$ ne pripada $G(n)$. Zato je $\mid G(n) \mid \leq \frac{n-1}{2}$.
Naj bo n sestavljeno število. Če je $a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$, potem je

$$\gcd(a, n) \neq 1 \text{ in zato } \left(\frac{a}{n}\right) \equiv 0.$$

V tem primeru algoritem vedno vrne pravilen odgovor. Če pa je $a \in \mathbb{Z}_n^*$, potem

$$\gcd(a, n) \neq 1,$$

algoritem vrne napačen odgovor le, če je $a \in G(n)$. Pokazali smo, da je $\mid G(n) \mid \leq \frac{n-1}{2}$. Verjetnost napake je $\frac{|\mathbb{Z}_n^*|}{n-1} \cdot \frac{|G(n)|}{|\mathbb{Z}_n^*|} \leq \frac{1}{2}$. □

5.6 Miller-Rabinov algoritem

Miller-Rabinov algoritem [32, str. 186-188] je Monte Carlo algoritem, ki je naklonjen odgovoru **DA**, za odločanje, ali je sestavljeno število. Gre za polinomski algoritem. Analiza pokaže, da je njegova časovna zahtevnost $O(\log n^3)$, tako kot pri Sollovay-Strassenovem algoritmu. Vendar se izkaže, da je v praksi boljši Miller-Rabinov algoritem. Pokazali bomo, da algoritem ne more odgovoriti, da je n sestavljeno število, če je n v resnici praštevilo.

Algoritem 8 Algoritem Miller-Rabin

Vhod: število n

1: zapišemo n kot $n = 2^k \cdot m + 1$, kjer je m liho število
2: izberi naključno število a iz intervala $[1, n-1]$
3: $b = a^m \bmod n$
4: **if** $b \equiv 1 \pmod{n}$ **then**
5: **return** "n je praštevilo" (in zaključi)
6: **end if**
7: **for** $i = 0$ to $k - 1$ **do**
8: **if** $b \equiv -1 \pmod{n}$ **then**
9: **return** "n je praštevilo" (in zaključi)
10: **else**
11: $b = b^2 \bmod n$
12: **end if**
13: **end for**
14: **return** "n je sestavljeno število" (in zaključi)

Izrek 5.7 *Miller-Rabinov algoritem je Monte Carlo algoritem, naklonjen odgovoru* **DA**, *za odločanje, ali je n sestavljeno število.*

Dokaz. Dokažimo zgornji izrek s protislovjem. Recimo, da je n praštevilo, algoritem pa je vrnil odgovor "n je sestavljeno število". To bi pomenilo, da

se algoritem na 4. koraku ne konča, zato $a^m \not\equiv 1 \pmod{n}$. Ker se b kvadrira v vsaki iteraciji zanke **for**, preverjamo vrednosti $a^m, a^{2m}, \ldots, a^{2^{k-1}m}$. Torej velja $a^{2^i m} \not\equiv -1$ za $i = 0, 1, \ldots, k-1$. Če uporabimo Mali Fermatov izrek, dobimo

$$a^{2^k m} \equiv 1 \pmod{n},$$

ker je $n - 1 = 2^k m$ in $n \in \mathbb{P}$. Če uporabimo še Eulerjev izrek, dobimo, da je $a^{(n-1)/2} = a^{2^{k-1}m}$ kvadratični koren enote v $\mathbb{Z}_n^*$. Če je n praštevilo, sta v $\mathbb{Z}_n$ natanko dva korena enote. To sta 1 in -1. (Zakaj? $x^2 - 1 = 0$). Polinomi imajo kvečjemu toliko ničel, kot je njihova stopnja. Zato ima $x^2 - 1 = 0$ kvečjemu dve ničli v $\mathbb{Z}_n$ ($1^2 = 1$ in $(-1)^2 = 1$). Nadaljujmo:
če $a^{2^{k-1}m} \not\equiv -1$ torej velja $a^{2^{k-1}m} \equiv 1 \pmod{n}$ in tudi $a^{2^{k-2}m} \equiv 1 \pmod{n}$, če to ponavljamo dobimo, da je $a^m \equiv 1$, kar pa je v protislovju s predpostavko. Algoritem bi torej vrnil odgovor "n je praštevilo." □

Verjetnost napake algoritma je $\frac{1}{4}$.

5.7 Lucasov algoritem

To je praštevilski test za naravno število n, pod pogojem, da poznamo praštevilske delitelje števila $n - 1$ [17]. Naj bo n pozitivno število. Denimo, da obstaja število a, $1 < a < n$, za katerega velja

$$a^{n-1} \equiv 1 \pmod{n} \tag{5.11}$$

in za vsak q, ki je praštevilski faktor $n - 1$, velja še

$$a^{(n-1)/q} \not\equiv 1 \pmod{n}. \tag{5.12}$$

Potem je n praštevilo. Če takega a ni, potem je n bodisi 1 ali pa sestavljeno število.

Pravilnost algoritma

Če enakost (5.11) drži, potem lahko sklepamo, da sta a in n tuji števili. Če velja še (5.12), potem je $\text{or}(a) = n-1$, kar pomeni, da je n praštevilo. Obratno: naj bo n praštevilo. Potem obstaja primitivni koren po modulu n oz. obstaja generator g grupe $\mathbb{Z}_n$. Red $\text{or}(g)$ je $n-1$ ter (5.11) in (5.12) držita. Opomba: če obstaja tak $a < n$, pri katerem (5.11) ne drži, potem rečemo, da je a *Fermatova priča* za sestavljenost n.

