NAJNUJNEJŠE O TESTIH PRAŠTEVILSKOSTI

ALEKSANDER SIMONIČ

1. Uvod

Vsi poznamo pojem *praštevila*: to je tako naravno število $n \geq 2$, ki ima natanko dva pozitivna delitelja, 1 in n. Vsi tudi znamo našteti nekaj prvih praštevil, poleg tega pa sta iz osnovne šole zagotovo marsikomu v spominu ostali dve lastnosti: praštevil je neskončno mnogo in vsako naravno število $n \geq 2$ se da zapisati kot produkt potenc praštevil. Toda, kako *vemo*, ali je neko *veliko* število res praštevilo? Morda se komu zdi raziskovanje v tej smeri preveč "neuporabno" ali celo trivialno, toda temu ni tako. Javni ključi *RSA kriptografskega sistema* [12, Poglavje 6], ki je priljubljen za šifriranje kratkih sporočil in digitalno podpisovanje, so dobljeni kot produkt dveh 1024-bitnih praštevil. Kako bi v *doglednem času* našli tako velika praštevila? Če že ne moremo povedati z gotovostjo, ali lahko vsaj z neko verjetnostjo trdimo, da smo v naključnem iskanju po neki podmnožici naravnih števil res našli praštevilo? Na predavanju smo se podrobneje seznanili z algoritmom, ki reši slednje vprašanje.

2. Naivni pristop

Kako smo v šoli določili, ali je dano naravno število praštevilo? Postopek je bil morda tak: preverimo, ali katero od števil $d \in \{2, \ldots, \lfloor\sqrt{n}\rfloor\}$ deli $n \geq 4$. Če tak delitelj obstaja, potem je n sestavljeno število, drugače pa je n praštevilo. V najslabšem primeru, ko je n praštevilo, bomo s tem postopkom opravili $\lfloor\sqrt{n}\rfloor - 1$ deljenj. Če bi za vsako deljenje porabili le eno mikrosekundo, bi za celoten postopek na 1024-bitnem številu potrebovali vsaj $3 \cdot 10^{140}$ let! Tudi če bi vrednosti za d omejili le na praštevila, ki ne presegajo $\sqrt{n}$, bi še vedno imeli opravka s $\pi(\sqrt{n})$ deljenj, kar je po *praštevilskem izreku* [6, Poglavje 6.2] $\pi(x) \sim x/\log x$ asimptotično enako $2\sqrt{n}/\log n$. Torej, tudi to nam ne prinese izrazitega napredka, pa še seznam praštevil bi potrebovali (kar še s trenutno najboljšimi algoritmi ni hitro). Potrebujemo močnejše orodje. Morda je malce presenetljivo, da se skriva v znamenitem izreku iz elementarne teorije števil 17. stoletja.

3. Učinkoviti algoritmi

"Čas je denar" je vsem dobro znan pregovor, ki še kako drži v poslovnem svetu. Z algoritmi, še posebno tistimi, ki nam krojijo vsakdan, ni nič drugače. Algoritem iz prejšnjega sestavka tako v realnem svetu nima mesta. Pravimo, da je algoritem *učinkovit*, če je njegova časovna zahtevnost polinomska funkcija na parametru *velikosti* vhodnega podatka. Torej, če je vhod algoritma naravno število $n \geq 2$, mora potem algoritem delovati v največ $p(\log n)$ časovnih enotah, kjer je $p(x)$ neki polinom s konstantnimi koeficienti. Naivni algoritmi za štiri osnovne računske operacije so vsi učinkoviti, npr. pisno množenje dveh naravnih števil $n \geq 2$ in $m \geq 2$ zahteva $\ll (\log n)(\log m)$ bit-operacij. Nekatere pomembne operacije v teoriji števil imajo učinkovite algoritme, npr. največji skupni delitelj (a, b) in modularno potenciranje $a^c \mod n$ za $1 \leq |a| < n$, $c \geq 2$ in $n \geq 2$, glej [4, Poglavji 4 in 5].

