Osnovni algoritmi teorije brojeva

Strmečki, Josip

Sveučilište J. J. Strossmayera u Osijeku

Odjel za matematiku

Josip Strmečki

Osnovni algoritmi teorije brojeva

Diplomski rad

Osijek, 2019.

Sveučilište J. J. Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni diplomski studij matematike, smjer Matematika i računarstva

Josip Strmečki

Osnovni algoritmi teorije brojeva

Diplomski rad

Mentor: izv. prof. dr. sc. Ivan Matić

Osijek, 2019.

Sadržaj

1 Uvod

Središnji objekti ovog rada su algoritmi korišteni u teoriji brojeva i kriptografiji. Algoritam je metoda, "recept", za rješavanje određene klase problema, koja za ulazne podatke daje odgovor (izlazne podatke) u konačnom vremenu. Algoritmi, kao alati za rješavanje problema, predmeti su proučavanja od samih početaka matematike. Jedan od najstarijih algoritama koji se i danas koristi je Euklidov algoritam za određivanje najvećeg zajedničkog djelitelja dvaju cijelih brojeva. Nagli razvoj računarstva u 20. stoljeću dao je novi značaj proučavanju algoritama. Daljnje detalje, kao i klasifikaciju algoritama u ovisnosti o efikasnosti dajemo u poglavlju 3. S napretkom u komunikaciji, pojavila se potreba za zaštitom povjerljivih podataka, te brzim i efikasnim provođenjem operacija nad velikim brojevima. Više o definiciji i razvoju kriptosustava dajemo u poglavlju 2. Jedan od temeljnih pojmova u teoriji brojeva je pojam djeljivosti. Kažemo da prirodan broj d dijeli broj a ukoliko postoji prirodan broj k takav da je $a = kd$ i pišemo $d|a$. U kriptografiji prosti brojevi zauzimaju posebno mjesto. To su cijeli brojevi veći od 1 koji imaju samo dva djelitelja: 1 i samog sebe. Za broj koji nije prost kažemo da je složen. Broj 1 nije niti prost niti složen. Posljedice sljedećeg teorema su nužne za daljnje proučavanje kriptografije, i teorije brojeva uopće.

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

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

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

Ideja korištenja velikih prostih brojeva u kriptografiji dolazi iz činjenice da nije poznat dovoljno efikasan algoritam za faktorizaciju. Ostatak rada, odnosno poglavlja 4-8 bave se algoritmima koji rješavaju osnovne probleme u teoriji brojeva i kriptografiji i njihovom složenosti.

2 Uvod u kriptografiju

Kroz povijest je problem sigurne komunikacije uvijek bio prisutan, dok je danas zbog povećanja broja komunikacijskih kanala taj problem još važniji. Rješavanjem ovih problema bavi se znanstvena disciplina *kriptografija*. Sama riječ kriptografija je grčkog podrijetla i mogla bi se doslovno prevesti kao tajnopis. Osnovni zadatak kriptografije je omogućiti komunikaciju dvije osobe, zovemo ih *pošiljalac* i *primalac*[1] na takav način da treća osoba, protivnik, koja može nadzirati komunikacijski kanal tu poruku ne može razumjeti. Ovu definiciju ne moramo shvaćati doslovno, pošiljalac i primalac mogu biti na primjer banka i klijent, poruka brojevi računa, a protivnik bilo koja treća osoba.

Poruku koju pošiljalac želi poslati zovemo *otvoreni tekst*. Postupak transformacije otvorenog teksta pomoću unaprijed dogovorenog *ključa* K zovemo *šifriranje*, a dobiveni tekst *šifrat*. Nakon šifriranja poruku šaljemo preko komunikacijskog kanala koji može biti korumpiran, međutim ukoliko protivnik nema ključ K, ne može odrediti sadržaj originalne poruke. Za razliku od njega, primalac korištenjem ključa K može *dešifrirati* poruku i odrediti otvoreni tekst. Sljedeća definicija formalizira gore navedene pojmove.