Algoritem 9 Lucasov algoritem

Vhod: število $n > 2$, parameter k število korakov algoritma

ponovi k-krat
izberi naključno število a iz intervala $[2, n-1]$
if $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
  return "n je sestavljeno" (in zaključi)
else
  za $\forall q$, kjer je q praštevilski delitelj $n-1$
  if $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ **then**
    if to drži za vse q **then**
      return "n je praštevilo" (in zaključi)
    end if
  end if
end if
return "n je verjetno sestavljeno število" (in zaključi)

5.8 Mersennova števila

Mersennova števila[2] so praštevila oblike $M_n = 2^n - 1$. Če je n sestavljeno število, potem je tudi $2^n - 1$ sestavljeno število. Mersennova števila tvorijo mnogo velikih praštevil, kar je zelo pomembno.[3] Za testiranje Mersennovih števil in njihove praštevilskosti se uporablja Lucas-Lehmerjev algoritem, ki bo opisan v nadaljevanju.

5.9 Lucas-Lehmerjev algoritem

Lucas-Lehmerjev algoritem za testiranje Mersennovih števil povzamemo po [6]. Test je zasnoval Edoard Lucas 1856. Izboljšal ga je najprej sam Lucas leta 1878 in nato še Derrich Henry Lehmer 1930. Zato ime Lucas-Lehmer.
Naj bo $M_p = 2^p - 1$ Mersennovo število, ki ga testiramo s pomočjo lihega praštevila p. Ker je p eksponentno manjši od M_p, lahko uporabimo algoritem deljenj. Definirajmo število s_i, $i \geq 0$, takole:

$$s_i = \begin{cases} 4, & \text{če je } i = 0; \\ s_{i-1}^2 - 2, & \text{sicer.} \end{cases} \tag{5.13}$$

Sledi, da je M_p praštevilo natanko tedaj, ko je $s_{p-2} \equiv 0 \pmod{M_p}$. Številu $s_{p-2} \pmod{M_p}$ rečemo Lucas-Lehmerjev ostanek p.

[2]Šele v 17. stoletju so začeli preučevati ta števila, ko je o njih napisal knjigo Marin Mersenne z naslovom Cogita physica mathematica (1644).

[3]25. 01. 2013 so izračunali Mersennovo število s 17.425.170 števkami.

Algoritem 10 Lucas-Lehmerjev algoritem

Vhod: število $p > 2$

1: $s = 4$
2: $M_p = 2^p - 1$
3: **for** $i = 1$ to $p - 2$ **do**
4: $\quad s_i = s_{i-1}^2 \pmod{M_p}$
5: $\quad$ **if** $s_i = 0$ **then**
6: $\quad\quad$ **return** "p je praštevilo" (in zaključi)
7: $\quad$ **else**
8: $\quad\quad$ **return** "p je sestavljeno število" (in zaključi)
9: $\quad$ **end if**
10: **end for**

Časovna zahtevnost

V algoritmu 10 sta dve zahtevni računski operaciji. V vsaki iteraciji se računata produkt $s \cdot s$ in modularna redukcija po modulu M_p. Modularno redukcijo se lahko enostavno reši z uporabo lastnosti $k \equiv (k \bmod 2^n) + \lfloor k/2^n \rfloor \pmod{2^n - 1}$. Drugače zapisano, če vzamemo n najmanj pomembnih bitov izmed k in to ponavljamo dokler ne ostane n bitov, lahko izračunamo ostanek pri deljenju k z Mersennovim številom, brez deljenj. Tako da je to zanemarljivo glede na računanje $s \cdot s$. Množenje zahteva $\mathrm{O}(p^2)$ bitnih operacij za izračun kvadrata. Ker je potrebnih p iteracij, je časovna zahtevnost $\mathrm{O}(p^3)$. Obstaja izboljšava, kjer je časovna zahtevnost $\mathrm{O}\left(p \log 2^{\mathrm{O}\left(log^* p\right)}\right)$[4] [6].

[4] V računalništvu je $\log^* n$ število, ki pove kolikokrat moramo logaritmirati n, da bo rezultat ≤ 1.

Poglavje 6

Deterministični algoritmi

6.1 Algoritem AKS

Algoritem AKS za dokazovanje praštevilskosti je poznan tudi pod imenom algoritem Agrawal-Kayal-Saxene [2]. To je deterministično algoritem. Sestavili in objavili so ga Manindra Agrawal, Neeraj Kayal in Nitin Saxena.[1] Algoritem v polinomskem času pravilno odloči ali gre za sestavljeno število ali za praštevilo. Gre za ogromno odkritje na področju teoretičnega računalništva, saj so bili pred tem znani le verjetnostni algoritmi in nekaj deterministčnih algoritmov, pri katerih bodisi časovna zahtevnost ni bila polinomska ali pa se ni dalo dokazati, da pri vseh vhodih algoritem res vrne pravilen odgovor. Algoritem AKS določi za poljubno število n ali gre za sestavljeno število ali za praštevilo.

Algoritem ima polinomsko časovno zahtevnost glede na število števk števila n. Če mu podamo veliko število n, bo v času $\mathrm{O}\big(\log^{\mathrm{O}(1)} n\big)$ vrnil pravilen odgovor, ali je dano število n praštevilo. Izhodišče je posplošitev Fermatovega malega izreka za polinome.

Izrek 6.1 *Za naravni števili a in n, kjer je $a \perp n$, velja, da je n praštevilo*

[1] Trije indijski računalničarji so ga 6. avgusta 2002 objavili v članku z naslovom Praštevila so v P.

natanko tedaj, ko velja naslednja kongruenca:

$$(X-a)^n \equiv (X^n - a) \pmod{n}. \tag{6.1}$$

Dokaz. ($\Rightarrow$) Naj bo

$$w(X) = (X-a)^n - (X^n - a). \tag{6.2}$$

Če upoštevamo binomske koeficiente po razvitju binoma $(X-a)^n$ dobimo

$$(X-a)^n = \sum_{k=0}^{n} \binom{n}{k} X^k \cdot (-a)^{n-k} = \sum_{k=0}^{n} \binom{n}{k} \cdot (-1)^{n-k} \cdot X^k \cdot (-a)^{n-k}$$

in zato

$$\begin{aligned} w(X) &= \sum_{k=0}^{n} \binom{n}{k} \cdot (-1)^{n-k} \cdot X^k \cdot (-a)^{n-k} - X^n + a \\ &= \sum_{k=1}^{n-1} \binom{n}{k} \cdot (-1)^{n-k} \cdot x^k \cdot (-a)^{n-k} + (-1)^n a^n + a. \end{aligned}$$