V skladu z uvodnimi vprašanji želimo najti *učinkovite* in *deterministične* algoritme, ki bi nam rešili naslednja odločitvena problema:

(1) PRAŠTEVILA: Za podano naravno število $n \geq 2$, ali je n praštevilo?
(2) SESTAVLJENA ŠTEVILA: Za podano naravno število $n \geq 2$, ali je n sestavljeno število?

Odločitvena problema PRAŠTEVILA in SESTAVLJENA ŠTEVILA sta si *komplementarna*: odgovor na en problem (da/ne) avtomatično poda odgovor na drug problem (ne/da). Kljub temu sta si z vidika teorije računske zahtevnosti, glej npr. [3] in [4, Poglavje 3], precej različna. Osrednja razreda problemov v tej teoriji sta P in NP, kjer razred P sestoji iz odločitvenih problemov, ki jih lahko rešimo z učinkovitimi in determinističnimi algoritmi, medtem ko razred NP sestoji iz tistih odločitvenih problemov, katerih rešitve lahko *preverimo* z učinkovitimi in determinističnimi algoritmi. Ker lahko vedno učinkovito preverimo

pravilnost razcepa danega naravnega števila, ki nam je bil posredovan kot dokaz za njegovo sestavljenost, imamo SESTAVLJENA ŠTEVILA $\in$ NP. Dokaz, da imamo tudi PRAŠTEVILA $\in$ NP, ni tako enostaven [4, Izrek 9.1.4]. Vsaj od leta 1976 je večina matematikov domnevala, da mora veljati PRAŠTEVILA $\in$ P, torej da obstaja "dovolj hiter" deterministični algoritem, ki nam odloči, ali imamo opravka s praštevilom ali ne. Indijski matematiki M. Agrawal, N. Kayal in N. Saxena so leta 2004 v [1] objavili tak algoritem, s časovno zahtevnostjo $\ll_\varepsilon (\log n)^{6+\varepsilon}$. Njihov rezultat (glej knjigo [8] za zelo dostopno študijo) predstavlja zgodovinski napredek na področju (algoritmične) teorije števil, vendar se redko uporablja v praksi. Za potrebe RSA kriptografije so bolj prikladni stohastični algoritmi.

4. Stohastični algoritmi

Stohastični (randomiziran) algoritem je tak algoritem, ki sloni na naključnih spremenljivkah in tako lahko poda različne odgovore pri enakem vhodu. To je očitno v nasprotju z determinističnimi algoritmi. Pomemben podrazred stohastičnih algoritmov se imenuje *Monte Carlo algoritmi*: izhod je lahko "da" ali "verjetno ne", in velja naslednje:

(1) če algoritem vrne "da", potem je vhod pozitiven primerek;
(2) obstaja realno število $p \in (0, 1]$, da za vsak pozitiven primerek algoritem poda (pravilen) "da" odgovor z verjetnostjo vsaj p.

Poglejmo si naslednji algoritem `fermat_test`, napisan v programskem jeziku *Python*.

```
1  from sympy.core.random import _randint
2  from sympy import gcd
3  def fermat_test(n):
4      # n>2 je naravno stevilo
5      if n%2 == 0:
6          return f"{n} je sestavljeno stevilo"
7      else:
8          a = _randint()(1,n-1)
9          if gcd(a,n) != 1:
10             return f"{n} je sestavljeno stevilo"
11         else:
12             if pow(a,n-1,n) == 1:
13                 return f"{n} je verjetno prastevilo"
14             else:
15                 return f"{n} je sestavljeno stevilo"
```

Algoritem 1. Fermatov test praštevilskosti.

Predno ga analiziramo, navedimo naslednje:

(1) Uporabljamo knjižnico *SymPy* za simbolno matematiko, iz katere smo si naložili metodi `_randint` in `gcd`.
(2) V vrstici 5 izraz `n%2 == 0` pomeni, da je ostanek pri deljenju n z 2 enak 0, tj. $2 \mid n$.
(3) V vrstici 8 je parameter `a` z metodo `_randint()(1,n-1)` izbran *(psevdo)naključno* v množici $\{1, 2, \ldots, n-1\}$.
(4) V vrstici 9 izraz `gcd(a,n) != 1` pomeni $(a, n) \neq 1$.
(5) V vrstici 12 izraz `pow(a,n-1,n) == 1` pomeni $a^{n-1} \mod n = 1$. Na tem mestu algoritem preko modularnega potenciranja preveri, ali velja $a^{n-1} \equiv 1 \pmod{n}$.

