SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Zvonimir Iveković

TESTOVI PROSTOSTI I METODE FAKTORIZACIJE BROJEVA

Diplomski rad

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

Zagreb, rujan, 2016.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Poglavlje 1

Uvod

Središnji objekt ovog rada su prosti brojevi. To su pozitivni cijeli brojevi veći od 1 koji imaju samo dva djeljitelja: 1 i samog sebe. Broj 1 nije niti prost niti složen.

Prosti brojevi su predmet promatranja od samih početaka matematike. Već je Euklid u trećem stoljeću pr. Kr. dokazao da prostih brojeva ima beskonačno mnogo, a Eratosten dao prvi algoritam za njihove pronalaženje (tzv. *Eratostenovo sito*). Neke od njihovih ideja još i danas se koriste u proučavanju prostih brojeva.

Nagli razvoj računarstva u 20. stoljeću dao je prostim brojevima, kao i cijeloj teoriji brojeva, novi značaj. S napretkom u komunikaciji, pojavila se potreba za zaštitom povjerljivih podataka, te brzim i preciznim provođenjem operacija nad velikim brojevima.

Danas se najrašireniji kriptosustavi zasnivaju upravo na svojstvima prostih brojeva. Koriste se metode za pronalaženje brojeva koje je teško ili nemoguće faktorizirati. Takvi se brojevi koriste kao ključevi. Kako biti siguran da smo takav broj našli? Prvo, naravno, broj mora biti velik, jer za male brojeve svatko može jednostavnim dijeljenjem (”grubom silom”) doći do odgovora, te npr. ukrasti naš privatni ključ. S druge strane, broj mora biti dovoljno malen da se nad njim daju izvesti operacije potrebne za kriptoalgoritam. Tu stupaju na pozornicu testovi prostosti. To su algoritmi koji za dani broj (prevelik za naivne provjere) pokušaju dati odgovor na pitanje je li taj broj prost. Neke od tih algoritama obrađujemo u prvom poglavlju ovog rada. Budući da takvih algoritama ima mnogo, a literatura je opsežna, odlučio sam se ograničiti na one algoritme koje smatram predstavnicima čitavih skupina. Jedan od algoritama je Fermatov test, čije varijante i optimizacije čine veliku klasu danas poznatih testova prostosti. Nadalje, donosimo Miller–Rabinov test koji predstavlja klasu vjerojatnosnih algoritama, koji žrtvuju nešto matematičke preciznosti u korist brzine. Takvi se algoritmi danas najčešće i koriste. Na kraju je obrađen algoritam AKS. Iako, uglavnom od teorijskog značaja, to je jedini poznati polinomni test prostosti, i kao takav zauzima posebno mjesto u povijesti matematike i računarske znanosti.

Druga vrsta algoritama, vrlo bliska testovima prostosti (čak se može smatrati i njiho-

vom podklasom), su algoritmi za rastav broja na proste faktore.

Osnovni teorem aritmetike. *Za svaki prirodni broj $n > 1$ postoje prosti brojevi $p_1, \ldots, p_k$ i prirodni brojevi $\alpha_1, \ldots, \alpha_k$ tako da je*

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}.$$

Nadalje, takva je faktorizacija broja n jedinstvena do na poredak faktora p_i.

Ideja korištenja velikih brojeva za zaštitu podataka zasniva se na jednostavnoj činjenici da nije poznat dovoljno brz algoritam koji bi u razumnom vremenu faktorizirao dovoljno veliki broj. Kao što ćemo vidjeti u ovom radu, čak i najbrži algoritmi nisu polinomne vremenske složenosti. Iako za neke brojeve specijalnih oblika postoje puno brže provjere prostosti i faktorizacije, za većinu brojeva to nije slučaj. U drugom poglavlju donosimo tri najkorištenija algoritma za rješavanje ovog problema. Čitatelj će opaziti da su oni puno složeniji od testova prostosti, što i nije čudno, budući da obavljaju i veći posao. Naime, lakše je reći je li broj složen nego naći neki njegov prosti faktor. Pojašnjene su ideje sita, te pokazani neki načini za dobivanje takvih sita. To su algoritmi QS i NFS. Zadnji algoritam obrađen u ovom radu je metoda eliptičkih krivulja (ECM). Vidjet ćemo po čemu se taj algoritam razlikuje od prethodna dva, te kada se koristi jedan, a kada drugi.

Ideja ovog rada je da čitatelj upozna osnovne principe modernih algoritama faktorizacije i testova prostosti, te da ga se uputi na daljne proučavanje. Zato se u svakoj sekciji nalazi cijeli algoritam, te samo najosnovniji teorijski pojmovi potrebni za njegovo razumijevanje. Ipak, svi koncepti su dobro poznati, te rad obiluje referencama na literaturu u kojoj su podrobno objašnjeni. Ta je literatura široko dostupna, pa smatram da je davanje prednosti konciznosti i jasnoći u ovom radu opravdano.

Poglavlje 2

Testovi prostosti

Na prvi pogled čini se da nema potrebe razdvajati ovo poglavlje od idućeg, koje se bavi metodama faktorizacije. Zaista, ukoliko uspješno provedemo faktorizaciju, dobili smo i odgovor na pitanje složenosti broja. Ipak, postoje mnogi algoritmi koji se bave upravo otkrivanjem složenosti broja, i u tu svrhu su pronađeni mnogi matematički alati. Razlog tomu je "složenost" dosad poznatih algoritama za faktorizaciju. Oni rade na relativno malim brojevima obzirom na one za koje je moguće ispitati prostost. Algoritmi za faktorizaciju se pri ispitivanju prostosti koriste, uglavnom, kao pomoćni alati: prvo ih se iskoristi za provjeravanje manjih brojeva, a za preostale, velike brojeve se koriste složeniji algoritmi, od kojih neke navodimo u ovom poglavlju.

Primjer 2.1. *Uzmimo kao primjer broj* $N = 260849323075371835669784094383812120359260783810157225730623388382401$. *Taj broj ima* 69 *znamenki, što u svijetu računarstva i nije mnogo, ali poslužit će kao ilustracija. Naime, kad bismo ga faktorizirali, otkrili bismo da ima* 34 *prosta faktora, od kojih je najveći* 829*, koji je mnogo manji od* $\sqrt{N}$*. Dakle, klasičnim algoritmom sita radili bismo mnogo uzaludnog posla provjeravajući sve proste faktore koji su manji od* $\sqrt{N}$*. Ipak, ako bismo se zadovoljili saznanjem da je broj složen, isti taj algoritam dao bi odgovor vrlo brzo, jer je najmanji prosti faktor broja* N *jednak* 17.

Naravno, u praksi su problemi faktorizacije i odlučivanja složenosti puno veći, i promatraju se brojevi sa desecima milijuna znamenaka, te je nepraktično koristiti jednostavne algoritme poput onog iz primjera.

Testovi koji provjeravaju prostost često imaju sljedeći format: ukoliko neki uvjet vrijedi za N, onda je N prost, inače je složen. Nažalost, obično su takvi testovi, ili kompliciranji, ili primjenjivi samo na brojeve specijalnog oblika (npr. Fermatovi ili Mersenneovi brojevi). Zbog toga se javlja potreba za brzim i jednostavnim algoritmima, koji pak, žrtvuju nešto svoje točnosti. Takvi algoritmi imaju pozitivnu vjerojatnost pogreške, koja se, obično,

može nauštrb jednostavnosti, ili resursa, proizvoljno smanjiti. Valja napomenuti da ti algoritmi griješe uvijek tako da složene brojeve proglašavaju prostim, a nikad obrnuto. U [4] autor radi razliku između navedene dvije vrste algoritama, te samo prvu vrstu algoritama naziva *testovima prostosti*, a drugu *testovima složenosti*. Tu ćemo podjelu i ovdje usvojiti.

2.1 Fermatov test

Sljedeći je teorem kamen temeljac nekih testova prostosti i mnogih testova složenosti.

Teorem 2.2 (**Mali Fermatov teorem**)**.** *Ako je p prost i a relativno prost s p, tada vrijedi* $a^{p-1} \equiv 1 \pmod{p}$.

On je, ustvari, posljedica općenitijeg **Eulerovog teorema**, kojeg ćemo ovdje dokazati, za što je potrebno navesti neke činjenice o strukturi skupa $\mathbb{Z}_n$ (skup klasa ostataka modulo n).

Na tom skupu definiramo operacije zbrajanja i množenja na sljedeći način:

$$[a]_n + [b]_n = [a + b]_n$$
$$[a]_n \times [b]_n = [ab]_n .$$

Pokazuje se da s tim operacijama $\mathbb{Z}_n$ čini komutativni prsten s jedinicom.

Definicija 2.3. *Definiramo $\mathbb{Z}_n^*$ kao skup onih klasa ostataka modulo n koji imaju multiplikativni inverz, tj.*

$$\mathbb{Z}_n^* := \left\{ [a]_n \in \mathbb{Z}_n : (\exists\, [b]_n \in \mathbb{Z}_n)\, [a]_n \times [b]_n = [1]_n \right\}.$$

Sada možemo definirati **Eulerovu φ funkciju** kao broj elemenata skupa $\mathbb{Z}_n^*$. Ukoliko uzmemo u obzir činjenicu da klasa $[a]_n$ ima multiplikativni inverz ako i samo ako je a relativno prost s n, uočavamo da je za prirodni broj n vrijednost fuknkcije $\varphi(n)$ upravo broj onih brojeva manjih od n koji su njim relativno prosti. Sada imamo sve što nam treba za iskaz sljedećeg teorema.

Teorem 2.4 (**Eulerov teorem**)**.** *Za svaki pozitivni cijeli broj n i svaki cijeli broj a koji je relativno prost s n vrijedi*

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Dokaz. Neka je $\alpha := [a]_n \in \mathbb{Z}_n^*$. Promotrimo funkciju $f : \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_n^*$, $f(\beta) = \alpha\beta$. Uočimo da je f injekcija. Zaista, iz $\alpha\beta = \alpha\beta'$, množenjem multiplikatvnim inverzom od α, dobivamo $\beta = \beta'$. Budući da je f preslikavanje s konačnog skupa na sama sebe, injektivnost povlači

surjektivnost, pa je f, ustvari, bijekcija. To znači da je svejedno množimo li elemente u domeni ili slici od f – to je naprosto isti skup. Dakle, imamo

$$\prod_{\beta\in\mathbb{Z}_n^*}\beta = \prod_{\beta\in\mathbb{Z}_n^*}(\alpha\beta) = \alpha^{\varphi(n)}\prod_{\beta\in\mathbb{Z}_n^*}\beta.$$

Sada, zbog zatvorenosti skupa $\mathbb{Z}_n^*$ na množenje (što je lako provjeriti), možemo pokratiti zajednički faktor $\prod_{\beta\in\mathbb{Z}_n^*}\beta$, te dobivamo

$$\alpha^{\varphi(n)} = [1]_n. \qquad \square$$

Sada teorem 2.2 slijedi iz Eulerovog teorema i činjenice da je $\varphi(n) = n - 1$ ako je n prost.