Naj bo $n \in \mathbb{P}$. Spomnimo se, da je $\binom{n}{k} = \frac{n\cdot(n-1)\cdot(n-2)\cdots(n-k+1)}{k!}$, kjer je $0 < k < n$. Ker je $k < n$ in n praštevilo, $k!$ ne deli n, zato je člen $\binom{n}{k} \cdot (-1)^{n-k} a^{n-k} X^k$ večkratnik n. To pa pomeni, da je celotna vsota enaka 0. Ostane nam še

$$(-1)^n \cdot a^n + a = (-a)^n + a,$$

kar lahko zapišemo kot

$$a^n = a \pmod{n},$$

tako da res dobimo, da je $w(X) = 0$.

($\Leftarrow$) Sedaj pa predpostavimo nasprotno. Naj bo n sestavljeno število. Naj bo $p \in \mathbb{P}$ in naj $p^j \mid n$, kjer je $j \in \mathbb{N}$ in $p^{j+1} \nmid n$.
Ker je $\binom{n}{p} = \frac{n\cdot(n-1)\cdots(n-p+1)}{p\cdot(p-1)\cdots 2}$, velja $p^{j-1} \mid \binom{n}{p}$ in $p^j \nmid \binom{n}{p}$, zato n ne deli $\binom{n}{p}$. Ker je $n \perp a$, je $\gcd(n, a^{n-p}) = \gcd(n, 1)$, zato koeficient $(-1)^{n-p} a^{n-p} X^p$ ni deljiv z n. $\square$

Na prvi pogled bi ta izrek že lahko bil test za praštevilskost. Da izračunamo n koeficientov na levi strani, lahko uporabimo algoritem Kvadriraj in množi. Vendar je časovna zahtevnost tega postopka eksponentna glede na n. Ideja je, da obe strani delimo s polinomom $X^r - 1$ in potem pri ostanku koeficiente reduciramo še po modulu n.[2] Tako dobimo

$$(X - a)^n = X^n - a \pmod{X^r - 1, n}. \tag{6.3}$$

Če r ni prevelik, levo stran lahko hitro izračunamo, saj so vsi polinomi reducirani na stopnjo, nižjo od r. Če je $r = \mathrm{O}\big(\log^{\mathrm{O}\left(1\right)} n\big)$, je algoritem časovne zahtevnosti $\mathrm{O}\big(\log^{\mathrm{O}\left(1\right)} n\big)$. Lahko se zgodi, da (6.3) drži, (6.1) pa ne. Če za vsa praštevila n velja (6.1), potem gotovo velja tudi (6.3). Obratno pa ni nujno res. Namreč, če je n sestavljeno število, potem lahko (6.3) velja tudi za nekatere izbrane a in r. Torej obstaja par (a, r), pri katerem bi algoritem odgovoril napačno. Torej je potrebno ponoviti test na nekaj omejenih vrednostih a. V svojem članku Agrawal, Kayal in Saxena opisujejo, da moramo testirati za take a, kjer velja $1 \leq a \leq 2\sqrt{r} \log n$.

[2] To zapišemo kot $\pmod{X^r - 1, n}$.

Algoritem 11 Algoritem AKS

Vhod: število $n > 1$

1: **if** n je oblike $a^b, b > 1$ **then**
2: **return** "n je sestavljeno število" (in zaključi)
3: **end if**
4: $r = 2$
5: **while** $(r < n)$ **do**
6: **if** $gcd(n, r) \neq 1$ **then**
7: **return** "n je sestavljeno število" (in zaključi)
8: **end if**
9: **if** r je praštevilo **then**
10: naj bo q največji praštevilski faktor števila $r - 1$
11: **if** $(q \geq 4\sqrt{r}\log n)\&(n^{\frac{r-1}{2}} \not\equiv 1 \pmod{r})$ **then**
12: **break;**
13: **end if**
14: $r = r + 1$
15: **end if**
16: **end while**
17: **for** $a = 1$ to $2\sqrt{r}\log n$ **do**
18: **if** $(X - a)^n \not\equiv (X^n - a)(\pmod{X^r - 1, n})$ **then**
19: **return** "n je sestavljeno število" (in zaključi)
20: **end if**
21: **end for**
22: **return** "n je praštevilo" (in zaključi)

Izrek 6.2 *Algoritem AKS odgovori "praštevilo", le če je n res praštevilo* [2].

Na algoritem AKS lahko gledamo kot na nekakšen filter, ki loči sestavljena števila od praštevil. Če gre za sestavljeno število, algoritem to ugotovi v vrsticah 1, 6 in 18. Do vrstice 22 pride le v primeru, ko je n praštevilo. Zanka while služi temu, da najdemo najmanjši r, ki bo ustrezen glede na n.

Lema 6.3 *Obstajata konstanti c_1 in c_2, za kateri je praštevilo r na intervalu $[c_1 \log n^6, c_2 \log n^6]$ in ima $r-1$ praštevilski faktor q, za katerega velja $q \geq 4\sqrt{r} \log n$ ter $q \mid or_r(n)$* [2, str. 4].

V zanki se vrstica 12 ne more izvesti več kot $c_2(\log n)^6$-krat, potem ko r doseže n. To zagotavlja polinomsko časovno zahtevnost.