Definicija 2.1. *Kriptosustav je uređena petorka* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, *gdje je* $\mathcal{P}$ *konačan skup svih otvorenih tekstova,* $\mathcal{C}$ *konačan skup svih šifrata,* $\mathcal{K}$ *konačan skup svih mogućih ključeva,* $\mathcal{E}$ *skup svih funkcija šifriranja i* $\mathcal{D}$ *skup svih funkcija dešifriranja. Za svaki* $K \in \mathcal{K}$ *postoji* $e_K \in \mathcal{E}$ *i odgovarajući* $d_K \in \mathcal{D}$. *Pritom su* $e_K : \mathcal{P} \to \mathcal{C}$ *i* $d_K : \mathcal{C} \to \mathcal{P}$ *funkcije sa svojstvom da je* $d_K(e_K(x)) = x$ *za svaki* $x \in \mathcal{P}$.

Gornja shema predstavlja tzv. *simetrični ili konvencionalni kriptosustav*. Funkcije šifriranja i dešifriranja ovise o ključu K koji pošiljalac i primalac moraju tajno razmijeniti prije same komunikacije. Ova činjenica predstavlja veliki problem, jer njima nije dostupan siguran komunikacijski kanal.

Jedno od rješenja ovog problema je učiniti ključ javnim. Diffie i Helhman[2] smatraju se začetnicima *kriptografije javnog ključa*. Ideja ovakvog sustava sastoji se od toga da je iz poznavanja funkcije šifriranja e_K nemoguće u razumnom vremenu izračunati funkciju dešifriranja d_k. Tada bi funkcija e_K mogla biti javna. Dakle, u kriptosustavu s javnim ključem svaki korisnik K ima dva ključa: javni e_K i tajni d_K. Ako Alice želi Bobu poslati poruku, tada je ona šifrira pomoću Bobovog javnog ključa e_B, tj. šalje šifrat $y = e_B(x)$. Bob dešifrira poruku koristeći svoj tajni ključ d_B, $d_B(y) = d_B(e_B(x)) = x$. Primijetimo da Bob mora posjedovati neku dodatnu informaciju (*trapdoor* - skriveni ulaz) o funkciji e_B da bi samo on mogao izračunati inverz d_B. Takve funkcije čiji je inverz teško izračunati bez poznavanja neke dodatne informacije zovemo *osobne jednosmjerne funkcije.*

Vrijedi spomenuti da su kriptosustavi s javnim ključem puno sporiji od modernih simetričnih kriptosustava (DES, IDEA), pa se u praksi ne koriste za šifriranje poruka, već za šifriranje ključeva, koji se potom koriste u simetričnim kriptosustavima. Još jedna važna primjena kriptosustava s javnim ključem je ta što omogućava da se poruka *digitalno potpiše*, tj. ako Alice Bobu pošalje šifrat $z = d_A(e_B(x))$, tada Bob može biti siguran da je poruku poslala Alice, jer jedino ona zna funkciju d_A.

Kao što smo rekli ranije, kriptosustavi s javnim ključem zasnivaju se na teškim matematičkim problemima. Jedan od takvih problema je i problem faktorizacije prirodnih brojeva. Do danas nije razvijen efikasan algoritam za faktorizaciju svih prirodnih brojeva, pa je za sada opravdano reći

[1] u literaturi se često zovu i Bob i Alice

[2] Whitfield Diffie i Martin Hellman, 1976. g.

da je gotovo nemoguće rastaviti na faktore pažljivo odabran prirodni broj koji ima više od 200 znamenaka (u nekom razumnom vremenu).

Najpoznatiji kriptosustav s javnim ključem koji koristi gore opisanu ideju je RSA[3] kriptosustav koji ćemo i formalno definirati.