Napomena 2.5. *Sve činjenice koje su upotrijebljene, a nisu opravdane, i općenito više o Eulerevom teoremu i strukturi $\mathbb{Z}_n$ može se naći u [5].*

Odmah je jasno da se Teorem 2.2 (točnije, njegov obrat) može koristiti kao test složenosti. Za broj n tražimo s njim relativno prost broj a tako da $a^{n-1} \not\equiv 1 \pmod{n}$. Ukoliko nađemo takav a, dokazali smo da je n složen. No, može li se taj teorem iskoristiti kao test prostosti? Odgovor je, nažalost, ne. Naime, postoje parovi brojeva a i n takvi da zadovoljavaju kriterij iz teorema, ali je n složen.

Definicija 2.6. *Za neparan **složen** broj n kažemo da je (Fermat) **pseudoprost za bazu a** ako vrijedi*

$$a^{n-1} \equiv 1 \pmod{n}.$$

Napomena 2.7. *Za mnoge testove složenosti, formulirane slično Frematovom testu, postoje pseudoprosti brojevi, npr. Lucas pseudoprosti ili Euler pseudoprosti brojevi. U skladu s time, u gornjoj definiciji bi trebalo stajati Fermat pseudoprosti, no u literaturi se obično to ispušta.*

Primjer 2.8. *Broj* 341 *je pseudoprost za bazu* 2*. Zaista,* $341 = 11 \cdot 31$*, pa je* 341 *složen, ali vrijedi* $2^{340} = \left(2^{10}\right)^{34} \equiv 1^{34} \equiv 1 \pmod{341}$.

Dakle, ako broj prođe Fermatov test za neku bazu, ne možemo zaključiti da je prost. Štoviše, pokazano je da pseudoprostih brojeva za danu bazu ima beskonačno mnogo. No, tu nije kraj iskoristivosti malog Fermatovog teorema. Naime, provodeći Fermatov test za različite baze, možemo smanjiti vjerojatnost da broj bude lažno prost. Npr., broj iz prošlog primjera je pseudoprost za bazu 2, no nije za bazu 3: $3^{340} \equiv 56 \pmod{341}$, pa saznajemo da je složen. Ukoliko stavimo gornju ogradu na brojeve koje promatramo, možemo na taj način ispitivati njihovu složenost.

Primjer 2.9. *Pokazalo se da među brojevima manjim od* $25 \cdot 10^9$ *ima* 21853 *pseudoprostih za bazu* 2*, a od njih je* 4709 *također pseudoprosto za bazu* 3*. Nadalje, baza* 5 *ostavlja* 2552*, dok testiranje za bazu* 7 *preživi samo* 1770 *brojeva. Dakle, ukoliko bismo imali listu tih* 1770*, za provjeravanje složenosti broja* $n < 25 \cdot 10^9$ *bilo bi dovoljno sprovesti* 4 *Fermatova testa.*

Preostaje riješiti pitanje složenosti Fermatovog testa. Kao što vidimo, problem je izračunati $a^d \pmod n$ što je efikasnije moguće. U tu svrhu koristimo algoritam (preuzet iz [4]) koji se temelji na binarnoj reprezentaciji eksponenta d:

$$d = \beta_0 + \beta_1 \cdot 2^1 + \cdots + \beta_k \cdot 2^k.$$

Tu je $k + 1 = \lfloor \log_2 d \rfloor + 1$ broj znamenaka od d u binarnom zapisu. Sada imamo

$$a^d = a^{\sum_{i=0}^{k} \beta_i 2^i} = \prod_i a^{\beta_i 2^i} = \prod_{\beta_j = 1} a^{2^j}.$$

Primjer 2.10. *Želimo izračunati* a^{13}*. Prvo zapišimo* 13 *u bazi* 2*, tj.* $13 = 1 + 4 + 8$*. Sada izračunajmo* a^2*,* a^4*,* a^8 *tako da ponavljamo kvadriranje počevši od a. Na kraju, pomnožimo* $a^{13} = a^1 \cdot a^4 \cdot a^8$*.*

Ovaj je postupak vrlo efikasan, pogotovo u računalu, u kojem je d već zapisan u binarnom obliku. Donosimo metodu napisanu u programskom jeziku *C#*, koja će pomoći u analizi složenosti gornjeg postupka:

```
1         /* vraca a^d mod n*/
2          public int computeAtoDModN(int a, int d, int n) {
3
4              int prod = 1, a2j = a;
5
6              while (d > 0) {
7                  /* množimo samo one potencije od a koje odgovaraju
  ↪  jedinicima u binarnom zapisu broja a */
8                  if (d % 2 == 1) prod = (prod * a2j) % n;
9                  d /= 2;  a2j = (a2j * a2j) % n;
10             }
11             return prod;
12         }
```

Iz ovog ilustrativnog koda možemo iščitati sljedeće: broj množenja i dijeljenja modulo n se nalazi između $\lfloor \log_2 d \rfloor$ i $2\lfloor \log_2 d \rfloor$, ovisno o tome koliko jedinica ima u binarnom

zapisu broja d. Provođenje Fermatovog testa (kad je $d = n - 1$) zahtijeva, u najgorem slučaju, $2\log_2 d$ množenja i dijeljenja modulo n, što znači da je to polinomni algoritam.

Nažalost, pokazuje se da, iako je za većinu brojeva dovoljno svega nekoliko puta sprovesti Fermatov test da bi se pokazala njihova složenost, postoje i takvi brojevi za koje test ne daje odlučiv odgovor.

Definicija 2.11. *Složene brojeve N koji zadovoljavaju* $a^{N-1} \equiv 1 \pmod N$ *za svaki broj a relativno prost s N nazivamo* ***Carmichaelovim brojevima****.*

Primjer 2.12. *Najmanji Carmichaelov broj je* $561 = 3 \cdot 11 \cdot 17$*. Trenuntno najveći otkriveni Carmichaelov broj s tri prosta faktora ima čak* 60351 *znamenki.*

Carmichaelovi brojevi su relativno rijetki u odnosu na ostale brojeve, no ima ih dovoljno da ih se ne može zanemariti pri testiranju Fermatovim testom (detaljnije o broju Carmichaelovih brojeva nalazi se u [2]). Zato se koriste i drugi kriteriji, koji ne dopuštaju takve anomalije, npr. Eulerov kriterij ili jaki test prostosti.

Teorem 2.13 (Eulerov test)**.** *Ako je N neparan prost broj i a relativno prost s N, onda vrijedi*

$$a^{(N-1)/2} \equiv \pm 1 \pmod N.$$

Dokaz. Dokaz ovog teorema nije težak, ali zahtijevao bi malo teorije kvadratnih ostataka, pa ga zato ovdje ispuštamo. Kompletan dokaz može se naći u [4]. □

Teorem 2.14 (Jaki test prostosti)**.** *Ako je N neparan prost broj i* $N - 1 = 2^s t$*, t neparan, te ako a nije djeljiv s N, onda vrijedi (samo) jedno od sljedećeg:*

$$\begin{cases} a^t \equiv 1 \pmod N, \\ a^{2^i t} \equiv -1 \pmod N \quad \text{za neki } i, 0 \le i \le s-1. \end{cases}$$

Dokaz. Iz malog Fermatovog teorema znamo da $a^{N-1} = a^{2^s t} \equiv 1 \pmod N$. Budući da je N neparan prost broj, jedina rješenja jedandžbe $x^2 \equiv 1 \pmod N$ su brojevi ± 1. To znači da $a^{\frac{2^s t}{2}} = a^{2^{s-1} t} \equiv \pm 1 \pmod N$. Ukoliko $a^{2^{s-1} t} \equiv -1 \pmod N$, dokaz je gotov ($i = s - 1$). Ukoliko, pak, $a^{2^{s-1} t} \equiv 1 \pmod N$, isto razmišljanje primijenimo na $a^{2^{s-2} t}$, i tako dalje, sve do a^t. □

Definicija 2.15. ***Složene*** *brojeve N koji zadovoljavaju uvjete Teorema 2.14 zovemo* ***jakim pseudoprostim brojevima*** *(za bazu a).*

Napomena 2.16. *Pokazano je da su jaki pseudoprosti brojevi, također, Euler pseudoprosti.*

Sada dajemo kôd metode jakog testa prostosti napisan u jeziku C#. Uočimo da se koristi prije napisana metoda za računanje $a^d \equiv \pmod n$. Kôd je prilagođen iz pseudokoda danog u [2].

```
/*ova metoda odlučuje je li n=1+2^s*t složen ili jaki vjerojatno
↪ prost broj za bazu a, 1<a<n-1. Vraća true ako je n složen, te
↪ false ako je test neodlučiv.*/
public bool testForCompositeness(int a,int n) {
            //prvo trebamo prikazati broj u obliku n=1+2^st
            int t = n-1;
            int s = 0;

            while (t % 2 == 0) {
                s++;
                t /= 2;
            }

            //prvi dio provjere
            int b=fermatTest.computeAtoDModN(a,t,n);
            if (b == 1 || b == n - 1) return false;

            //drugi dio
            for (int j = 1; j <= s - 1; j++) {
                b = b * b % n;
                if (b == n - 1) return false;
            }

            return true;
        }
```

Ovo nas dovodi do sljedeće sekcije ovog poglavlja.

2.2 Miller–Rabinov test

Ovaj se test sastoji od uzastopnog slučajnog odabiranja baze za jaki test prostosti, te provođenja tog testa. Da bismo opravdali takav postupak, treba pokazati da vjerojatnost pogreške možemo proizvoljno smanjiti. U tu svrhu dokazujemo dvije leme koju su potrebne za dokaz teorema koji nam dopušta upravo to.

Lema 2.17. *Neka je n neparan složen broj, $n-1 = 2^s t$, t neparan. Neka je $v(n)$ najveći cijeli broj takav da $2^{v(n)}$ dijeli $p-1$ za svaki prosti faktor p od n. Ako je n jaki pseudoprosti broj za bazu a, onda vrijedi $a^{2^{v(n)-1}t} \equiv \pm 1 \pmod{n}$.*

Dokaz. Ukoliko je $a^t \equiv 1 \pmod{n}$, tvrdnja leme je očita. Pretpostavimo dakle, da vrijedi drugi dio definicije pseudoprostosti, tj. $a^{2^i t} \equiv -1 \pmod{n}$ za neki i, $0 \leq i \leq s-1$. Neka je p neki prosti faktor od n. Tada također vrijedi $a^{2^i t} \equiv -1 \pmod{p}$. Iz teorije grupa znamo da $a^s \equiv 1 \pmod{p}$ ako i samo ako $k \mid s$, gdje je k red od $a \pmod{p}$. Dakle, znamo da red od $a \pmod{p}$ dijeli $2^{i+1}t$, te da ne dijeli $2^i t$. To znači da se u rastavu k na proste faktore 2 pojavljuje s eksponentom točno $i+1$. No, k dijeli i $p-1$ (mali Fermatov teorem), pa i $2^{i+1} \mid p-1$. Kako ovo vrijedi za svaki prosti faktor p od n, zaključujemo $i+1 \leq \nu(n)$. Sada vidimo, ukoliko je $i+1 < \nu(n)$, $a^{2^{\nu(n)-1}t} \equiv 1 \pmod{n}$, a ako $i+1 = \nu(n)$, onda $a^{2^{\nu(n)-1}t} \equiv -1 \pmod{n}$. □

Za drugu lemu nam treba sljedeći poznati teorem kojeg navodimo bez dokaza, koji se može naći u [5].