Nekaj lastnosti algoritma:

- če je n oblike a^b, $b > 1$, vrne odgovor "n je sestavljeno število"
- če je n sestavljeno število, toda ni oblike a^b ali pa je n praštevilo, potem
  - če ne obstaja tak r da velja $q \geq 4\sqrt{r} \log n$ in $n^{\frac{r-1}{q}} \not\equiv \pmod{r}$
    - n je sestavljeno število (izvedeta se vrstici 6 in 7 in algoritem vrne odgovor "n je sestavljeno število")
    - n je praštevilo. Zanka while se izvaja, dokler ni $r = n$. For zanka pa se izvaja, dokler ni $a = 2\sqrt{r} \log n$ (algoritem vrne odgovor "n je praštevilo"v vrstici 22)
  - obstaja tak r da velja $q \geq 4\sqrt{r} \log n$ in $n^{\frac{r-1}{q}} \not\equiv \pmod{r}$
    - n je sestavljeno število
      - algoritem bodisi vrne odgovor "n je sestavljeno število"(vrstica 7)
      - ali pa vrne odgovor "n je sestavljeno število"(vrstica 19)
    - n je praštevilo. Zanka while najde ustrezen r, ki je manjši kot n. Izvede se vrstica 12. Algoritem nadaljuje s for zanko,

ta se zaključi, ko dobi a vrednost $2\sqrt{r}\log n$. Algoritem vrne odgovor "n je praštevilo"(vrstica 22).

Časovna zahtevnost algoritma je asimptotično $\mathrm{O}\big(\log^{12} n\big)$.

6.2 Izboljšave algoritma AKS

Od leta 2002 naprej so se znanstveniki trudili izboljšati prvotni algoritem AKS. Nastale so različne variante. Osnovna ideja je ostala ista, le da so poskušali izboljšati red r. Dokazi pravilnosti algoritma pa so postajali vedno bolj enostavni in razumljivejši. Osredotočili so se na:

- kako zmanjšati število ponovitev for zanke (vrstica 17)
- iskanje bolj učinkovitega načina za izračun r

6.2.1 Lenstra-Pomerancova izboljšava

Lenstra je s pomočjo Pomeranca našel izboljšavo AKS algoritma [18, str. 13] kmalu potem, ko je ta izšel v članku. Pomerance je ugotovil, da ni potrebno, da je q največji praštevilski delitelj $r-1$. Razlika s prvotnim algoritmom je konstrukcija števila r. Ta modifikacija zmanjša število iteracij v for zanki. Časovna zahtevnost algoritma pa je $\mathrm{O}\big(\log^6 n\big)$.

Algoritem 12 Algoritem Lenstra-Pomerance

Vhod: število $n > 1$

if n je oblike $a^b, b > 1$ **then**
  return "n je sestavljeno število" in algoritem se ustavi
end if
$r = 2$
while $(r < n)$ **do**
  if $gcd(n, r) \neq 1$ **then**
    return "n je sestavljeno število" in algoritem se ustavi
  end if
  if $\mathrm{or}_r(n) > \lfloor \log^2 n \rfloor$ **then**
    break;
    $r = r + 1$
  end if
end while
for $a = 1$ to $\phi(r) - 1$ **do**
  if $(X - a)^n \not\equiv (X^n - a)\,(\pmod{X^r - 1, n})$ **then**
    return "n je sestavljeno število" in algoritem se ustavi
  end if
end for
return "n je praštevilo" in algoritem se ustavi

6.2.2 Bernsteinova izboljšava

Daniel Bernstein je po zgledu Lenstre zasnoval svojo verzijo algoritma AKS [15, 18].

Algoritem 13 Bernstein algoritem

Vhod: število $n > 1$

1: **if** n je oblike $a^b, b > 1$ **then**
2: **return** "n je sestavljeno število" in algoritem se ustavi
3: **end if**
4: $r = 2$
5: **while** $(r < n)$ **do**
6: **if** $gcd(n, r) \neq 1$ **then**
7: **return** "n je sestavljeno število" in algoritem se ustavi
8: **end if**
9: **if** r je praštevilo **then**
10: naj bo q največji praštevilski faktor $r - 1$
11: **if** $\binom{2q-1}{q} \geq 2^{2\lfloor\sqrt{r}\rfloor \log n}$ in $n^{\frac{r-1}{q}} \leq 1$ **then**
12: **break;**
13: **end if**
14: $r = r + 1$
15: **end if**
16: **end while**
17: **for** $a = 1$ to $2\sqrt{r}\log n$ **do**
18: **if** $(X - a)^n \not\equiv (X^n - a)\,(\pmod{X^r - 1, n})$ **then**
19: **return** "n je sestavljeno število" in algoritem se ustavi
20: **end if**
21: **end for**
22: **return** "n je praštevilo" in algoritem se ustavi

Algoritem spremeni način konstrukcije r in zato tudi zmanjša število iteracij for zanke.

6.2.3 Algoritem AKS verzija 6

Leta 2004 so Agrawal, Kayal in Saxena opisali novo verzijo algoritma AKS. Izboljšali so red r.

Algoritem 14 Algoritem AKS verzija 6

Vhod: število $n > 1$

1: **if** n je oblike $a^b, b > 1$ **then**
2: **return** "n je sestavljeno število" in algoritem se ustavi
3: **end if**
4: Poišči najmanjši r tako, da velja: $\mathrm{or}_r(n) > \log^2 n$
5: **if** $1 < gcd(a, n) < n$ za nek $a \leq r$ **then**
6: **return** "n je sestavljeno število" in algoritem se ustavi
7: **end if**
8: **if** $n \leq r$ **then**
9: **return** "n je praštevilo" in algoritem se ustavi
10: **end if**
11: **for** $a = 1 \ldots \lfloor \sqrt{\phi(r)} \log n \rfloor$ **do**
12: **if** $(X - a)^n \not\equiv (X^n - a) (\pmod{X^r - 1, n})$ **then**
13: **return** "n je sestavljeno število" in algoritem se ustavi
14: **end if**
15: **end for**
16: **return** "n je praštevilo" in algoritem se ustavi

Izrek 6.4 *Algoritem 14 odgovori "n je praštevilo" natanko tedaj, ko je n praštevilo.*

Zgornji izrek bomo dokazali s pomočjo naslednjih lem.