Če je $n \geq 3$ praštevilo, potem algoritem `fermat_test(n)` vrne `'n je verjetno prastevilo'`, kar je posledica *malega Fermatovega izreka:* $a^{p-1} \equiv 1 \pmod{p}$ *za praštevilo* p *in naravno število* a*, pri čemer moramo imeti* $p \nmid a$, glej [9, Izrek 6.3]. Od tod tudi ime algoritmu. Če pa algoritem vrne `'n je sestavljeno stevilo'`, potem imamo $(a, n) \neq 1$ ali $a^{n-1} \not\equiv 1 \pmod{n}$ za neki $a \in \{1, \ldots, n-1\}$. V tem primeru je n res sestavljeno število. Torej, `fermat_test` je stohastični algoritem, kjer "da" pomeni `'n je sestavljeno stevilo'` in pozitiven primerek je sestavljeno število. S tem je algoritem kandidat za Monte Carlo algoritem za odločitveni problem SESTAVLJENA ŠTEVILA. Na žalost pa ne izpolnjuje drugega pogoja. Z uporabo *teorije grup* se da pokazati naslednje [8, Lema 3.3.3].

Trditev 1. *Če za dano sestavljeno število* n *obstaja* $a \in \mathbb{Z}_n^* := \{k \in \{1, \ldots, n-1\} \colon (k, n) = 1\}$ *tako da velja* $a^{n-1} \not\equiv 1 \pmod{n}$*, potem je verjetnost da najdemo tak* a *v množici* $\mathbb{Z}_n^*$ *vsaj* $1/2$.

Kot primer navedimo, da so za $n = 15$ ustrezne vrednosti $a \in \{2, 7, 8, 13\} \subset \mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. Torej, če a iz trditve 1 vedno obstaja, bo `fermat_test` učinkovit Monte Carlo algoritem (s $p = 1/2$) za odločitveni problem SESTAVLJENA ŠTEVILA. Toda tak a ne obstaja vedno. Sestavljenim številom n, za katera velja $a^{n-1} \equiv 1 \pmod{n}$ *za vsak* $a \in \mathbb{Z}_n^*$, pravimo *Carmichaelova števila*. Najmanjše tako število je $561 = 3 \cdot 11 \cdot 17$, Alford, Granville in Pomerance pa so leta 1994 v [2] celo dokazali, da jih je neskončno.

5. Miller–Rabinov test

Fermatov test ni Monte Carlo algoritem, saj obstaja (neskončna) množica pozitivnih primerkov (Carmichaelova števila), kjer algoritem *vedno* poda napačen odgovor. S tem bi si mislili, da je algoritem obsojen na propad. Toda, ali se bi ga dalo nekako "popraviti" v smislu, da se izognemo Carmichaelovim številom? Gary L. Miller je leta 1976 v [5] to storil z uporabo ustrezne različice malega Fermatovega izreka.

Izrek 1 (Miller, 1976)**.** *Naj bo p liho praštevilo. Pišimo $p - 1 = d \cdot 2^l$, kjer je $d \in \mathbb{N}$, $l \in \mathbb{N}$ in $2 \nmid d$. Če je a tako naravno število, da velja $p \nmid a$, potem:*

(1) $a^d \equiv 1 \pmod{p}$, *ali*
(2) $a^{d \cdot 2^i} \equiv -1 \pmod{p}$ *za neki* $i \in \{0, 1, \ldots, l-1\}$.

Miller je na podlagi izreka 1 konstruiral učinkovit deterministični algoritem za problem PRAŠTEVILA, vendar pod predpostavko o pravilnosti še vedno nerešene *posplošene Riemannove domneve* (za poseben neskončen podrazred Dirichletovih L-funkcij). Originalno Riemannovo domnevo lahko ekvivalentno formuliramo [6, Izrek 13.1] z enačbo

$$\pi(x) = \int_2^x \frac{\mathrm{d}u}{\log u} + O\left(\sqrt{x} \log x\right).$$

Najti dokaz te na videz preproste ocene spada med najpomembnejše odprte probleme v matematiki.