Teorem 2.18 (**Kineski teorem o ostacima**). *Neka su $m_1, \ldots, m_r$ u parovima relativno prosti prirodni brojevi, te $a_1, \ldots, a_r$ cijeli brojevi. Tada sustav kongruencija*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

ima rješenja. Ako je z jedno rješenje tog sustava, onda su sva druga rješenja z' dana sa $z' \equiv z \pmod{m_1 m_2 \cdots m_r}$.

Definirajmo sada

$$\overline{S}(n) := \left\{a : a^{2^{\nu(n)-1}t} \equiv \pm 1 \pmod{n}\right\}, \quad \overline{S}(n) = \#\overline{S}(n).$$

Lema 2.19. *Označimo s $\omega(n)$ broj različitih prostih faktora od n. Vrijedi*

$$\overline{S}(n) = 2 \cdot 2^{(\nu(n)-1)\omega(n)} \prod_{p \mid n} nzm(t, p-1).$$

Dokaz. Označimo $m := 2^{\nu(n)-1}t$. Neka je $n = p_1^{j_1} \cdots p_k^{j_k}$ rastav broja n na proste faktore, gdje je $k = \omega(n)$. Budući da su svi $p_i^{j_i}$ potencije različitih prostih brojeva, pa onda i u parovima relativno prosti, imamo $a^m \equiv 1 \pmod{n}$ ako i samo ako $a^m \equiv 1 \pmod{p_i^{j_i}}$, za sve $i = 1, \ldots, k$. Znamo da je za neparan prost broj p i pozitivni cijeli broj j, grupa $\mathbb{Z}^*_{p^j}$ ciklična reda $p^{j-1}(p-1)$, tj. postoji primitivni korijen modulo p^j (dokaz ove tvrdnje može se naći u [5]). Dakle, broj rješenja $a \pmod{p_i^{j_i}}$ kongruencije $a^m \equiv 1 \pmod{p_i^{j_i}}$ je

$$nzm(m, p_i^{j_i-1}(p_i-1)) = nzm(m, p_i-1) = 2^{\nu(n)-1} nzm(t, p_i-1),$$

gdje $nzm(a,b)$ označava najveću zajedničku mjeru brojeva a i b. Prva jednakost vrijedi jer $m \mid n-1$, pa onda $p_i \nmid m$. Sada se pomoću Kineskog teorema o ostacima (2.18) može zaključiti da je broj rješenja $a \pmod n$ kongruencije $a^m \equiv 1 \pmod n$ dan s

$$\prod_{i=1}^{k} \left(2^{\nu(n)-1} nzm(t, p_i - 1)\right) = 2^{(\nu(n)-1)\omega(n)} \prod_{i=1}^{k} nzm(t, p_i - 1).$$

Dokažimo sada da točno toliko ima i rješenja kongruencije $a^m \equiv -1 \pmod n$. Uočimo da $a^m \equiv -1 \pmod{p_i^{j_i}}$ ako i samo ako $a^{2m} \equiv 1 \pmod{p_i^{j_i}}$ i $a^m \not\equiv 1 \pmod{p_i^{j_i}}$. Budući da $2^{\nu(n)}$ dijeli $p_i - 1$, kao i gore slijedi da je broj rješenja kongruencije $a^m \equiv -1 \pmod{p_i^{j_i}}$ jednak

$$2^{\nu(n)} \cdot nzm(t, p_i - 1) - 2^{\nu(n)-1} \cdot nzm(t, p_i - 1) = 2^{\nu(n)-1} \cdot nzm(t, p_i - 1).$$

Dakle, ima jednako rješenja za obje kongruencije, $a^m \equiv 1 \pmod n$ i $a^m \equiv -1 \pmod n$, čime je lema dokazana. □

Uvedimo sljedeću oznaku:

$$\mathcal{S}(n) := \left\{a\colon n \text{ je jaki pseudoprost broj za bazu } a\right\}, \quad S(n) = \#\mathcal{S}(n).$$

Sada možemo dokazati teorem koji opravdava Miller–Rabinov test:

Teorem 2.20. *Za svaki neparni složeni cijeli broj $n > 9$ imamo $S(n) < \frac{1}{4}\varphi(n)$.*

Dokaz. Iz Leme 2.17 slijedi da je dovoljno pokazati da vrijedi $\overline{S}(n)/\varphi(n) \leq \frac{1}{4}$ ako je n neparan složen broj veći od 9. Iz Leme 2.19 imamo

$$\frac{\varphi(n)}{\overline{S}(n)} = \frac{1}{2} \prod_{p^a \| n} p^{a-1} \frac{p-1}{2^{\nu(n)-1} nzm(t, p-1)},$$

gdje oznaka $p^a \parallel n$ znači da je p^a ona potencija prostog broja p koja se javlja u rastavu broja n na proste faktore. Svaki faktor $(p-1)/(2^{\nu(n)-1} nzm(t, p-1))$ je paran cijeli broj, tako da je $\varphi(n)/\overline{S}(n)$ cijeli broj. Nadalje, ako je $\omega(n) \geq 3$, slijedi da je $\varphi(n)/\overline{S}(n) \geq 4$. Ako je $\omega(n) = 2$ i n nije kvadratno slobodan (tj. djeljiv je nekim potpunim kvadratom), umnožak različitih p^{a-1} je najmanje 3, pa je $\varphi(n)/\overline{S}(n) \geq 6$.

Pretpostavimo sada da je $n = pq$, gdje su p,q prosti i $p < q$. Ako $2^{\nu(n)+1} \mid q-1$, onda je $2^{\nu(n)-1} nzm(t, q-1) \leq (q-1)/4$ i $\varphi(n)/\overline{S}(n) \geq 4$. Pretpostavimo sada da $2^{\nu(n)} \parallel q-1$. Uočimo da $n - 1 \equiv p - 1 \pmod{q-1}$, pa $q - 1 \nmid n - 1$. To implicira da postoji neparan prost broj koji u rastavu broja $q-1$ na proste faktore ima veću potenciju nego u rastavu $n-1$, tj. $2^{\nu(n)-1} nzm(t, q-1) \leq (q-1)/6$. U tom slučaju zaključujemo da je $\varphi(n)/\overline{S}(n) \geq 6$.

Konačno, pretpostavimo da je $n = p^a$, $a \geq 2$. Tada je $\varphi(n)/\overline{S}(n) = p^{a-1}$, pa vrijedi $\varphi(n)/\overline{S}(n) \geq 5$, osim ako je $p^a = 9$. □

Definicija 2.21. *Ako je n neparan složen broj i a takav da n* ***nije*** *jaki pseudoprost broj za bazu a, tada broj a zovemo* ***svjedokom*** *za broj n.*

Dakle, svjedok za n "svjedoči" da je n složen. Miller–Rabinov test traži svjedoka za broj n među brojevima $1, \ldots, n-1$, i to tako da nasumično odabere jedan broj i za njega provede jaki test prostosti. Prema Teoremu 2.20, ukoliko je n složen, vjerojatnost da slučajno odabrani broj a nije svjedok je $\leq \frac{1}{4}$. Ukoliko test ponavljamo k puta, vjerojatnost da se ni u jednoj iteraciji ne pojavi svjedok je $\leq (\frac{1}{4})^k$. Dakle, zaista možemo proizvoljno smanjiti vjerojatnost da je broj n složen. U nastavku donosimo kôd Miller–Rabinovog testa napisan u jeziku C#. Baziran je na pseudokodu danom u [2] i koristi ranije napisane metode.

```
1  /*metoda iterira jaki test prostosti za slučajno odabrane
   ↪  vrijednosti a, 1<a<n-1, te vraća true ukoliko je neki a svjedok
   ↪  za broj n, inače vraća false*/
2          public bool isNComposite(int n, int numberOfIterations) {
3              Random rnd = new Random();
4              for (int i = 0; i < numberOfIterations; i++) {
5                  int a = rnd.Next(2,n-1);
6                  if (strongPrimalityTest.testForCompositeness(a, n))
7                  {
8                      //a je svjedok za n, n je složen
9                      return true;
10                 };
11             }
12             //test nije našao niti  jednog svjedoka
13             return false;
14         }
```

Napomena 2.22. *Pretpostavimo da smo napravili* 20 *iteracija Miller–Rabinovog testa za neki dovoljno veliki broj n, te nismo uspjeli naći svjedoka za n. Vjerojatnost da je broj ipak složen je* 4^{-20}*, što je u praksi gotovo zanemarivo. Ipak, ne možemo sa sigurnošću znati je li n prost. Ukoliko bismo željeli biti sigurni, morali bismo provesti jedan od dokaza prostosti. Dokazi prostosti izlaze iz okvira ovog rada, ali je čitatelj upućen na [2] i [4], u kojima je detaljno razrađeno više takvih dokaza.*

2.3 Algoritam AKS

U prošloj smo sekciji vidjeli na koji način možemo u polinomnom vremenu s prilično velikom sigurnošću utvrditi je li broj prost. Dugo vremena pokušavalo se dobiti deterministički

algoritam koji bi uklonio faktor nesigurnosti, a svejedno ostao polinoman. Neki su algoritmi došli "vrlo blizu". Npr., algoritam testiranja prostosti Gaussovim sumama (opisan u [2]) je "skoro polinoman", s vremenom izvršavanja $(\ln n)^{c \ln\ln\ln n}$. Funkcija $\ln\ln\ln n$ nije ograničena odozgo, no raste jako sporo (u [2] autori duhovito kažu da ova funkcija možda ide u beskonačnost, ali još nitko nije vidio da je tamo došla). Tek su 2002. godine, indijski matematičari M. Agrawal, N. Kayal i N. Saxena ponudili deterministički polinomni algoritam. Zanimljivo je da su dvojica od trojice autora u vrijeme objavljivanja rada još imali samo prvostupničku diplomu. U ovom poglavlju donosimo obradu algoritma preuzetu iz originalnog članka ([1]).

Prvo navodimo pomoćnu tvrdnju, a zatim teorem koji je osnovica ovog algoritma.

Lema 2.23. *Ako je $n \in \mathbb{N}$ prost i $1 < k < n$, tada je binomni koeficijent $\binom{n}{k}$ djeljiv s n.*

Dokaz. Neka je $N = \binom{n}{k} = \frac{n!}{k!(n-k)!}$. Tada $n! = Nk!(n-k)!$. Naravno, n dijeli $n!$, pa mora dijeliti i desnu stranu jednakosti. No, budući da je n prost, i $k!$ i $(n-k)!$ su njim relativno prosti, pa mora vrijediti $n \mid N$. □

Teorem 2.24. *Neka je $a \in \mathbb{Z}$, $n \in \mathbb{N}$, $n \geq 2$ i $nzm(a, n) = 1$. Tada je n prost ako i samo ako vrijedi sljedeća jednakost u prstenu $\mathbb{Z}_n[x]$:*

$$(X + a)^n = X^n + a. \tag{2.1}$$

Dokaz. Za $0 < i < n$, koeficijent uz X^i u razvoju $((X + a)^n - (X^n + a))$ jednak je $\binom{n}{i} a^{n-i}$.