Lema 6.5 *Če je n praštevilo, algoritem 14 vrne odgovor "n je praštevilo".*

Dokaz. Če je n praštevilo, potem vrstice 12,13 in 5,6 nikoli ne odgovorijo, da je n sestavljeno število. Zaradi izreka (6.1) tudi vrstici 12,13 ne moreta vrniti odgovora "n je sestavljeno število", če gre za praštevilo. Algoritem

mora določiti bodisi v vrsticah 8, 9 bodisi v vrstici 16 ali je res praštevilo. Če algoritem v vrstici 9 vrne "praštevilo", potem je n praštevilo, sicer bi v 5. vrstici dobili netrivialen faktor števila n. Torej nas zanima le še vrstica 16. Algoritem ima dva ključna koraka. Vrstico 4 in 12. Vrstica 4 je pomembna zato, da najdemo ustrezen r, vrstica 12 pa zato, da se preveri enačbe na številu a.

Lema 6.6 *Naj LCM(m) označuje lcm prvih m števil. Če je* $m \geq 7$*, potem je LCM(m)*$\geq 2^m$ [19, str. 34].

Da dokažemo izrek 6.4 moramo najprej pokazati da obstaja število $r \in \mathbb{N}$, za katero velja:

- $r \leq max\{3, \lceil \log^5 n \rceil\}$
- $\gcd(r, n) = 1$
- $\mathrm{or}_r(n) > \log^2 n$

Lema 6.7 *Obstaja* $r \leq max\{3, \lceil \log^5 n \rceil\}$*, tako da velja* $or_r(n) > \log^2 n$.

Če je $n = 2$ in $r = 3$ je izpolnjena lema 6.7, saj imamo $2^2 = 4 \equiv 1 \pmod 3$, $\mathrm{or}_3(2) = 2$ in $\log^2 2 = 1$ iz tega sledi, da je res $\mathrm{or}_3(2) > \log^2 2$.
Zato lahko predpostavimo, da je $n > 2$. Naj bo $B = \lceil \log^5 n \rceil$. Po lemi 6.6 je $\mathrm{LCM}(B) \geq 2^B$. Najprej s protislovjem pokažimo, da ne obstaja tak $r \leq B$, ki bi delil produkt

$$N = n^{\lfloor \log B \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1).$$

Naj vsak r, $1 \leq r \leq B$, velja $r \mid N$, potem je $\mathrm{LCM}(B) \leq N$, ker je N zmnožek vseh števil $\leq B$.

$$\begin{aligned}
N &= n^{\lfloor \log B \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1) \\
&< n^{\lfloor \log B \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} n^i \\
&= n^{\lfloor \log B \rfloor + 1 + 2 + \cdots \log^2 n} \\
&= n^{\lfloor \log B \rfloor + \frac{1}{2} \log^2 n (\log^2 n + 1)} \\
&< n^{\log^4 n} \\
&= (2^{\log n})^{\log^4 n} \\
&= 2^{\log^5 n} \\
&\leq 2^B
\end{aligned}$$

Pri izpeljavi smo v tretji vrstici uporabili vsoto $\sum_{i=1}^{\log^2 n} i = \frac{\log^2 n(\log^2 n+1)}{2}$, pri vrstici 6 pa $n = 2^{\log n}$.

Dobili smo, da je $N < 2^B$. Lema 6.6 pa pravi, da je LCM$(B) \geq 2^B$. To je protislovje. Torej množica števil med 1 in B, ki ne deli N, ne more biti prazna. Naj bo r najmanjši element take neprazne množice.

Sedaj pokažimo, da je $\gcd(r, n) = 1$. Naj bo $r = ab$. Število a naj bo sestavljeno iz istih praštevilskih faktorjev kot r. Naj $r \mid n$, število b pa naj bo sestavljeno iz prafaktorjev različnih od števila a. Potem je $\gcd(b, n) = 1$. Opazimo, da potenca kateregakoli prafaktorja a ne more presegati $\lfloor \log B \rfloor$, drugače bi bil $a > B$. Ker so isti praštevilski faktorji prisotni tako pri a kot pri n, sledi, da $a \mid n^{\lfloor \log B \rfloor}$.

Od tod sledi, da število $b \nmid \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1)$, ker bi $r = ab$ delil N; mi pa smo izbrali tak r, da ta ne deli N. Velja pa še $b \nmid n^{\lfloor \log n \rfloor}$, ker je $\gcd(n, b) = 1$. Torej $b \nmid N$. Vemo, da je r najmanjše tako število, ki ne deli N, ker je $r = ab$, mora biti $b \leq r$. Iz zapisanega sledi, da je $b = r$. Ker je $\gcd(b, n) = 1$ in $b = r$, sledi $\gcd(r, n) = 1$.

Pokazati moramo še, da $\mathrm{or}_r(n) > \log^2 n$. Predpostavimo nasprotno:

$$\mathrm{or}_r(n) = d \leq \log^2 n.$$

Po definiciji je: $n^d \equiv 1 \pmod{r}$, če damo 1 na levo stran dobimo $n^d - 1 \equiv 0 \pmod{r}$ in iz tega sledi $r \mid (n^d - 1)$ za vrednost $d \leq \log^2 n$. Toda potem bi r delil kak faktor od N. Vemo, da $r \nmid N$, zato je $\mathrm{or}_r(n) > \log^2 n$. □

Ker je $\mathrm{or}_r(n) > 1$, ima n praštevilski faktor p, tako da velja $\mathrm{or}_r(p) > 1$. Veljati mora še $p > r$, sicer bi se algoritem ustavil v vrstici 5 ali 8. Osredotočimo se sedaj na vrstico 12.
Naj bo $l = \lfloor \sqrt{\phi(r)} \log n \rfloor$. Ta vrstica preverja, če velja $(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}$, za $1 \leq a \leq l$.