Michael O. Rabin je leta 1980 v [7] pokazal, da se da Millerjev algoritem brezpogojno preoblikovati v učinkovit Monte Carlo algoritem s $p = 3/4$. Različica takega algoritma, znana kot *Miller–Rabinov test praštevilskosti*, je zapisana spodaj.

```
from sympy.core.random import _randint
from sympy import gcd
def miller_rabin_test(n):
    # n>2 je naravno stevilo
    if n%2 == 0:
        return f"{n} je sestavljeno stevilo"
    else:
        d = n-1
        l = 0
        while d%2 == 0:
            l = l+1
            d = d//2
        a = _randint()(1,n-1)
        if gcd(a,n) != 1:
            return f"{n} je sestavljeno stevilo"
        else:
            b = pow(a,d,n)
            if b == 1:
                return f"{n} je verjetno prastevilo"
            else:
                for i in range(l):
                    if (b+1)%n == 0:
                        return f"{n} je verjetno prastevilo"
                    else:
                        b = pow(b,2,n)
                return f"{n} je sestavljeno stevilo"
```

Algoritem 2. Miller–Rabinov test praštevilskosti.

Algoritem 2 lahko analiziramo podobno kot algoritem 1, pri čemer namesto malega Fermatovega izreka uporabimo izrek 1. Najtežja naloga je seveda dokaz, da imamo $p = 3/4$. Pri tem moramo trditev 1 nadomestiti s podobnim, toda težje dokazljivim rezultatom.

Izrek 2 (Rabin, 1980)**.** *Naj bo n liho sestavljeno naravno število. Pišimo $n-1 = d \cdot 2^l$, kjer je $d \in \mathbb{N}$, $l \in \mathbb{N}$ in $2 \nmid d$. Potem imamo*

$$\left|\left\{a \in \{1,\dots,n-1\} \colon a^d \not\equiv 1 \pmod{n} \wedge \forall i \in \{0,\dots,l-1\}.\ a^{d\cdot 2^i} \not\equiv -1 \pmod{n}\right\}\right| \geq \frac{3}{4}(n-1).$$

Časovna zahtevnost Miller–Rabinovega testa je $\ll (\log n)^3$. Ta test pa ni prvi učinkovit Monte Carlo algoritem: leta 1977 sta takega (s $p = 1/2$) objavila R. M. Solovay in V. Strassen v [11]. Njun algoritem temelji na Eulerjevem kriteriju za kvadratne ostanke [9, Izrek 11.3] in zakonu kvadratne recipročnosti za Jacobijev simbol [9, Poglavje 11.3], ter ima enako časovno zahtevnost kot algoritem 2. Vseeno pa v praksi deluje počasneje od Miller–Rabinovega testa, zato se slednji uporablja bolj pogosto.

6. Uporaba Miller–Rabinovega testa

V zaključnem razdelku se bomo soočili z zadnjim vprašanjem iz uvoda. Recimo, da želimo poiskati n-bitno praštevilo, torej, iščemo praštevila na intervalu $\left(2^{n-1}, 2^n\right)$. Spomnimo, da trenutni standard RSA kriptografije zahteva vsaj 2048-bitne ključe, torej $n \geq 1024$. Kako bi si pri tem pomagali z algoritmom 2? Poglejmo si naslednji postopek.

```
1  def iskalnik_prastevil(n,k,j=1):
2      # n>2 je naravno stevilo
3      # k>0 je naravno stevilo
4      num = _randint()(2**(n-1)+1,2**n-1)
5      a = miller_rabin_test(num)
6      if a == f"{num} je verjetno prastevilo":
7          i = 1
8          while i <= k-1:
9              a = miller_rabin_test(num)
10             if a == f"{num} je verjetno prastevilo":
11                 i = i + 1
12             else:
13                 return f"{num} je sestavljeno stevilo"
14         return [j, f"{num} je verjetno prastevilo"]
15     else:
16         j = j + 1
17         return iskalnik_prastevil(n,k,j)
```

Algoritem 3. Iskalnik praštevil na intervalu $\left(2^{n-1}, 2^n\right)$.