RSA kriptosustav: Neka je $n = pq$, gdje su p i q prosti brojevi.
Neka je $\mathcal{P} = C = \mathbb{Z}_n$, te
$$\mathcal{K} = \{(n, p, q, d, e) \ : \ n = pq, \ de \equiv 1 \pmod{\varphi(n)}\}.$$
Za $K \in \mathcal{K}$ definiramo
$$e_K(x) = x^e \pmod{n}, \quad d_K(y) = y^d \pmod{n}, \quad x, y \in \mathbb{Z}_n.$$

Vrijednosti n i e su javne, a vrijednosti p, q, d su tajne, tj. (n, e) čini javni ključ, a (p, q, d) tajni ključ, i gdje je $\varphi(n)$ Eulerova funkcija. Prisjetimo se, $\varphi(n)$ je funkcija koja broji broj prirodnih brojeva manjih od n koji nemaju zajedničkih djelitelja s n. Jedno od svojstava Eulerove funkcije koje navodimo bez dokaza jest da je ona multiplikativna, tj. vrijedi $\varphi(1) = 1$ i $\varphi(pq) = \varphi(p)\varphi(q)$ ako p i q nemaju zajedničkih djelitelja većih od 1. Dakle, imamo $\varphi(n) = \varphi(pq) = (p-1)(q-1) = n - p - q + 1$.

Sigurnost RSA kriptosustava temelji se na pretpostavci da je funkcija $e_K(x) = x^e \pmod{n}$ jednosmjerna. Dodatni podatak (trapdoor) koji možemo iskoristiti za dešifriranje je poznavanje faktorizacije $n = pq$. Zaista, ukoliko znamo faktorizaciju, tada vrlo lako možemo izračunati $\varphi(n) = (p-1)(q-1)$, te eksponent d dobiti rješavanjem linearne kongruencije

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

U poglavlju 6 ćemo pokazati proceduru za rješavanje takvih kongruencija pomoću Euklidovog algoritma.

[3] nazvan po tvorcima Riestu, Shamiru i Adelmanu, 1977. g.

3 Složenost algoritama

Neformalno, *algoritam* je dobro definirana procedura koja za određene ulazne podatke daje izlazne podatke u konačnom vremenu. Isto tako, algoritam možemo gledati kao alat za rješavanje dobro definiranog računarskog problema. Iskaz problema određuje željenu vezu između ulaza i izlaza, dok algoritam opisuje specifičan niz koraka kojima ostvarujemo tu vezu. Na primjer, jedan od problema s kojim se često susrećemo je sortiranje niza brojeva u neopadajućem poretku. Evo kako bismo formalno definirali *problem sortiranja.*

Ulaz: Niz od n cijelih brojeva $(a_1, a_2, ..., a_n)$.
Izlaz: Permutacija $(a'_1, a'_2, ..., a'_n)$ ulaznog niza takva da je $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Na primjer, za zadani ulaz (30, 51, 1, 56), algoritam za sortiranje vraća niz (1, 30, 51, 56). Takav ulazni niz nazivamo *instanca* problema. Problem sortiranja se u praksi vrlo često pojavljuje, bilo kao samostalan problem ili kao potproblem nekog složenijeg problema, pa je razvijeno mnogo algoritama koji ga rješavaju. (pogledati [3].)

Osnovna svojstva algoritama su: **definiranost** - uz svaki algoritam moraju biti definirani početni podaci nad kojima se obavljaju operacije, i **konačnost** - algoritam mora biti sastavljen od konačno mnogo koraka koji definiraju slijed operacija koje se vrše nad podacima kako bi dobili rezultat. Za algoritam kažemo da je *korektan* ako za svaku instancu daje točno rješenje. Kažemo da korektan algoritam *rješava* dani problem.

Analizirati algoritam znači procijeniti resurse koje će algoritam koristiti. Najčešći resurs koji želimo procijeniti je vrijeme računanja, odnosno broj "osnovnih koraka". Osnovnim korakom smatramo jednu "bitnu operaciju", tj. logičku operaciju disjunkcije, konjukcije ili negacije nad bitovima.

Definicija 3.1. *Vremenska složenost algoritma koji za ulaz ima cijele brojeve je dana brojem izvršenih bitnih operacija prilikom rada algoritma.*

Vrlo često je teško, pa i nemoguće odrediti točan broje operacija koje algoritam izvede. Međutim, za usporedbu algoritama i njihovo proučavanje dovoljno nam je promatrati asimptotsko ponašanje broja operacija u odnosu na veličinu ulaza. Kako računalo radi u binarnom sustavu, veličinu ulaznih podataka mjerit ćemo kao broj bitova potrebnih za njihov prikaz. Tu je korisno uvesti pojam "duljine" ili veličine broja.