Ako je n prost, prema Lemi 2.23 vrijedi $\binom{n}{i} = 0 \pmod{n}$, tj. svi su koeficijenti 0.

Pretpostavimo sada da je n složen. Neka je q neki prosti faktor od n i $q^k \parallel n$ (sjetimo se oznake iz dokaza Teorema 2.20). Tada q ne dijeli $\binom{n}{q}$, i $nzm(a^{n-q}, q) = 1$. Prva tvrdnja vidi se iz $\binom{n}{q} = \frac{n(n-1)\ldots(n-q+1)}{q!}$, jer je brojnik djeljiv s q^k, i nijednom višom potencijom od q, a nazivnik je djeljiv s q i nijednom višom potencijom od q, tj. najveća potencija od q koja može dijeliti $\binom{n}{q}$ je q^{k-1}. Druga tvrdnja slijedi iz činjenica da je q prosti faktor od n i da je a relativno prost s n, pa i sa svim prostim faktorima od n. Dakle, koeficijent uz X^q nije nula $\pmod{n}$, pa $((X + a)^n - (X^n + a))$ nije identički jednak nuli u prstenu $\mathbb{Z}_n[x]$. □

Odmah je jasno da ovaj teorem daje jednostavan način provjere prostosti: za dani broj n naprosto odaberemo a i izračunamo gornju kongurenciju. Nažalost, takav bi postupak zahtijevao $\Omega(n)$ vremena, jer je potrebno evaluirati n koeficijenata s lijeve strane jednakosti. Treba, dakle taj broj smanjiti. Jednostavan način za to je reduciranje obje strane (2.1) polinomom oblika $X^r - 1$ za prikladno odabran mali r. Tada se problem svodi na testiranje sljedeće jednakosti:

$$(X + a)^n = X^n + a \pmod{X^{r-1}, n}. \tag{2.2}$$

Napomena 2.25. *Notacija u (2.2) predstavlja jednakost*

$$(X + a)^n = X^n + a$$

u prstenu $\mathbb{Z}_n[X]/(X^{r-1})$.

Označimo s $o_r(a)$ najmanji cijeli broj k takav da $a^k \equiv 1 \pmod{r}$. Broj $o_r(a)$ je takozvani *red od a modulo r*. Sada donosimo algoritam za provjeru prostosti preuzet iz [1]:

Algoritam 1 (AKS) Za dani složen broj $n > 1$ vraća PRIME ako je broj prost i COMPOSITE ako je složen.

1: **if** ($n = a^b$, za neki $a \in \mathbb{N}$, $b > 1$) **then return** COMPOSITE;
2: Nađi najmanji r takav da $o_r(n) > \log_2^2 n$;
3: **if** ($1 < nzm(a, n) < n$) za neki $a \leq r$ **then return** COMPOSITE;
4: **if** $n \leq r$ **then** return PRIME;
5: **for** $a = 1$ **to** $\lfloor \sqrt{\varphi(r)} \log_2 n \rfloor$ **do**
    if $\left(\left(X + a\right)^n \neq X^n + a \pmod{X^{r-1}, n}\right)$ **return** COMPOSITE;
   end for
6: return PRIME;

Analiza složenosti

Sljedeće su leme potrebne za analizu složenosti algoritma AKS.

Lema 2.26. *Označimo s NZV*(m) *najmanji zajednički višekratnik (NZV) prvih m brojeva. Za* $m \geq 7$ *vrijedi* $NZV(m) \geq 2^m$.

Lema 2.27. *Postoji* $r \leq \max\left\{3, \lceil \log_2^5 \rceil\right\}$ *takav da je* $o_r(n) > \log_2^2 n$.

Dokaz. Ako je $n = 2$, $r = 3$ zadovoljava sve uvjete. Pretpostavimo zato da je $n > 2$. Tada je $\lceil \log_2^5 n \rceil > 10$, pa možemo primijeniti Lemu 2.26. Neka su $r_1, \ldots, r_t$ svi brojevi takvi da ili $o_{r_i}(n) \leq \log_2^2 n$ ili r_i dijeli n. Svaki od tih brojeva mora dijeliti produkt

$$\prod_{i=1}^{\lfloor \log_2^2 n \rfloor} \left(n^i - 1\right) < n^{\log_2^4 n} \leq 2^{\log_2^5 n}.$$

Prema Lemi 2.26, NZV prvih $\lceil \log_2^5 n \rceil$ brojeva je najmanje $2^{\lceil \log_2^5 n \rceil}$, pa mora postojati broj s, $s \leq \log_2^5 n$ takav da $s \notin \{r_1, \ldots, r_t\}$. Ako je $nzm(s, n) = 1$, onda je $o_s(n) > \log_2^2$ i dokaz je gotov. Ako je, pak, $nzm(s, n) > 1$, onda, budući da s ne dijeli n i $nzm(s, n) \in \{r_1, \ldots, r_t\}$, $r = \frac{s}{nzm(s,n)} \notin \{r_1, \ldots, r_t\}$, pa mora biti $o_r(n) > \log_2^2 n$. □

Označimo s $O^{\sim}(t(n))$ vremensku složenost $O(t(n) \cdot poly(\log_2 t(n)))$.

Teorem 2.28. *Asimptotska vremenska složenost gornjeg algoritma je* $O^{\sim}(\log_2^{21/2} n)$.

Dokaz. Prvi korak algoritma je vremenske složenosti $O^{\sim}(\log_2^3 n)$, što je pokazano u [3].

U drugom koraku tražimo r takav da je $o_r(n) > \log_2^2 n$. To se može napraviti isprobavanjem uzastopnih vrijednosti r i testiranjem vrijedi li $n^k \not\equiv 1 \pmod r$ za svaki $k \leq \log_2^2 n$. Za određeni r, za to će trebati najviše $O(\log_2^2 n)$ množenja modulo n, pa će ih ukupno trebati $O^{\sim}(\log_2^2 n \log_2 r)$. Iz Leme 2.27 znamo da je potrebno provjeriti samo $O(\log_2^5 n)$ različitih r. Dakle, vremenska složenost drugog koraka je $O^{\sim}(\log_2^7 n)$

Treći korak se sastoji od izračunavanja najveće zajedničke mjere za r brojeva. Znamo da je Euklidov algoritam složenosti $O(\log_2 n)$, pa je vremenska složenost trećeg koraka $O(r \log_2 n) = O(r \log_2^6 n)$.

Vremenska složenost 4. koraka je samo $O(\log_2 r)$. Ovaj korak je bitan samo kad je $n \leq 5{,}690{,}034$, jer iz Leme 2.27 znamo da je $r \leq \lceil \log_2^5 n \rceil$.

U 5. koraku moramo provjeriti $\lfloor \sqrt{\varphi(r)} \log_2 n \rfloor$ jednadžbi, od kojih svaka zahtijeva $O(\log_2 n)$ množenja polinoma reda r koeficijentima veličine $O(\log_2 n)$. Dakle, svaka se jednadžba može provjeriti u vremenu $O^{\sim}(r \log_2^2 n)$. To znači da je vremenska složenost ovog koraka $O^{\sim}(r\lfloor \sqrt{\varphi(r)} \rfloor \log_2^3 n) = O^{\sim}(r^{\frac{3}{2}} \log_2^3 n) = O^{\sim}(\log_2^{21/2} n)$. Ovo vrijeme dominira vremena svih koraka, pa je to vremenska složenost algoritma. □

Pokazano je da se, uz neke hipoteze, vremenska složenost može još poboljšati.

Slutnja 2.29 (**Atkinova hipoteza**). *Za dani broj* $n \in \mathbb{N}$ *koji nije potpuni kvadrat, broj prostih brojeva* $q \leq m$ *za koje je* $o_q(n) = q - 1$ *je asimptotski* $A(m) \cdot \frac{m}{\ln m}$*, gdje je* $A(m) > 0.35$ *Atkinova konstanta.*

Slutnja 2.30 (**O gustoći prostih brojeva Sophie Germain**). *Broj prostih brojeva* $q \leq m$ *takvih da je* $2q + 1$ *također prost je asimtotski jednak* $\frac{2C_2 m}{\ln^2 m}$*, gdje je* C_2 *konstanta susjednih prostih brojeva (eng. twin prime constant), procijenjena na približno* 0.66*. Prosti brojevi s gornjim svojstvom zovu se* ***prosti brojevi Sophie Germain****, po francuskoj matematičarki koja ih je koristila u proučavanju posljednjeg Fermatovog teorema.*

Iz Atkinove hipoteze bi, ako vrijedi za $m = \log_2^2 n$, odmah slijedilo da postoji $r = O(\log_2^2 n)$ s traženim svosjstvima, dok iz hipoteze 2.30 možemo zaključiti da postoji $r = O^{\sim}(\log_2^2 n)$. Tada bi složenost algoritma bila $O^{\sim}(\log_2^6 n)$.

Uz sljedeću lemu, koju je dokazao Étienne Fouvry ([6]), možemo ipak nešto smanjiti složenost.

Lema 2.31. *Neka je* s $P(m)$ *označen najveći prost broj koji dijeli* m*. Postoje konstante* $c > 0$ *i* n_0 *takve da za sve* $x \geq n_0$ *vrijedi:*

$$\left| \left\{ q : q \text{ prost}, q \leq x, P(q-1) > q^{\frac{2}{3}} \right\} \right| \geq c \frac{x}{\ln x}.$$

Teorem 2.32. *Asimptotska vremenska složenost AKS algoritma je* $O^{\sim}(\log_2^{15/2} n)$.

Dokaz. Velika gustoća prostih brojeva q takvih da $P(q-1) > q^{\frac{2}{3}}$ implicira da će 2. korak algoritma naći r takav da $o_r(n) > \log_2^2 n$ u vremenu $O(\log_2^3 n)$. To spušta ogradu vremenske složenosti na $O^{\sim}(\log_2^{15/2} n)$. □

Važno je reći da postoji verzija algoritma koja je složenosti $O^{\sim}(\log_2^6 n)$. Tu su verziju pronašli Lenstra i Pomerance 2005. godine. Verzija tog rada može se naći na web-adresi `https://math.dartmouth.edu/~carlp/aks080709.pdf`. Oni su koristili Gaussove periode, umjesto korijena jedinice koji su korišteni u originalnom algoritmu koji je ovdje predstavljen.

Dokaz korektnosti algoritma AKS nije težak, ali zahtijeva razradu nekih pojmova iz teorije grupa, pa ćemo ga ispustiti. Kompletan dokaz se može naći u [1]. Originalni članak, s nadopunama iz 2004., može se naći i besplatno preuzeti na `http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf` ili na Microsoft Research(`https://www.microsoft.com/en-us/research/publication/primes-is-in-p/`).

Još valja napomenuti da je vrijednost ovog algoritma još uvijek uglavnom teorijska. Naime, vjerojatnosni algoritmi poput Miller–Rabinovog testa u prošloj sekciji su još uvijek mnogo brži, a za praktične probleme vjerojatnost pogreške je zanemariva.