Definicija 6.8 *Naj bo $f \in \mathbb{Z}[x]$ in $m \in \mathbb{N}$. Rečemo, da je m introspektivno število za f, če velja $f(x)^m \equiv f(x^m) \pmod{x^r - 1, p}$. Če sta n in p introspektivni števili za linearen polinom $(x + a)$, potem je tudi $\frac{n}{p}$ introspektivno za $(x + a)$.*

Lema 6.9 *Naj bo $n \in \mathbb{N}$ in p praštevilski faktor števila n, $a \in \mathbb{N}, 0 \leq a \leq l$. Če sta n in p introspektivni števili za $(x + a)$, potem je tudi $\frac{n}{p}$ introspektivno za $(x + a)$.* [19, str. 25]

Množica introspektivnih števil je zaprta za množenje.

Lema 6.10 *Naj bosta $f, g \in \mathbb{Z}[x]$, $s, t \in \mathbb{N}$. Potem velja:*

- *če sta s in t introspektivni števili za f, potem je tudi $s \cdot t$ introspektivno število za f;*
- *če je s introspektivno za f in g, potem je introspektivno tudi za $f \cdot g$* [19, str. 26]

Posledica leme 6.9 in 6.10 je to, da je množica $I = \{(\frac{n}{p})^i p^j : i, j \geq 0\}$ introspektivna za vsak polinom iz množice $P = \{\prod_{a=0}^{l} (x + a)^{e_a} : e_a \geq 0\}$.
Definirali bomo dve grupi na teh dveh množicah.

Definicija 6.11 *Označimo z I_r množico, definirano z* $I_r = \{i \pmod r : i \in I\}$ [19, str. 27].

I_r je multiplikativna podgrupa grupe $\mathbb{Z}_r^*$. Naj bo $|I_r| = t$. Ker je $\mathrm{or}_r(n) > \log^2 n$, sledi $t > \log^2 n$. Da definiramo drugo grupo pa potrebujemo najprej definicijo ciklotomičnih polinomov.

Ciklotomični polinomi

Naj bo $n \in \mathbb{Z}$, $n > 0$. Potem n-ti ciklotomični polinom spremenljivke X, označimo z $\Phi_n(X)$ [9, str. 5]. To je polinom, ki ima natanko n primitivnih korenov enote. Zapišemo ga kot:

$$\Phi_n(X) = \prod_{\zeta^n=1} (X - \zeta).$$

Nadaljujmo z našim dokazom. Naj bo $\Phi(x)$ r-ti ciklotomični polinom nad poljem $\mathbb{F}_p$. Potem $\Phi(x)|(x^r - 1)$ na nerazcepne faktorje stopnje $\mathrm{or}_r(p)$. Naj bo $h(x)$ tak faktor. Ker je $\mathrm{or}_r(p) > 1$ je $\deg(h(x)) > 1$.

Druga grupa, ki jo bomo definirali, je množica ostankov polinomov iz P pri deljenju s $h(x)$ in deljenju s številom p. Naj bo G ta grupa.
$P = \{f \pmod{h(x), p}\colon\ f \in G\}$. To grupo generirajo linearni polinomi $x, x + 1, x + 2, \ldots, x + l$ polja $\mathbb{F} = \mathbb{F}_p[x]/(h(x))$.

Lema 6.12 $|G| \geq \binom{t+l}{t-1}$. Dokaz: [3, str. 5].

Lema 6.13 *Če n ni oblike p^i, potem* $|G| \leq n^{\sqrt{t}}$. Dokaz: [3, str. 5].

Lema 6.14 *Če algoritem odgovori "praštevilo,"je n praštevilo.*

Lema 6.15 *Za vsak naravni n velja* $\binom{2n+1}{n} > 2^{n+1}$. Dokaz: [19, str. 35].

Dokaz. $|I_r| = t$ potem za t velja: $t \geq \mathrm{or}_r(n) > \log^2 n$ iz tega sledi $t^2 > t \log^2 n$ zato $t > \lfloor \sqrt{t} \log n \rfloor$. Za $l = \lfloor \sqrt{\phi(r)} \log n \rfloor$ iz leme 6.12 sledi:

$$\begin{aligned}
|G| &\geq \binom{t+l}{t-1} \\
&\geq \binom{l+1+\lfloor \sqrt{t} \log n \rfloor}{\lfloor \sqrt{t} \log n \rfloor} && \text{(ker je } t > \sqrt{t} \log n) \\
&\geq \binom{2\sqrt{t} \log n + 1}{\sqrt{t} \log n} && (\phi(r) \geq t \text{ zato } \lfloor \sqrt{\phi(r)} \log n \rfloor \geq \lfloor \sqrt{t} \log n \rfloor) \\
&> 2^{\lfloor \sqrt{t} \log n \rfloor + 1} && \text{(zaradi leme 6.15)} \\
&\geq 2^{\lfloor \sqrt{t} \log n \rfloor} \\
&\geq n^{\sqrt{t}} && \text{(ker je } n = 2^{\log n})
\end{aligned}$$

Dobili smo, da je $|G| \geq n^{\sqrt{n}}$. Po lemi 6.13 velja, da n ni oblike p^i za nek $i > 0$, če velja neenakost v drugo smer. Torej bi n moral biti oblike p^i. Ampak, če bi to držalo, bi algoritem že v prvi vrstici vrnil sestavljeno število za tak n. Torej sledi, da je $i = 1$ in $n = p$. Torej je n praštevilo. □