Definicija 3.2. *Neka je n cijeli broj i β cijeli broj veći od* 1. *Kažemo da je k duljina broja n u bazi β, u oznaci $L_\beta(n)$, ako vrijedi:*

$$\beta^{k-1} \leq n < \beta^k,$$

što znači, $n = \sum_{i=0}^{k-1} a_i\beta^i = (a_{k-1}, ..., a_0)_\beta$, gdje je $a_{k-1} \neq 0$.

Primjer 1. *Imamo da je $L_{10}(205) = 3$, dok je $L_2(205) = 8$, jer je $205 = (11001101)_2$.*

Ukoliko bi nejednakost iz definicije logaritmirali, dobivamo

$$k - 1 \leq \log_\beta(n) < k$$

pa je $L_\beta(k) = \lfloor \log_\beta(n) \rfloor + 1$. Dakle, duljina cijelog broja n u bazi β jednaka je $\lfloor \log_\beta(n) \rfloor + 1$. Prilikom analize složenosti algoritama koristi se sljedeća notacija.

Definicija 3.3. *Neka su $f, g : \mathbb{R} \to \mathbb{R}$ dvije funkcije. Tada pišemo:*

(i) $f(n) = O(g(n))$ *ako postoje* $B, C > 0$ *tako da je* $|f(n)| \leq C|g(n)|$ *za sve* $n > B$;

(ii) $f(n) \sim g(n)$ *ako je* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$;

(iii) $f(n) = o(g(n))$ *ako je* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Primjer 2. *Pokažimo da je* $3n^3 + n^2 + 2n + 6 = O(n^3)$.
Prema definiciji, za $B = 1$ *imamo* $3n^3 + n^2 + 2n + 6 \leq 3n^3 + n^3 + 2n^3 + 6n^3 \leq 12n^3 = O(n^3)$. *Isto tako vidimo i da je* $3n^3 + n^2 + 2n + 6 = O(n^4)$, *međutim u praksi se uvijek uzima najbolja ocjena.*

Prilikom ocjene broja operacija, razlikujemo ocjene za

- broj operacija u najlošijem slučaju (proizvoljan input) - **složenost algoritma**;
- prosječan broj operacija (prosjek za sve ulaze fiksne duljine) - **prosječna složenost algoritma.**

Navodimo klase funkcija na koje često nailazimo prilikom analiziranja složenosti algoritama. Funkcije koje sporije rastu su prije navedene i c je proizvoljna konstanta.

oznaka	**ime**
$O(1)$	konstantno
$O(log(n))$	logaritamsko
$O((log(n))^c)$	polilogaritamsko
$O(n)$	linearno
$O(n^2)$	kvadratno
$O(n^c)$	polinomijalno
$O(c^n)$	eksponencijalno

Definicija 3.4. *Polinomijalan algoritam je algoritam čiji je broj operacija u najlošijem slučaju reda veličine* $O(n^k)$, *gdje je n veličina ulaza, a k konstanta. Algoritme koji nisu polinomijalni zovemo eksponencijalni.*

Vrlo često za polinomijalne akgoritme kažemo da su "brzi" ili "efikasni", a za eksponencijalne da su "spori" ili "neefikasni". Iako nam ovakva analiza govori puno o efikasnosti algoritma, u praksi treba biti oprezan. Treba imati u vidu konstantu C i stupanj polinoma. Na primjer, ako jedan algoritam radi $100n$ koraka, reći ćemo da je $O(n)$ složenosti. Neka je drugi algoritam složenosti $O(n)^2$. Vidimo da je drugi algoritam asimptotski sporiji od prvog, međutim za male vrijednosti n, drugi algoritam je brži. Nadalje, u kriptografiji je često važniji prosječan broj operacija od broja operacija u najlošijem slučaju, jer želimo da kriptosustav složen u prosjeku, a ne u izoliranom slučaju. Iz definicije 3.4. i činjenice da je duljina cijelog broja n u bazi $L_2(k) = \lfloor \log_2(n) \rfloor + 1 = O(\log_2(n))$ slijedi da je algoritam na takvom ulazu polinomijalan ako je složenosti $O(\log_2(n)^k)$.