Poglavlje 3

Metode faktorizacije

Druga cjelina ovog rada bavi se algoritmima za rastav broja na proste faktore. Donosimo tri algoritma koji se smatraju okosnicom suvremenog faktoriziranja: metodu kvadratnog sita (QS), metodu sita poljem brojeva (NFS), te metodu eliptičkih krivulja (ECM). Ti su algoritmi svi subeksponencijalne vremenske složenosti. Subeksponencijalni su oni algoritmi čije je vrijeme izvršavanja $2^{o(n)}$.

3.1 Metoda kvadratnog sita (QS)

Neka je n složen broj kojeg želimo faktorizirati. Ukoliko nađemo brojeve x i y takve da vrijedi $x^2 \equiv y^2 \pmod{n}$ i $x \neq \pm y$, tada smo našli i jedan faktor od n. Zaista, u tom slučaju $n \mid (x-y)(x+y)$, no n ne dijeli niti $(x-y)$, niti $(x+y)$. To znači da jedan dio n dijeli $(x-y)$, a drugi $(x+y)$, tj. $nzm(x-y, n) > 1$ je netrivijalni faktor od n. Postavlja se, dakako, pitanje kako naći takve x i y? Na to pitanje odgovara metoda kvadratnog sita.

Napomena 3.1. *Uočimo da, ako je n potencija prirodnog broja, nećemo moći naći tražene brojeve. Zaista, QS se bavi samo neparnim brojevima koji nisu potencije. No, to i nije ograničenje. Naime, ukoliko je broj paran već znamo jedan njegov faktor, te preostaje faktorizirati neparni dio. Također, provjera je li n potencija može se izvršiti pomoću vađenja k-tog korijena iz n, te provjere je li* $n = (\lceil n^{\frac{1}{k}} \rceil)^k$. *Vađenje korijena može se napraviti Newtonovom metodom, te postupak treba napraviti za sve* $k < \log_2 n$.

Osnovna ideja QS metode je pronaći kongruencije $x_i^2 \equiv a_i \pmod{n}$, tako da je $\prod a_i$ kvadrat modulo n, tj. $\prod a_i \equiv y^2 \pmod{n}$ za neki y. Tada za $x = \prod x_i$ vrijedi $x^2 \equiv y^2 \pmod{n}$. Uvjet $x \neq y$ se ignorira. Ukoliko se pokaže da su nađeni x i y jednaki ili suprotni, naprosto pokušamo s drugim parom kongruentnih kvadrata. Vidjet ćemo da to možemo, i to deterministički, tako da QS nije vjerojatnosni algoritam.

Primjer 3.2. *Uzmimo broj* 1649. *Počinjemo tražiti* x^2 *među najmanjim potpunim kvadratima većim od n, tj. za x uzimamo redom brojeve* $\lceil\sqrt{n}\rceil$, $\lceil\sqrt{n}\rceil + 1, \ldots$.

$$\begin{aligned} 41^2 &\equiv 32 \pmod{1649} \\ 42^2 &\equiv 115 \pmod{1649} \\ 43^2 &\equiv 200 \pmod{1649}. \end{aligned}$$

Dobili smo $41^2 \cdot 43^2 \equiv 32 \cdot 200 = 6400 = 80^2$. *Također, imamo* $41 \cdot 43 = 1763 \equiv 114 \pmod{1649}$*, pa smo dobili ono što smo i tražili. Sada izračunamo* $nzm(114 - 80, 1649) = 17$*, te smo tako dobili jedan faktor. Drugi dobijemo, naprosto, dijeljenjem* $1649 \div 17 = 97$.

Ovaj je primjer jednostavan, ali što je s velikim brojevima, za koje nećemo odmah naći prave x_i? Promatramo skup brojeva $x^2 \pmod{n}$, za x koji prolazi po brojevima počevši s $\lceil\sqrt{n}\rceil$ (nazovimo ga $\mathcal{Q}$). Želimo, dakle, pronaći podskup tog skupa čiji članovi u umnošku daju kvadrat modulo n (nazovimo taj podskup $\mathcal{QS}$).

Potragu započinjemo redukcijom problema. Prvo, uočimo da, ako u skup $\mathcal{QS}$ uvrstimo x_1^2 koji ima veliki prosti faktor na neparnu potenciju, morat ćemo naći i drugi x_2^2 s istim velikim prostim faktorom. Zato takve brojeve izbacujemo (kao npr. 115 u primjeru, koji ima faktor 23, koji je relativno velik u odnosu na faktore u ostalim kongruencijama).

Definicija 3.3. *Ukoliko pozitivni cijeli broj n nema prostih faktora većih od B, zovemo ga* ***B-glatkim****.*

Dakle, recimo da se odlučimo za neki B, te izbacimo sve kongruencije u kojima se pojave brojevi koji nisu B-glatki. Sljedeće je pitanje koliko nam B-glatkih brojeva u skupu $\mathcal{S}$ treba da bismo bili sigurni da postoji podskup $\mathcal{QS}$. Za odgovor na ovo pitanje trebat će malo linearne algebre.

Neka je $m = \prod p_i^{e_i}$ neki B-gladak broj, gdje su $p_1, \ldots, p_{\pi(B)}$ prosti brojevi manji od B i $e_i \geq 0$. Takvom broju m pridružimo vektor eksponenata:

$$\vec{v}(m) = (e_1, \ldots, e_{\pi(B)}).$$

Ako su $m_1, \ldots, m_k$ svi B-glatki, onda je $\prod_{i=1}^k m_i$ kvadrat ako i samo ako vektor $\sum_{i=1}^k \vec{v}(m_i)$ ima sve komponente parne.

Nadalje, eksponente možemo reducirati modulo 2, te ih promatrati u vektorskom prostoru nad poljem skalara $\mathbb{F}_2 = (\{0, 1\}, +, \cdot)$. Dakle, naš je vektorski prostor sada $\mathbb{F}_2^{\pi(B)}$. Sjetimo se, tražimo podskup skupa $\mathcal{S}$ čiji elementi u umnošku daju kvadrat. Prevedeno u naš vektorski prostor, tražimo linearnu kombinaciju vektora koja daje vektor $(0, \ldots, 0)$, tj. tražimo linearno zavisan skup u prostoru dimenzije $\pi(B)$. Sada znamo koliko brojeva moramo uzeti u skup S da bismo bili sigurni da možemo naći linearno zavisan podskup. Trebamo ih najmanje $\pi(B) + 1$.

Dakle, sada imamo teorem koji nam kaže koliko ostataka $x_2 \pmod n$ moramo naći i algoritam za njihovo nalaženje. No, još imamo problem koji nismo riješili: kako odabrati B. Ukoliko odaberemo mali B, nećemo morati imati velik skup $\mathcal{S}$, pa će biti lakše naći $\mathcal{QS}$. No, s druge strane, moglo bi se dogoditi da je teško uopće dobiti bilo koji B-gladak broj za skup $\mathcal{S}$, jer to svojstvo nije pretjerano uobičajeno. Potrebno je naći dovoljno velik B da B-glatkih brojeva bude dovoljno, a opet dovoljno malen da ih ne moramo imati previše za traženje podskupa. Također, postavlja se i pitanje koliko nam treba da provjerimo je li x^2 B-gladak za neki x. Na prvi pogled, kao odgovor bi se moglo ponuditi $\pi(B)$, budući da jednostavnim testom dijeljenja (*eng.* trial division) možemo to lako provjeriti. No, postoji brži način. Pomoću varijanti Eratostenovog sita (opisanih u [2]), ovo se vrijeme može svesti na $\ln \ln B$. Te metode zahtijevaju upotrebu prostih brojeva i potencija prostih brojeva kod kojih je potencija najveća moguća, koja bi mogla dijeliti neku od vrijednosti $x^2 - n$. Prosti brojevi p, koji su baze tih potencija, su oni za koje $x^2 - n \equiv 0 \pmod p$ ima rješenja. To je broj $p = 2$ i neparni prosti brojevi $p \leq B$, za koje je Legendrov simbol $\left(\frac{n}{p}\right) = 1$. Za svaki od takvih brojeva p i svaku bitnu potenciju od p, postoje dvije klase ostataka za provjeru. Neka je K broj prostih brojeva manjih ili jednakih B koje koristimo za provjeru. Budući da, heuristički, nemamo razloga smatrati da neki brojevi imaju veću vjerojatnost da budu B-glatki od drugih, te prema tome nemamo razloga diskriminirati ni proste brojeve koje bi trebalo promatrati, možemo smatrati da je K približno $\frac{1}{2}\pi(B)$. Dakle, da bismo bili sigurni u linearnu zavisnost među našim vektorima eksponenata, dovoljno nam je njih $K + 1$.

U [2] je, heurističkom analizom, vjerojatnost da x dovede do B-glatkog broja procijenjena na u^{-u}, gdje je $u = \frac{1}{2} \ln / \ln B$. Tada je očekivani broj x-ova da dobijemo jedan takav u^u, a vjerojatnost da pogodaka bude $K + 1$ je $u^u(K + 1)$. To još pomnožimo s $\ln \ln B$, očekivanim vremenom potrebnim za obradu svakog x, pa dobivamo očekivano vrijeme izvršavanja ovisno o B:

$$T(B) = u^u(K + 1) \ln \ln B, \text{ gdje je } u = \frac{\ln n}{2 \ln B}.$$

Minimiziranjem ove funkcije dobivamo da je optimalna vrijednost B približno jednaka $\exp(\frac{1}{2}\sqrt{\ln n \ln \ln n})$, te je procijenjeno vrijeme izvršavanja gornjih koraka $\exp(\sqrt{\ln n \ln \ln n})$ (dakle B^2). Također, može se pokazati i da je vrijeme potrebno za korak koji koristi linearnu algebru, također, približno B^2. To znači da je vrijeme izvršavanja algoritma jednako $L(n)^{1+o(1)}$, uz oznaku

$$L(n) = \exp(\sqrt{\ln n \ln \ln n}).$$

Ova je funkcija subeksponencijalna, tj. oblika $n^{o(1)}$.

Algoritam

Sada donosimo algoritam u cijelosti, preuzet iz [2].

Algoritam 2 (QS) Za dani neparan složen broj $n > 1$ koji nije potencija, vraća netrivijalni faktor.

1: $B = \lceil L(n)^{1/2} \rceil$;
$p_1 = 2, a_1 = 1$;
Nađi proste brojeve $p \leq B$ takve da je $\left(\frac{n}{p}\right) = 1$ i označi ih s $p_1, \ldots, p_K$;
for $(2 \leq i \leq K)$
Nađi $\pm a_i$ takve da je $a_i^2 \equiv n \pmod{p_i}$;
end for

2: Među vrijednostima $(x^2 - n)$ za $x \in \{\lceil \sqrt{n} \rceil, \lceil \sqrt{n} \rceil + 1, \ldots\}$, nađi one koji su B-glatki, dok se ne dobije $K + 1$ par $(x, x^2 + 1)$, koje stavljamo u skup $\mathcal{S}$;

3: **for** $((x, x^2 + 1) \in \mathcal{S})$ **do**
Nađi rastav na proste faktore $(x^2 - n) = \prod_{i=1}^{K} p_i^{e_i}$;
$\vec{v}(x^2 - n) \leftarrow (e_1, \ldots, e_K)$;
end for

4: **if** $n \leq r$ **then** return PRIME;

5: **for** $a = 1$ **to** $\lfloor \sqrt{\varphi(r)} \log_2 n \rfloor$ **do**
if $\left(\left(X + a\right)^n \neq X^n + a \pmod{X^{r-1}, n}\right)$ **return** COMPOSITE;
end for
Stvori $(K + 1) \times K$ matricu s recima $\vec{v}(x^2 - n)$ reduciranima (mod 2);
Nađi linearno zavisni podskup redaka (označimo ih s $\vec{v}(x_1), \ldots, \vec{v}(x_k)$);

6:
$x \leftarrow x_1 \cdots x_k \pmod{n}$;
$y \leftarrow \sqrt{(x_1^2 - n) \cdots (x_k^2 - n)}$;
$d \leftarrow nzm(x - y, n)$;
return d;

3.2 Metoda NFS

Ova metoda se zasniva na istom principu kao i QS. Kao što smo vidjeli, QS kreće od jedne strane kongruencije, koja već jest kvadrat, i traži drugu. NFS polazi od druge ideje.