Algoritem `iskalnik_prastevil(n,k,j=1)` ima tri argumente. Argument `n` predstavlja bitno dolžino iskanega praštevila. Argument `j` je na začetku postavljen na vrednost 1 in nam šteje število poskusov, pri katerih je bil algoritem 3 neuspešen pri iskanju morebitnih praštevil (preko Miller–Rabinovega testa – algoritem 2), pri čemer štejemo uspeh `'n je verjetno prastevilo'` za zadnji poskus. Če se to zgodi, nam potem `k` predstavlja maksimalno možno število dodatnih sklicev (vključno s tistim pri zadnjem poskusu) na algoritem 2, glej vrstice 6–14 algoritma 3. Razjasnimo to s primerom uporabe.

```
In [1]: iskalnik_prastevil(1024,10)
Out[1]: [839,
 '16435462081548883896178202651294322404698754113272092330964741831026818877068464587032584522111483776758026774564299215532019945003679758036456429908187830610484441024071654126583615621617057143652792368623948512555100207588923335221379263366310265280785916992644892771282013901543460745247246357894833271778 1 je verjetno prastevilo']
```

V tem primeru je algoritem 3 v 839. poskusu na intervalu $\left(2^{1023}, 2^{1024}\right)$ našel 309-mestnega kandidata za praštevilo. To število je potem opravilo Miller–Rabinov test v vseh devetih ponovitvah. Kako bi interpretirali to ugotovitev v jeziku verjetnosti? Za natančno obravnavo potrebujemo definiciji:

(1) Naj bo $\mathcal{P}$ dogodek, da je naključno izbrano število N (ki ga testiramo) iz množice $\left(2^{n-1}, 2^n\right) \cap \mathbb{N}$, $n \geq 3$, praštevilo.
(2) Naj bo $\mathcal{MR}_k$ dogodek, da N opravi Miller–Rabinov test k-krat. V tem primeru lahko predpostavimo, da je ta k enak tistemu iz algoritma 3.

Skladno s prejšnjim vemo naslednje:

(1) $0 < P(\mathcal{P}) < 1$: vsa števila v množici zagotovo niso praštevila, po Bertrandovem postulatu [6, str. 49] pa praštevilo tam vedno obstaja.
(2) $P(\mathcal{MR}_k \mid \mathcal{P}) = 1$ za vsak $k \in \mathbb{N}$.
(3) $P\left(\overline{\mathcal{MR}_1} \mid \overline{\mathcal{P}}\right) \geq 3/4$ po izreku 2. S tem je $P\left(\mathcal{MR}_1 \mid \overline{\mathcal{P}}\right) \leq 1/4$, torej $P\left(\mathcal{MR}_k \mid \overline{\mathcal{P}}\right) \leq 4^{-k}$.

Zanima nas $P(\mathcal{P}|\mathcal{MR}_k)$, tj. verjetnost, da je testirano število N praštevilo, ko je Miller–Rabinov test že vrnil `'N je verjetno prastevilo'` k-krat. Uporabimo *Bayesovo formulo*:

$$P(\mathcal{P}|\mathcal{MR}_k) = 1 - P\left(\overline{\mathcal{P}}|\mathcal{MR}_k\right) = 1 - \frac{1}{1 + \frac{P(\mathcal{MR}_k|\mathcal{P})P(\mathcal{P})}{P\left(\mathcal{MR}_k|\overline{\mathcal{P}}\right)P\left(\overline{\mathcal{P}}\right)}} \geq 1 - \frac{4^{-k}}{P(\mathcal{P})} = 1 - \frac{2^{n-1}-1}{\pi(2^n) - \pi\left(2^{n-1}\right)} 4^{-k}.$$

Praštevilski izrek zagotavlja $\pi(2x) - \pi(x) \sim x/\log x$, *eksplicitna* različica [10] pa

$$\pi(2x) - \pi(x) \geq \left(1 - \frac{2}{\log(2x)}\right) \frac{x}{\log x}$$

za vse $x \geq 4$. Seveda obstajajo boljše ocene, vendar je zapisana za naše potrebe dovolj ostra, pa še enostavno si jo je zapomniti. S tem imamo naslednji rezultat.