Isto tako primijetimo da je vremenska složenost logaritamskih funkcija neovisna o bazi. Zaista, iz dobro poznatog svojstva logaritama

$$\log_\beta(n) = \frac{\log_\alpha(n)}{\log_\alpha(\beta)}$$

vidimo da je $O(\log_\beta(n)) = O(\log_\alpha(n))$.

Umjesto analiziranja algoritama, možemo i same probleme koje ti algoritmi rješavaju klasificirati na lake i teške.

Definicija 3.5. *Klasa složenosti **P** se sastoji od svih problema odluke za koje postoji polinomijalan algoritam. Klasa složenosti **NP** se sastoji od svih problema odluke za koje se odgovor DA može provjeriti u polinomijalnom vremenu korištenjem dodatne informacije, certifikata. Klasa složenosti **co-NP** se definira na isti način za odgovor NE. Gdje je problem odluke svaki problem na koji je odgovor DA ili NE.*

Vrijedi $\mathbf{P} \subset \mathbf{NP}$ i $\mathbf{P} \subset \mathbf{co\text{-}NP}$. Pitanje vrijedi li $\mathbf{P} = \mathbf{NP}$ je otvoreno i smatra se jednim od najvažnijih otvorenih pitanja u matematičkoj teoriji računarstva. Postoji još mnogo klasa složenosti, za detalje pogledati u [3]

4 Osnovne računske operacije

Prije proučavanja složenijih algoritama iz teorije brojeva, pogledajmo složenost osnovnih računskih operacija nad prirodnim brojevima.

4.1 Zbrajanje i oduzimanje prirodnih brojeva

Iz same definicije bitnih operacija, jasno je da zbrajanje i oduzimanje prirodnih brojeva x i y možemo obaviti u $O(\ln N)$ bitnih operacija, pri čemu vrijedi $x, y \leq N$.
Neka su sada $x = (x_n, ..., x_1, x_0)_b$ i $x = (y_n, ..., y_1, y_0)_b$ dva prirodna broja zapisana u bazi b. Tada zbroj brojeva $x + y = (w_{n+1}, w_n, ..., w_1, w_0)_b$ računamo na sljedeći način:

Algoritam za zbrajanje:

c = 0
for ($0 \leq i \leq n$) {
    if ($x_i + y_i + c < b$) then $w_i = x_i + y_i + c$ and $c = 0$
        else $w_i = x_i + y_i + c - b, c = 1$ }
$w_{n+1} = c$
return w

Oduzimanje provodimo na sličan način, te zbog toga ne navodimo algoritam.

4.2 Množenje i dijeljenje prirodnih brojeva

Neka su $x = (x_n, ..., x_1, x_0)_b$ i $y = (y_n, ..., y_1, y_0)_b$ dva prirodna broja zapisana u bazi b. Želimo izračunati njihov umnožak $x \cdot y = (w_{n+t+1}, w_n, ..., w_1, w_0)_b$.
Jedan od pristupa je "školsko" ("naivno") množenje.

Algoritam za "školsko" množenje:

for $(0 \leq i \leq n + t + 1)$ $w_i = 0$
for $(0 \leq i \leq t)$ {
    $c = 0$
    for $(0 \leq j \leq t)$ {
        $(uv)_b = w_{i+j} + x_j \cdot y_j + c; w_{i+j} = v; c = u$ }
    $w_{n+t+1} = u$ }
return w