Strategija

Neka θ leži u nekom prstenu algebarskih brojeva $\mathbb{R}$, i neka je $\phi(\theta)$ homomorfizam s tog prstena u $\mathbb{Z}_n$. Pretpostavimo da imamo k parova $(\theta_i, \phi(\theta)_i)$ takvih da je $\prod \theta_i$ kvadrat u prstenu $\mathcal{R}$ (recimo γ^2), te $\prod \phi(\theta_i)$ kvadrat u $\mathbb{Z}_n$ (recimo v^2). Sada, ako je $\phi(\gamma) \equiv u \pmod{n}$ za neki cijeli broj u, onda imamo $u^2 \equiv v^2 \pmod{n}$, što smo i tražili. Sada možemo, kao i u QS, naći faktor $nzm(u - v, n)$.

Neka je n neparan složen broj koji nije potencija, te neka je

$$f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_0$$

ireducibilni polinom u $\mathbb{Z}_n[x]$. Odaberimo neki α i promatrajmo prsten $\mathbb{Z}_\alpha$. Možemo ga promatrati kao skup d-torki $(a_0, \ldots, a_{d-1})$, od kojih svaka predstavljaja element $a_0 + a_1\alpha + \ldots + a_{d-1}\alpha^{d-1}$. Takva dva entiteta zbrajamo po komponentama, a množimo kao polinome uz redukciju preko identiteta $f(\alpha) = 0$. Veza s brojem n kojeg želimo faktorizirati dolazi od cijelog broja m za kojeg vrijedi

$$f(m) \equiv 0 \pmod{n}.$$

Kako naći takve $f(x)$ i m? Nasreću, to je vrlo lako, ukoliko odaberemo d. U analizi složenosti pokazuje se da je optimalan d približno $\left(\frac{3\ln n}{\ln\ln n}\right)^{1/3}$ ([2]). Stavimo $m = \lfloor n^{1/d} \rfloor$ i raspišimo n u bazi m:

$$n = m^d + c_{d-1}m^{d-1} + \ldots + c_0,$$

gdje su svi $c_j \in [0, \ldots, m-1]$. Sada je jasno da će $f(x)$ biti upravo taj polinom:

$$f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_0.$$

Taj je polinom, očito, normiran, ali je li ireducibilan? Ustvari, ako nije, to je dobra situacija za faktoriziranje. Naime, ukoliko uspijemo dobiti netrivijalnu faktorizaciju (što je relativno lako) $f(x) = g(x)h(x)$, tada smo dobili i netrivijalnu faktorizaciju broja $n = f(m) = g(m)h(m)$. Ukoliko je, pak f ireducibilan, nastavljamo dalje s algoritmom NFS.

Homomorfizam prstena ϕ sa $\mathbb{Z}[\alpha]$ na $\mathbb{Z}_n$ definiramo ovako:

$$a_0 + a_1\alpha + \ldots + a_{d-1}\alpha^{d-1} \longmapsto a_0 + a_1 m + \ldots + a_{d-1}m^{d-1} \pmod{n}.$$

Elementi θ u prstenu $\mathbb{Z}[\alpha]$ koje ćemo promatrati bit će oblika $a - b\alpha$, gdje su $a, b \in \mathbb{Z}$ relativno prosti.

Da sumiramo, tražimo skup $\mathcal{S}$ relativno prostih parova (a, b) takvih da vrijedi

$$\prod_{(a,b)\in\mathcal{S}} (a - b\alpha) = \gamma^2, \text{ za neki } \gamma \in \mathbb{Z}[\alpha],$$

$$\prod_{(a,b)\in\mathcal{S}} (a - bm) = v^2, \text{ za neki } v \in \mathbb{Z}_n.$$

Tada, ako je $\phi(\gamma) \equiv u \pmod{n}$, imamo $u^2 \equiv v^2 \pmod{n}$. Parovi u $\mathcal{S}$ su relativno prosti da bi se izbjegle redundancije.

Ovo traženje je nešto kompliciranije nego kod QS, pa ga ispuštamo, te algoritam ilustrativno primijenjujemo na konkretan broj. Detaljna razrada s obradom svih problema koji mogu iskrsnuti nalazi se u [2].

Algoritam

Počinjemo s nekoliko definicija.

Definicija 3.4. *Racionalna faktorska baza je konačan skup prostih brojeva.*

Definicija 3.5. *Neka je $\mathcal{R}$ racionalna faktorska baza. Kažemo da je cijeli broj n $\mathcal{R}$-gladak ako $\mathcal{R}$ sadrži sve proste djelitelje broja n.*

Definicija 3.6. *Algebarska faktorska baza je konačni skup $\{a + b\theta\} \subset \mathbb{Z}[\theta]$, gdje za svaki element $a + b\theta$ vrijedi da se ne može netrivijalno faktorizirati, tj. ne postoje $c, d \in \mathbb{Z}[\theta]$ takvi da je $a + b\theta = c \cdot d$ i $c, d \neq 1$.*

Sljedeća je činjenica važna za promatranje algebarskih baza.

Lema 3.7. *Neka je $f(x) \in \mathbb{Z}[n]$ i $\theta \in \mathbb{C}$ neka nultočka od f. Tada postoji bijekcija između algebarske baze i skupa parova (r, p), gdje je p prost, $r \in \mathbb{Z}_n$, te $f(r) \equiv 0 \pmod{p}$.*

Napomena 3.8. *Gornja lema efektivno kaže da algebarsku bazu možemo "prevesti" u promatranje cjelobrojnih parova, što uvelike olakšava donošenje zaključaka.*

Definicija 3.9. *Neka je $\mathcal{A}$ algebarska baza. Kažemo da par (s, q) dijeli $(a + b\theta) \in \mathbb{Z}[\theta]$ ako je (s, q) reprezentacija nekog elementa $(c + d\theta) \in \mathcal{A}$ i $(c + d\theta) | (a + b\theta)$.*

Definicija 3.10. *Neka je $\mathcal{A}$ algebarska faktorska baza. Kažemo da je $l \in \mathbb{Z}[\theta]$ $\mathcal{A}$-gladak ako postoji $W \subset \mathcal{A}$ takav da $\prod_{(c,d)\in W}(c + d\theta) = l$.*

Definicija 3.11. *Neka je U skup parova (a, b) takav da je $\prod_{(a,b)\in U}(a + b\theta)$ kvadrat u $\mathbb{Z}[\theta]$. Neka je $\mathcal{Q}$ skup parova (s, q) takav da za svaki $(s, q) \in \mathcal{Q}$ vrijedi:*

1. $(\forall (a, b) \in U)$ (s, q) *ne dijeli* $(a + b\theta)$
2. $\forall (s, q) \in \mathcal{Q}$ *umnožak Legendreovih simbola*

$$\prod_{(a,b)\in U} \left(\frac{a + bs}{q} \right) = 1.$$

Tada skup $\mathcal{Q}$ nazivamo bazom kvadratnih simbola, a njegove elemente kvadratnim simbolima.

Sljedeći je primjer preuzet sa stranice `http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.219.2389`, gdje se, također, može naći opis algoritma.

Primjer 3.12 (**Algoritam NFS**). *Želimo faktorizirati broj* $n = 45113$. *Pretpostavimo da znamo da je broj složen i da nije potencija. Budući da je očito neparan, možemo iskoristiti algoritam NFS.*

Kao što smo rekli, potrebno je izabrati polinom f *i cijeli broj* m, *no odaberimo prvo* d, *stupanj polinoma* f. *Iz prošle sekcije dobivamo* $d \approx \left(\frac{3 \ln 45113}{\ln \ln 45113}\right)^{1/3} = 2.38$, *pa ćemo uzeti* $d = 3$. *Iako je* $\lfloor 45113^{1/d} \rfloor = 35$, *autori su za* m *uzeli broj* 31. *Iako to nije u skladu s našim razmatranjima, držat ćemo se primjera radi jednostavnosti. Naime, broj* m *također možemo odabrati proizvoljno, dok god možemo naći polinom* f *tako da* $f(m) \equiv 0 \pmod{n}$.

Prikaz broja 45113 *u bazi* $m = 31$ *je* $45113 = 31^3 + 15 \cdot 31^2 + 29 \cdot 31 + 8$. *Dakle, definiramo:*

$$f(x) = x^3 + 15x^2 + 29x + 8.$$

Sljedeći je korak odabiranje skupova koji će nam poslužiti kao faktorske baze za traženje glatkih brojeva. Trebamo dva takva skupa: jedan za algebarske glatke brojeve (u prstenu $\mathbb{Z}[\theta]$*) i jedan za racionalne (u prstenu* $\mathbb{Z}_n$*).*

Za racionalnu faktorsku bazu možemo samo uzeti sve proste brojeve manje od 30*:*

$$\mathcal{R} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}.$$

Za nalaženje algebarske faktorske baze koristimo činjenicu da se dovoljno njenih elemenata može prikazati kao parovi (r, p) *takvi da je* $f(r) \equiv 0 \pmod{p}$, *za neke proste brojeve* p. *U ovom primjeru, proizvoljno su odabrani prosti brojevi* $p < 90$. *Za svaki* p *traženi su* r_i *tako da je* $f(r_i \equiv 0 \pmod{p})$. *Ova je potraga dala sljedeći skup:*

$$\begin{aligned}\mathcal{A} = \{&(0,2), (6,7), (13,17), (11,23), (26,29), (18,31), (19,41),\\ &(13,43), (1,53), (46,61), (2,67), (6,67), (44,67), (50,73),\\ &(23,79), (47,79), (73,79), (28,89), (62,89), (73,89)\}.\end{aligned}$$

Uočimo da je $k := |\mathcal{R}| = 10$ *i* $l := |\mathcal{A}| = 20$. *To će poslije biti važno.*

Uz definirane faktorske baze, trebamo naći i bazu kvadratnih simbola. Naprosto odaberemo neke proste brojeve koji se nisu pojavili u algebarskoj bazi, npr. 97, 101, 103, 107. *Sada za svaki odabrani* q *tražimo sve* s *tako da je* $f(s) \equiv 0 \pmod{q}$. *To daje bazu kvadratnih simbola*

$$\mathcal{Q} = \{(28, 97), (87, 101), (47, 103), (4, 107), (8, 107), (80, 107)\}.$$

Uočimo da je $u := |\mathcal{Q}| = 6$. *Onda nam treba više od* $1 + k + l + u = 37$ *parova* (a, b) *takvih da je* $a + bm$ $\mathcal{R}$*-gladak i* $a + b\theta$ $\mathcal{A}$*-gladak.*

Sada na sljedeći način tražimo glatke brojeve.