Časovna zahtevnost algoritma

vrstica 1: $a^b = n \Rightarrow b = \log_a n$, naj bo m tako število, za katerega velja $n = m^m$, torej je $m = \mathrm{O}\big(\log n\big)$. Algoritem testira vse $a \in \mathbb{Z}$ iz intervala $[2, m]$ in preveri, ali drži, da je $b = \log_a n \in \mathbb{Z}$, nato pa mora preveriti še za b, ponovno iz intervala $[2, m]$, če velja $a = \sqrt[b]{n} \in \mathbb{Z}$. Časovna zahtevnost tega je $\mathrm{O}\big(\log^3 n\big)$ bitnih operacij.

vrstica 4: algoritem najde najmanjši r tako, da velja $\mathrm{or}_r(n) > \log^2 n$. Po Lemi 6.7 obstaja tak $r < \lfloor \log^5 n \rfloor$. Način, kako najti r je, da izračunamo $n^k \pmod r$ za $k = 1, 2, \ldots, \log^2 n$. Tukaj imamo $\mathrm{O}\big(\log^2 n\big)$ množenj po modulu r za vsak r, torej skupaj $\mathrm{O}\big(\log^7 n\big)$ bitnih operacij.

vrstica 5: računamo $\gcd(a, n)$, za $a = 1, \ldots, r$, da ugotovimo, če je $\gcd(a, n) > 1$ za nek $a \leq r$, izračun vsakega gcd porabi $\mathrm{O}\big(\log^2 n\big)$, skupaj $\mathrm{O}\big(\log^7 n\big)$.

vrstica 12: tukaj se preverja, če drži kongruenca:

$$(x+a)^n \equiv (x^n + a) \pmod{x^r - 1, n},$$

za $a = 1, 2, 3, \ldots, \lfloor \sqrt{\phi(r) \log n} \rfloor$, če je $a \leq \lfloor \sqrt{\phi(r) \log n} \rfloor$.

$(x+a)^n$ v obsegu $\mathbb{Z}_n^*$ reducirano po modulu $x^r - 1$ je enostavno, samo zamenjamo x^s z x^{s-r}. Da izračunamo $(x+a)^n$, imamo $\mathrm{O}\big(\log n\big)$ množenj, polinomov stopnje manjše kot r, s koeficienti velikosti $\mathrm{O}\big(\log n\big)$. Izračun vsake kongruence vzame $\mathrm{O}\big(\log^7 n\big)$ bitnih operacij. Vrstica 12 tako vzame $\mathrm{O}\big(\sqrt{\phi(r)} \log n \log^7 n\big)$ časa, kar je enako $\mathrm{O}\big(\sqrt{\phi(r)} \log^8 n\big)$, kar je $\mathrm{O}\big(\log^{\frac{21}{2}} n\big)$ bitnih operacij. Ta vrstica je časovno najbolj zahtevna, zato je skupna časovna zahtevnost $\mathrm{O}\big(\log^{10.5} n\big)$.

Časovno zahtevnost se da še izboljšati. Najboljši možni scenarij bi bil, da je $r = \mathrm{O}\big(\log^2 n\big)$, v tem primeru bi imel algoritem časovno zahtevnost $\mathrm{O}\big(\log^6 n\big)$.

Poglavje 7

Zaključek

Algoritmi za testiranje praštevilskosti so še danes zelo aktualno področje v računalništvu. Raziskovalci težijo k temu, da bi našli čim bolj enostaven deterministični algoritem, pri katerem bi bil enostaven tako dokaz pravilnosti kot tudi samo delovanje algoritma.

Danes algoritem AKS zadošča štirim zahtevam, ki naj bi jim zadoščali deterministični polinomski algoritmi za testiranje praštevil:

- AKS algoritem odloči za poljubno število n ali gre za sestavljeno število ali za praštevilo. Mnogi hitri verjetnostni testi so znani po tem, da delujejo le za števila s točno določenimi lastnostmi. Npr. Lucas-Lehmerjev test deluje pravilno samo za Mersennova števila.
- najslabša časovna zahtevnost algoritma je polinomska glede na število bitov števila, ki ga testiramo. Npr. deterministični algoritem APR, ki so ga odkrili Leonard Adleman, Carl Pomerance in Robert Rumely, ima časovno zahtevnost $\mathrm{O}\big((\ln n)^{c \ln \ln \ln n}\big)$, kar pa ni polinomskega reda velikosti.

- algoritem je deterministič̆en, saj pri vsakem vhodu pravilno odloči za kako število gre. Verjetnostni algoritmi, kot npr. Miller-Rabinov, imajo polinomsko časovno zahtevnost, ampak ne zagotavljajo, da je odgovor pravilen.

- pravilnost algoritma je brezpogojna, saj temelji na dejstvih, ki so jih uspeli dokazati. Npr. Millerjev algoritem temelji na razširjeni Riemanovi hipotezi, ki pa še ni dokazana niti ovržena.

Ne glede na vse to pa se v praksi še vedno najbolj uporabljajo verjetnostni algoritmi, kot sta Sollovay-Strasenov in Miller-Rabinov algoritem. Miller-Rabbinov algoritem še bolj pogosto kot Sollovay-Strassenov. Čeprav verjetnostni algoritmi niso 100% zanesljivi, rajši večkrat poženemo tak algoritem na naključnih podatkih in potem primerjamo odgovore. S tem postopkom poljubno zmanjšujemo verjetnost napake. In čeprav niso zanesljivi, so še vedno hitrejši kot deterministični algoritmi.

Literatura

[1] M. Agrawal, Primality Tests Based on Fermat's Little Theorem, ICDCN, 2006. http://www.cse.iitk.ac.in/users/manindra/survey/FLTBasedTests.pdf

[2] M. Agrawal, N. Kayal in N. Saxena. "Primes is in P", Preprint, avgust 2002.
http://www.cse.iitk.ac.in/users/manindra/algebra/primality_original.pdf

[3] M. Agrawal, N. Kayal in N. Saxena. "Primes is in P."Ann. Math. 160, 781-793, 2004.
http://homepages.math.uic.edu/ saunders/MCS425_2013/Agrawaletal-PrimesinP.pdf

[4] D. J. Bernstein. "An Exposition of the Agrawal-Kayal-Saxena Primality-Proving Theorem."Preprint. 2002. http://cr.yp.to/papers/aks.ps

[5] F. Borneman. PRIMES is in P: A breakthrough for 'Everyman'. Notices of the American Mathematical Society, 2003 545–552.