Iz algoritma je vidljivo da je njegova složenost $O(n^2)$, odnosno da je vremenska složenost množenja $O(\ln^2 N)$.
Pogledajmo sada algoritam za dijeljenje prirodnih brojeva s ostatkom. Neka su x i y kao gore i dodatno neka je $n \geq t \geq 1$. Želimo naći kvocijent $q = (q_{n-t}, ..., q_0)_b$ i ostatak $r = (r_t, ..., r_0)_b$ pri dijeljenju broja x brojem y, tj. brojeve q i r koji zadovoljavaju $x = qy + r, 0 \leq r < y$. Postojanje takvih brojeva garantira nam sljedeći teorem kojeg navodimo bez dokaza, dokaz možemo pronaći u ([2], str. 14)

Teorem 4.1. *Za proizvoljan prirodan broj y i cijeli broj x postoje jedinstveni cijeli brojevi q i r takvi da je $x = qy + r, 0 \leq r < y$.*

Algoritam za dijeljenje s ostatkom:

```
for  (0 ≤ i ≤ n − t) q_i = 0
while  (x ≥ y b^{n−t}) q_{n−t} = q_{n−t} + 1; x = x − y b^{n−t}
for  (n ≥ i ≥ t − 1)  {
      if  (x_i = y_t)  then  q_{i−t−1} = b − 1
            else  q_{i−t−1} = ⌊(x_i b + x_{i−1})/y_t⌋
      while  (q_{i−t−1}(y_t b + y_{t−1}) > x_i b^2 + x_{i−1} b + x_{i−2})
            q_{i−t−1} = q_{i−t−1} − 1
      x = x − q_{i−t−1} y b^{i−t−1}
      if  (x < 0) x = x + y b^{i−t−1}; q_{i−t−1} = q_{i−t−1} − 1
r = x
return q, r
```

Analizom gornjeg algoritma vidimo da mu je složenost $O(\ln^2 N)$.

Za razliku od zbrajanja i oduzimanja, kod množenja i dijeljenja postoje algoritmi koji su, barem u teoriji, dosta efikasniji od gore navedenih "naivnih" algoritama. Najbolji poznati algoritmi za množenje i dijeljenje imaju složenost

$$O(\ln N(\ln\ln N)(\ln\ln\ln N)).$$

(Schönhage-Strassenov algoritam, 1971. g.)
Cjeloviti algoritam i detalji mogu se naći u [7]
Nadalje, uočimo da je

$$\ln N(\ln\ln N)(\ln\ln\ln N) = O((\ln N)^{1+\epsilon}), \forall \epsilon > 0.$$

Dakle, vidimo da je množenje tek neznatno složenije od zbrajanja. No, to je ipak teoretski zaključak koji ignorira ogromnu konstantu koja se krija iza O. Takvi (najbolji) teoretski algoritmi koriste *brzu Fourierovu transformaciju* (FFT). Njihova primjena dolazi do izražaja tek za brojeve od nekoliko tisuća znamenki. Međutim, postoje algoritmi koji su bolji od "naivnog", a koji su od praktične važnosti za brojeve od stotinjak znamenki, kakvi se danas uglavnom rabe u kriptografiji.

Jedan od takvih algoritama je tzv. *Karatsubina metoda* (1962.) za množenje prirodnih brojeva. Karatsubina metoda koristi "podijeli pa vladaj" tehniku za množenje dvaju brojeva, odnosno množenje brojeva s velikim brojem znamenki zamjenjuje se nekolicinom množenja, ali ovaj puta množimo brojeve s manje znamenki.
Neka su $x = (x_{2n-1}, ..., x_1, x_0)_2$ i $y = (y_{2n-1}, ..., y_1, y_0)_2$ dva $2n$-bitna prirodna broja, te pretpostavimo da je n potencija broja 2. Zapišimo ih u obliku

$$x = 2^n u_1 + u_0, y = 2^n v_1 + v_0$$

dakle, u_1 i v_1 su lijeve polovice, a u_0 i v_0 desne polovice od x, odnosno y respektivno.
Sada je

$$x \cdot y = 2^{2n} u_1 v_1 + 2^n (u_1 v_0 + u_0 v_1) + u_0 v_0. \tag{1}$$

Vidimo da smo produkt $2n$-bitnih brojeva sveli na računanje četiri produkta n-bitnih brojeva, te kao što smo i naveli, to je glavna ideja Karatsubine metode. Račun možemo još dodatno pojednostavniti, jer je