1. *Fiksiramo* $b \in \mathbb{Z}$ *i odaberemo proizvoljni cijeli broj* N.

2. *Pustimo a da varira između N i $-N$. Stvorimo dva niza: jedan za vrijednosti $a + b\theta$ i jedan za vrijednosti $a + bm$.*

3. *Isto kao u QS, prosijemo brojeve $a + bm$ s brojevima iz $\mathcal{R}$, te zapamtimo one koji su $\mathcal{R}$-glatki.*

4. *Tražimo vrijednosti a koje zadovoljavaju uvjet $(r_i, p_i) \in \mathcal{A}$ dijeli $a + b\theta$ ako i samo ako $a \equiv -br_i \pmod{p_i}$, za neki element (r_i, p_i) od $\mathcal{A}$, te pamtimo taj faktor za a. Poslije tog postupka, za sve $a + b\theta$ imat ćemo listu faktora iz $\mathcal{A}$. Ukoliko za neki $a + b\theta$ vrijedi $\prod_{(r_i,p_i)|(a+b\theta)} p_i = (-b)^d f(-a/b)$, $a + b\theta$ je $\mathcal{A}$-gladak.*

5. *Uzmemo one (a, b) koji su i $\mathcal{R}$-glatki i $\mathcal{A}$-glatki.*

Ovaj postupak možemo ponavljati mijenjajući b, sve dok ne dobijemo dovoljno parova (a, b).

U našem primjeru, uzeli smo da je $-400 \leq a \leq 400$ i $1 \leq b \leq 41$. Uz pomoć gornje metode, nađeno je sljedećih 38 parova (a, b) koji su istovremeno $\mathcal{R}$-glatki i $\mathcal{A}$-glatki:

$$\begin{aligned} U = \Big\{ & (-73,1),(-13,1),(-6,1),(-2,1),(-1,1),(1,1),(2,1),(3,1),(13,1),(15,1),\\ & (23,1),(61,1),(1,2),(3,2),(33,2),(2,3),(5,3),(19,4),(14,5),(37,5),(313,5),\\ & (11,7),(15,7),(-7,9),(119,11),(-247,12),(9,41),(175,13),(5,17),(-1,19),\\ & (35,19),(17,25),(49,26),(375,29),(9,32),(1,33),(78,37),(5,41) \Big\}. \end{aligned}$$

Definiramo matricu X, koja se razlikuje od matrice koju smo koristili u algoritmu QS, ali služi istoj svrsi. Za svaki par $(a, b) \in \mathcal{U}$, pripadni red matrice izgleda ovako:

- *prva komponenta retka je 0 ako je $a + bm > 0$, inače je 1*
- *sljedećih k komponenata odgovara vektoru eksponenata istom kao u QS*
- *daljnih l komponenata odgovara vektoru eksponenata $(a + b\theta)$ (on, naime, ima faktorizaciju u $\mathbb{Z}[\theta]$, pogledati definiciju glatkosti)*
- *posljednih u komponenti odgovara bazi kvadratnih simbola: za svaki par $(s, q) \in \mathcal{Q}$, vrijednost komponente je 0 ako Legendrov simbol $\left(\frac{a+bs}{q}\right) = 1$, inače je 1.*

Sada, kao i u QS, tražimo linearno zavisni podksup redaka matrice X, koji će nam reći koje elemente skupa U trebamo pomnožiti da bismo dobili potpune kvadrate u oba prstena.

Ispada da je jedan takav podskup onaj skup redaka koji odgovara sljedećim elementima:

$$\begin{aligned} V = \{ & (-2,1),(1,1),(13,1),(15,1),(23,1),(3,2),(33,2),(5,3),(19,4),\\ & (14,5),(15,7),(119,11)(175,13),(-1,19),(49,26)\}. \end{aligned}$$

Sada smo dobili:

$$\prod_{(a,b)\in V}(a+bm) = 45999712751795195582606376960000,$$

$$\prod_{(a,b)\in V}(a+b\theta) = 58251363820606365\cdot\theta^2 + 149816899035790332\cdot\theta + 75158930297695972.$$

Nadalje:

$$25530453172224002^2 = \prod_{(a,b)\in V}(a+bm),$$

$$(108141021\cdot\theta^3 + 235698019\cdot\theta + 62585630)^2 = \prod_{(a,b)\in V}(a+b\theta).$$

Napokon, možemo iskoristiti homomorfizam ϕ:

$$\phi(108141021\cdot\theta^3 + 235698019\cdot\theta + 62585630) = 111292745400.$$

Sada možemo zaključiti:

$$111292745400^2 \equiv 25530453172224002^2 \pmod{n}.$$

Pokušaj faktoriziranja, uz $x = 111292745400$ *i* $y = 25530453172224002$*, daje:*

$$nzm(x+y, 45113) = 197,$$
$$nzm(x-y, 45113) = 229.$$

Dakle, NFS algoritam je uspješno faktorizirao broj 45113.

Složenost

Analizom složenosti sličnom onoj za QS, Crandall i Pomerance su pokazali ([2]) da je složenost ovog algoritma dana asimptotski s

$$\exp\left(\left((64/9)^{1/3} + o(1)\right)(\ln n)^{1/3}(\ln\ln n)^{2/3}\right).$$

To čini ovaj algoritam trenutno najbržim algoritmom, a time i najrašireniji, od svih subeskponencijalnih algoritama. Ipak, zbog svoje složenosti, ta je superiornost asimptotska, tj. očituje se za velike brojeve. Za manje brojeve bolji su jednostavniji algoritmi, pa se u praksi pojavljuju razni hibridi eksponencijalnih i subeksponencijalnih algoritama, gdje se prvi koriste za pronalaženje malih faktora, a drugi za faktoriziranje velikih.

Napomena 3.13. *U ovom smo algoritmu koristili normirani polinom $f(x)$. No, za neke brojeve n mogu se naći posebno "dobri" polinomi koji olakšavaju faktoriziranje. Zato se algoritam koji je ovdje predstavljen naziva općim NFS (eng. General NFS, GNFS), dok se u drugom slučaju algoritmi nazivaju posebnim NFS (eng. Special NFS, SNFS).*

3.3 Metoda ECM

U metodama NFS i QS vidjeli smo da vrijeme izvršavanja ovisi o veličini broja n kojeg faktoriziramo. Algoritmi koji se baziraju na tom principu nazivaju se *algoritmima druge kategorije*. *Algoritmi prve kategorije* su algoritmi poput ECM, čija vremenska složenost ovisi o veličini najmanjeg prostog faktora od n.

Za poptuno razumijevanje ovog algoritma, čitatelju se preporuča upoznavanje s osnovnom aritmetikom eliptičkih krivulja. Potrebna pozadina za ovaj algoritam može se naći u [2].

Lenstrina ECM metoda

Iako se koristi mnogim konceptima iz teorije eliptičkih krivulja, ova metoda ih primjenjuje na konstrukt koji to nije.

Promotrimo polinom u dvije varijable u prstenu $\mathbb{F}[x, y]$ izjednačen s nulom:

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j = 0. \tag{3.1}$$

Afina rješenja ove jednadžbe su parovi $(x, y) \in \mathbb{F} \times \mathbb{F}$ koji zadovoljavaju gornju jednadžbu. No, možemo promatrati i projektivna rješenja. To su uređene trojke $(x, y, z) \in \mathbb{F} \times \mathbb{F} \times \mathbb{F}$ koje su rješenja sljedeće jednadžbe:

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2z + fxyz + gy^2z + hxz^2 + iyz^2 + jz^3 = 0. \tag{3.2}$$

Uočimo da je (x, y, z) rješenje jedndažbe (3.2) ako i samo ako je (tx, ty, tz), također, rješenje te jednadžbe, pri čemu je $t \in \mathbb{F}$, $t \neq 0$. Zato, u projektivnom slučaju, govorimo o rješenjima $[x, y, z]$ koja predstavljaju sva rješenja oblika (tx, ty, tz).

Afina i projektivna rješenja su povezana na sljedeći način: ako je (x, y) afino rješenje jednadžbe (3.1), onda je $[x, y, 1]$ rješenje jednadžbe (3.2). Ako je, pak $[x, y, z]$ rješenje (3.2), za neki $z \neq 0$, onda je $(x/z, y/z)$ rješenje (3.1). Rješenja jednadžbe (3.2) oblika $[x, y, 0]$ nemaju odgovarajuću afinu točku. Njih zovemo *točkama u beskonačnosti* krivulje dane jednadžbom (3.2). Ukoliko ta jednadžba ima bar jedno netrivijalno rješenje čije su koordinate u $\mathbb{F}$, te ako je nesingularna, zovemo ju ***jednadžbom eliptičke krivulje nad*** $\mathbb{F}$.

Definicija 3.14. *Neka su $a, b \in \mathbb{Z}_n$ i $nzm(n, 6) = 1$, te neka vrijedi $nzm(4a^3 + 27b^2, n) = 1$.* ***Eliptička pseudokrivulja*** *je skup*

$$E_{(a,b)}(\mathbb{Z}_n) = \{ (x, y) \in \mathbb{Z}_n \times \mathbb{Z}_n : \ y^2 = x^3 + ax + b \} \cup \{O\},$$

gdje je O točka u beskonačnosti.

Pokazuje se da, ako je n prost, za krivulju danu jednadžbom iz gornje definicije postoji samo jedna točka u beskonačnosti, te sve točke na krivulji čine grupu uz sljedeću operaciju zbrajanja.

Definicija 3.15. *Neka je $\mathbb{F}$ polje čija je karakteristika $\neq 2, 3$, te neka je $E(\mathbb{F})$ eliptička krivulja nad poljem $\mathbb{F}$ dana u Weierstrassovoj formi:*

$$y^2 = x^3 + Cx^2 + Ax + B, \quad A, B, C \in \mathbb{F}.$$

Označimo s O točku u beskonačnosti i s $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ dvije (ne nužno različite) točke na krivulji E. Definiramo operaciju zbrajanja točaka ovako:

(1) $-O = O$,

(2) $-P_1 = (x_1, -y_1)$,

(3) $O + P_1 = P_1$,

(4) ako $P_2 = -P_1$, *onda* $P_1 + P_2 = O$,

(5) ako $P_2 \neq -P_1$, *onda* $P_1 + P_2 = (x_3, y_3)$, *gdje:*

$$\begin{aligned} x_3 &= m^2 - C - x_1 - x_2, \\ -y_3 &= m(x_3 - x_1) + y_1. \end{aligned}$$

Tu je nagib m definiran s:

$$f(n) = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{ako } x_2 \neq x_1, \\ \dfrac{3x_1^2 + 2Cx_1 + A}{2y_1}, & \text{ako } x_2 = x_1. \end{cases}$$

Kod nas je n složen, pa postoji više točaka u beskonačnosti, no mi dopuštamo svejedno samo jednu, onu predstavljenu s $[0, 1, 0]$.