Trditev 2. *Za vse $n \geq 3$ in $k \geq 1$ velja*

$$P(\mathcal{P}|\mathcal{MR}_k) \geq 1 - \left(1 - \frac{2}{n \log 2}\right)^{-1} \left(1 - \frac{1}{2^{n-1}}\right) 4^{-k}(n-1)\log 2.$$

Trditev 2 nam teoretično upravičuje pričakovano hevristično načelo: v primeru, da nam algoritem 3 za *dovolj velik k* obljubi "n je verjetno praštevilo", je število n *skoraj zagotovo* res praštevilo. V našem primeru imamo $n = 1024$ in $k = 10$. Po trditvi 2 imamo $P(\mathcal{P}|\mathcal{MR}_k) \geq 0.99932$, torej lahko rečemo naslednje: *Verjetnost, da je po algoritmu 3 izbrano število* $1643546 \ldots 781$ *res praštevilo, je vsaj* 99.932%. Z večanjem števila k lahko še izboljšamo to verjetnost, seveda na račun hitrosti algoritma 3. Zgornja neenakost nam natačno pove, kako to storiti.

Pri algoritmu 3 nimamo "deterministočnega" zagotovila, da se bo res ustavil, torej da bomo naleteli na uspešen poskus. Vseeno lahko podamo zgornjo mejo za *pričakovano* število poskusov, tj. vrednost parametra j. Naj bo $n \geq 3$, $p = \mathcal{P}(\mathcal{MR}_1)$ in $X \sim$ število poskusov. Potem imamo

$$p = P(\mathcal{MR}_1 \mid \mathcal{P})\, P(\mathcal{P}) + P\left(\mathcal{MR}_1 \mid \overline{\mathcal{P}}\right) P\left(\overline{\mathcal{P}}\right) \geq P(\mathcal{P}) = \frac{\pi(2^n) - \pi\left(2^{n-1}\right)}{2^{n-1} - 1}$$

in s tem

$$\mathbb{E}[X] = p \sum_{m=1}^{\infty} m(1-p)^{m-1} = \frac{1}{p} \leq \left(1 - \frac{2}{n \log 2}\right)^{-1} \left(1 - \frac{1}{2^{n-1}}\right)(n-1)\log 2.$$

Torej, za $n = 1024$ imamo $\mathbb{E}[X] \leq 712$, kar ni tako daleč od števila poskusov pri našem primeru. Podobno lahko pokažemo tudi, da je pričakovana časovna zahtevnost algoritma 3 enaka $O\left(kn^3\right)$.

Literatura

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793.
[2] W. R. Alford, A. Granville, C. Pomerance, *There are infinitely many Carmichael numbers*, Ann. of Math. (2) **139** (1994), no. 3, 703–722.
[3] S. Arora, B. Barak, *Computational complexity: A modern approach*, Cambridge University Press, New York, 2009.
[4] E. Bach, J. Shallit, *Algorithmic number theory. Vol. 1: Efficient algorithms*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1996.
[5] G. L. Miller, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. **13** (1976), no. 3, 300–317.
[6] H. L. Montgomery, R. C. Vaughan, *Multiplicative number theory. I. Classical theory*, Cambridge Studies in Advanced Mathematics 97, Cambridge University Press, Cambridge, 2007.
[7] M. O. Rabin, *Probabilistic algorithm for testing primality*, J. Number Theory **12** (1980), no. 1, 128–138.
[8] L. Rempe-Gillen, R. Waldecker, *Primality testing for beginners*, Student Mathematical Library 70, AMS, Providence, RI, 2014.
[9] K. H. Rosen, *Elementary number theory and its applications*, 6th ed., Addison-Wesley, USA, 2011.

[10] J. B. Rosser, L. Schoenfeld, *Approximate formulas for some functions of prime numbers*, Illinois J. Math. **6** (1962), 64–94.
[11] R. Solovay, V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Comput. **6** (1977), no. 1, 84–85.
[12] D. R. Stinson, M. B. Paterson, *Cryptography: theory and practice*, 4th. ed., Textbooks in Mathematics, CRC Press, Boca Raton, FL, 2019.

Fakulteta za matematiko, naravoslovje in informacijske tehnologije, Univerza na Primorskem