[6] R. Crandall, C. Pomerance. Section 4.2.1: The Lucas–Lehmerjev test, Prime Numbers: A Computational Perspective (1st ed.), Berlin: Springer, 2001,p. 167–170.
http://en.wikipedia.org/wiki/Lucas–Lehmer_primality_test

[7] Euler's analytic proof of the infinitude of the primes, junij 2012. http://automorph.wordpress.com/2012/06/20/eulers-analytic-proof-of-the-infinitude-of-the-primes/

[8] M. J. Fischer. Number Theory Summary, Yale University Department of Computer Science, februar 2010.
http://zoo.cs.yale.edu/classes/cs467/2010s/handouts/ho05.pdf

[9] Y. Ge. Elementary Properties of Cyclotomic Polynomials, Mathematical Reflections 2, 2008.
http://www.yimin-ge.com/doc/cyclotomic_polynomials.pdf

[10] A. Granville. It is easy to determine whether a given integer is prime, Bull. Amer. Math. Soc. (N.S.) 42 (2005), no. 1, 3–38. MR 2115065 (2005k:11011).
http://dx.doi.org/10.1090/S0273-0979-04-01037-7

[11] N. Guid. Izbrani Algoritmi, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 2010

[12] Heath. The Thirteen Books of Euclid's Elements, 2nd edition, 3 vol., Cambridge University Press, 1926 Repr. Dover, New York, 1956.

[13] S. Horsley. The Sieve of Eratosthenes. Being an Account of His Method of Finding All the Prime Numbers, Philosophical Transactions (1683–1775), Vol. 62. 1772, pp. 327–347

[14] F. Jaboeuf. Zgodovina matematike, Zgodbe o problemih-1. del, DMFA, 2001.

[15] T. Jin. Researching and Implementing on AKS Algorithm, University of Bath, maj 2005.
http://www.cs.bath.ac.uk/ mdv/courses/CM30082/projects.bho/2004-5/TongJin-2004-5.pdf

[16] D. Knuth. The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley, 1969.

[17] M. Krížek, F. Luca, L. Somer. 17 Lectures on Fermat Numbers: From Number Theory to Geometry. CMS Books in Ma-

thematics 9. Canadian Mathematical Society/Springer. p. 41. 2001. http://en.wikipedia.org/wiki/Lucas_primality_test

[18] H. Li. The Analysis and Implementation of the AKS Algorithm and Its Improvement Algorithms, Batchelor of Science in Computer Science with Honours, University of Bath, man 2007.

[19] B. Linowitz. An Exposition of the AKS Polynomial Time Primality Testing Algorithm, University of Pennsylvania, 2006. http://www-personal.umich.edu/ linowitz/papers/mastersthesis.pdf

[20] Z. S. Mcgregor-Dorsey. Methods of Primality Testing, MIT Undergraduate Journal of Mathematics, man 2005. http://www-math.mit.edu/phase2/UJM/vol1/DORSEY-F.PDF

[21] J. McKee, Špeeding Fermat's factoring method, Mathematics of Computation, 68:1729-1737 1999.

[22] I. Niven, H. S. Zuckerman, H. L. Montgomery. An Introduction to the Theory of Numbers, fifth edition, QA241.N56, 1991

[23] T. Novak. Odločitveni problem praštevilo, Obzornik za matematiko in fiziko, 2003, številka 5.

[24] G. Pavlič. Porazdelitev praštevil v množici N, DMFA, Ljubljana, 1997

[25] R. Petek. RSA kriptosistem, diplomsko delo, Univerza v Ljubljani, Fakulteta za matematiko in fiziko, december 2000.http://valjhun.fmf.uni-lj.si/ ajurisic/diplome/petek_dip.pdf

[26] M. Petkovšek. Rešitev problema praštevil, DMFA, Ljubljana, 2002/2003. http://www.presek.si/30/1519-Petkovsek.pdf

[27] R. G. E. Pinch, The Carmichael numbers up to 10^{21}, Math. Comp. 61, maj 2007. http://s369624816.websitehome.co.uk/rgep/p82.pdf

[28] M. Pipan. Elektronski plačilni sistemi na internetu, Diplomsko delo, Univerza v Ljubljani, Ekonomska fakulteta, junij 2002
http://www.cek.ef.uni-lj.si/u_diplome/pipan325.pdf

[29] B. Riemann, Ueber die Anzahl der Primzahlen unter einer gegebenen Grösse,
http://www.maths.tcd.ie/pub/HistMath/People/Riemann/Zeta/

[30] C. Rotella. An Efficient Implementation of the AKS Polynomial-Time Primality Proving Algorithm, Carnegie Mellon University, man 2005. http://www.cs.cmu.edu/afs/cs/user/mjs/ftp/thesis-program/2005/rotella.pdf

[31] R. Schoof. Four primality testing algorithms, In Algorithmic number theory : lattices, number fields, curves and cryptography, Math. Sci. Res. Inst. Publ., pages 101–126. Cambridge Univ. Press, Cambridge, 2008. http://www.mat.uniroma2.it/ schoof/05rene.pdf

[32] D. Stinson. Cryptography: Theory and Practice, Third Edition, Taylor & Francis Group, LLC, 2006.

[33] I. Vidav, Algebra, DMFA-založništvo, 4. natis, 1989.

[34] M. Vuk. Polinomski algoritmi za testiranje praštevilskosti, Magistrska naloga, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana, 2006.

Stvarno kazalo