Kada n nije prost, $\mathbb{Z}_n$ nije polje, pa pri računanju $P + Q$ nećemo uvijek moći naći inverz potreban za računanje nagiba m u definiciji 3.15. To znači da točke na pseudokrivulji ne čine grupu ako je n složen (zbog toga naziv *pseudokrivulja*). Nasreću, postoje moćni alati za iskorištavanje koncepta pseudogrupe. Jedan takav je i prošireni Euklidov algoritam. Naime, taj će algoritam, pri pokušaju nalaženja inverza broja a u $\mathbb{Z}_n$, ako a nije invertibilan, pronaći neki njegov faktor. Upravo na toj ideji se temelji Lenstrin algoritam: probamo naći inverz, i ako ne uspijemo – uspjeli smo!

Algoritam 3 (Lenstrin ECM algoritam) Za dani složen broj n, uz $nzm(n, 6) = 1$ i n nije potencija, pronalazi netrivijalni faktor.

Odaberi B_1; ▷ neka gornja granica, pogledati analizu složenosti
[Nađi krivulju $E_{(a,b)}(\mathbb{Z}_n)$ i točku $(x, y) \in E$]:
    Odaberi x,y,$a \in [0, n-1]$;
    $b \leftarrow (y^2 - x^3 - ax) \pmod{n}$;
    $g \leftarrow nzm(4a^3 + 27b^2, n)$;
    if $(g == n)$ **goto** 2;
    if $(g > 1)$ **return** g;
    $E \leftarrow E_{(a,b)}(\mathbb{Z}_n)$; ▷ Krivulja
    $P \leftarrow (x, y)$; ▷ Točka na krivulji
for $(1 \leq i \leq \pi(B_1))$ **do** ▷ petlja po prostim brojevima manjim od B_1
Nađi najveći a_i t.d. $p_i^{a_i} \leq B_1$;
    for$(1 \leq j \leq a_i)$ **do**
        $P \leftarrow [p_i]\, P$;
        Stani ako ne postoji inverz nazivnika d nagiba ▷ vidi definiciju 3.15
        i vrati $g = nzm(n, d)$;
    end for

end for
[Promašaj] ▷ Nismo uspjeli naći netrvijalan faktor od n
    Povećaj granicu B_1;
    goto 2;

Ideja ECM je, dakle, da nam ilegalna operacija traženja inverza signalizira da za neki prosti faktor p od n imamo:

$$[k]P = O, \text{ gdje je } k = \prod_{p_i^{a_i} \leq B_1} p_i^{a_i},$$

za neku pravu eliptičku krivulju $E_{(a,b)}(\mathbb{F}_p)$.

Teorem 3.16 (Hasse)**.** *Red $\#E$ eliptičke krivulje $E_{(a,b)}(\mathbb{F}_{p^k})$ zadovoljava*

$$\left|(\#E) - (p^k + 1)\right| \leq 2\sqrt{p^k}.$$

Iz Hasseovog teorema 3.16 slijedi da je red $\#E_{(a,b)}(\mathbb{F}_p)$ u intervalu $(p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p})$. Očito, možemo očekivati faktorizaciju ako je k djeljiv s $\#E(\mathbb{F}_p)$, što će se i dogoditi ako je $\#E(\mathbb{F}_p)$ B_1-gladak. Dakle, B_1 možemo interpretirati kao granicu glatkoće za redove krivulja u grupi određenoj (skrivenim) faktorom p.

Složenost

Neka je p najmanji prosti faktor od n. U [2] provedena je heuristička analiza složenosti kojom je utvrđeno da B_1 treba odabrati tako da vrijedi

$$B_1 = \exp\left(\left(\sqrt{2}/2 + o(1)\right)\sqrt{\ln p \ln\ln p}\right)$$

i za tu ogradu, složenost je procijenjena na

$$\exp\left(\left(\sqrt{2} + o(1)\right)\sqrt{\ln p \ln\ln p}\right).$$

Naravno, odmah uočavamo glavni problem: ne možemo unaprijed znati p, pa ni predvidjeti B_1. Zato se obično kreće od nekog relativno malog B_1 (npr. 10000) i onda ga se povećava, ako je potrebno. U praksi, B_1 se povećava nakon dovoljno pokušaja s istim B_1. Kako se B_1 povećava i napokon približava kritičnoj vrijednosti, sve je veća vjerojatnost za uspjeh, te je vrijeme provedeno s manjim B_1 zanemarivo.

Dakle, heuristička procjena složenosti algoritma ECM, primijenjenog na složen broj n s najmanjim prostim faktorom p, je $L(p)^{\sqrt{2}+o(1)}$ koraka s cijelim brojevima veličine n. Iz toga slijedi da je broj koraka to veći što je veći najmanji prosti faktor od n. Najgori slučaj nastupa kad je n produkt dva prosta faktora približno jednake veličine. U tom je slučaju očekivani broj koraka $L(n)^{1+o(1)}$, što je i vremenska složenost QS. Ipak, za najgori slučaj se koriste QS ili NFS, zbog veće preciznosti koraka u ECM.

Za slučaj kad ne znamo je li n instanca najgoreg slučaja, obično se preporuča probati algoritam ECM, pa tek nakon dovoljno vremena inicijalizirati QS ili NFS.

Ako, pak znamo da je broj n prevelik za QS i NFS, onda je ECM jedina metoda koja je preostala. Koristeći nju, može nam se "posrećiti" na dva načina. Jedan je da n ima dovoljno mali prosti faktor da ga ECM nađe. Drugi način je da nabasamo na pravi izbor parametara ranije nego što je očekivano. U stvari, zanimljivo svojstvo ECM-a je velika varijanca u broju koraka. To je posljedica činjenice da se uspjeh ECM-a oslanja na samo jedan povoljan događaj.

Za kraj, valja napomenuti da se heuristička analiza složenosti može potpuno formalizirati, osim pretpostavke da cijeli brojevi u Hasseovom intervalu imaju jednaku vjerojatnost da budu glatki kao i svi ostali cijeli brojevi.

Bibliografija

[1] Manindra Agrawal, Neeraj Kayal i Nitin Saxena, *Primes is in P*, Annals of Mathematics (2004), br. 160, 781–793.

[2] Richard Crandall i Carl B. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, 2010.

[3] Joachim Von Zur Gathen i Jurgen Gerhard, *Modern Computer Algebra*, Cambridge University Press, New York, NY, USA, 2003, ISBN 0521826462.

[4] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser Verlag, 1985.

[5] Victor Shoup, *A computational introduction to number theory and algebra*, Cambridge University Press, 2006.

[6] Étienne Fouvry, *Théorème de Brun-Titchmarsh; application au théorème de Fermat*, Inventiones Mathematicae (1985), br. 79, 383–407.

Sažetak

U ovom je radu prikazano šest algoritama, od kojih su tri testovi prostosti, a tri faktorizacijski algoritmi.

Faktorizacijski algoritmi su oni koji za dani broj n pronalaze neki rastav broja n u smislu osnovnog teorema aritmetike (Teorem 1).

Testovi prostosti su algoritmi koji za dani broj n odgovaraju na pitanje je li n prost. Iako se i faktorizacijski algoritmi mogu koristiti u tu svrhu, u praksi to nije uobičajeno. Naime, puno lakše je odrediti složenost broja nego dati konkretnu faktorizaciju, u smislu vremenske složenosti. Dok su najbrže faktorizacijske metode subeksponencijalne složenosti, testovi prostosti su polinomni. Zato se faktorizacijske metode koriste kao testovi prostosti samo za male brojeve.

Testovi prostosti koji su obrađeni su:

- Fermatov test koji je osnova mnogih testova prostosti koji se danas koriste
- Miller–Rabinov test koji je predstavnik vjerojatnosnih testova prostosti
- algoritam AKS kao jedini poznati polinomni test prostosti.

U praksi se najviše koriste upravo algoritmi poput Miller–Rabinovog testa, koji imaju mogućnost pogreške, ali se ona može proizvoljno smanjiti. Algoritam AKS je još uvijek najviše od teorijskog značaja.

Od faktorizacijskih algoritama predstavljeni su neki od danas najkorištenijih algoritama koji su svi subeksponencijalne složenosti:

- metoda kvadratnog sita (QS)
- metoda sita poljem brojeva (NFS)
- metoda eliptičkih krivulja (ECM).

Algoritmi koji, poput ECM, ovise o veličini najmanjeg prostog faktora broja kojeg želimo faktorizirati zovu se *algoritmi prve kategorije*. Algoritmi QS i NFS ovise isključivo o veličini samog broja kojeg želimo faktorizirati i zovu se *algoritmi druge kategorije*.

Za svaki od algoritama objašnjen je princip na kojem se zasnivaju, te osnovni pojmovi potrebni za njihovo razumijevanje.

Summary

In this paper six algorithms are presented: three primality tests and three factorization algorithms.

Factorization algorithms (methods) are those algorithms that give some decomposition of a given integer n in the sense of Fundamental theorem of arithmetic (1).

Primality test are those algorithms that, for a given positive integer n, decide whether n is prime or composite. Although factoriaztion algorithms can also be used for that purpose, it is not common to do so. If they are used for testing, they are only useful for small numbers, because of their complexity. Fastest known factorization methods, as one can see in this paper, are of subexponential time complexity, while primality tests are polynomial. One might find this logical, since it is far easier to decide whether a number is prime, than to give a concrete factorization.

Primality tests discussed here are:

- Fermat test, which is a corner stone of many primality tests,
- Miller–Rabin test, which represents nondeterministic, but fast primality tests,
- algorithm AKS, which is so far the only known polynomial algorithm.

In practice, AKS usually gives way to algorithms such as Miller–Rabin test, because it is faster, and its nondeterministic component can be arbitrarily reduced to satisfyingly small probability of mistake. Therefore, the importance of AKS is, for now, mostly theoretical.

Factorization methods chosen to be represented are:

- quadratic sieve method (QS),
- number field sieve (NFS),
- elliptic curve method (ECM).

It is stated here that the time complexity of ECM depends strongly on the size of the least prime factor of the number it is trying to factor. Algorithms with this feature are

called *first category algorithms*. Complexity of QS and NFS depends solely on the size of the number which they are trying to factor. Such algorithms are called *Second Category* or *Kraitchik family* algorithms. All three algorithms are of subexponential time complexity, which means that they are faster than exponential algorithms, but slower than any polynomial algorithm (of course, asymptotically).

For each algorithm discussed in this paper, the principle on which it is based is explained, as are the basic concepts necessary for understanding it.

Životopis

Zvonimir Iveković je 2010. godine završio gimnaziju Lucijana Vranjanina u Zagrebu, te upisao Prirodoslovno–matematički fakultet Sveučilišta u Zagrebu. 2013. je završio preddiplomski sveučilišni studij matematike stekavši prvostupničku diplomu, te upisao diplomski sveučilišni studij Računarstvo i matematika. U akademskoj godini 2014./2015. proveo je jedan semestar na međunarodnoj razmjeni na Jagiellonskom sveučilištu u Krakowu u sklopu programa Erasmus.