$$u_1 v_0 + u_0 v_1 = u_1 v_1 + u_0 v_0 - (u_1 - u_0)(v_1 - v_0). \tag{2}$$

Pa je prema tome dovoljno izračunati samo tri produkta, te vrijedi

$$x \cdot y = 2^{2n}u_1v_1 + 2^n(u_1v_1 + u_0v_0 - (u_1 - u_0)(v_1 - v_0)) + u_0v_0.$$
$$x \cdot y = (2^{2n} + 2^n)u_1v_1 + 2^n(u_1 - u_0)(v_1 - v_0) + (2^n + 1)u_0v_0. \tag{3}$$

Ovaj proces možemo nastaviti rekurzivno, tj. svaki od tri produkta računamo na isti, gore opisan način.

Iako slutimo da je ovakav algoritam brži od "naivnog", te slutnje je potrebno dokazati. Označimo s $T(n)$ broj bitnih operacija potrebnih za množenje dvaju n-bitnih brojeva Karatsubinom metodom.

Pogledajmo prvo (1). Produkt $2n$-bitna broja sveli smo na četiri produkta n-bitnih brojeva, te još $O(n)$ bitnih pomaka koji predstavljaju množenje s 2^{2n} i 2^n.

Iz gornjeg razmatranja imamo:

$$T(2n) = 4T(n) + O(n) \tag{4}$$

Supstitucijom $n = \frac{n}{2}$ i prema definiciji $O(n)$, jednadžbu (4) možemo zapisati kao:

$$T(n) = 4T\left(\frac{n}{2}\right) + cn. \tag{5}$$

Proširivanjem člana s desne strane dobivamo

$$\begin{aligned} T(n) &= 4^2T\left(\frac{n}{2^2}\right) + 2cn + cn \\ &= 4^3T\left(\frac{n}{2^3}\right) + 2^2cn + 2cn + cn \\ &\vdots \\ &= 4^kT\left(\frac{n}{2^k}\right) + cn(2^{k-1} + 2^{k-2} + \ldots + 2 + 1) \\ &= 4^kT\left(\frac{n}{2^k}\right) + 2^kcn. \end{aligned}$$

Uz pretpostavku $T(1) = 1$, te ako stavimo $2^k = n$ ili $k = \log_2 n$ imamo:

$$T(n) = n^2 + cn^2 = O(n^2), \tag{6}$$

jer je $4^k = 2^k \cdot 2^k = n^2$.

Dakle, vidimo da nam sama podjela na manje potprobleme nije dovoljna. Promotrimo slučaj gdje računanje dva produkta zamjenimo s (2), odnosno slučaj (3). Ovdje računamo tri produkta $n-$bitnih brojeva, dok su ostale operacije zbrajanje, oduzimanje i bitno pomicanje.

Prema tome, vremenska složenost je:

$$T(2n) = 3T(n) + O(n) \tag{7}$$

Ako gornju jednadžbu transformiramo kao i jednadžbu (5) dobivamo:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n). \tag{8}$$

Relaciju (8) dokazat ćemo direktnom primjenom tzv. Master teorema, dokaz možemo pronaći u [3]

Teorem 4.2. *(Master teorem za podijeli i vladaj algoritme) Neka su $a \geq 1$ i $b > 0$ konstante, $f(n)$ funkcija. Rekurzivna relacija $T(n) = aT(\frac{n}{b}) + f(n)$ tada ima sljedeće ograde*

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), \text{ ako je } f(n) = O(n^{\log_b a - \varepsilon}), \text{ za neki } \varepsilon > 0, \\ \Theta(n^{\log_b a} \log_2 a), \text{ ako je } f(n) = \Theta(n^{\log_b a}), \\ \Theta(f(n)), \text{ ako je } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ za neki } \varepsilon > 0 \text{ i ako je } af(\frac{n}{b}) \leq cf(n) \text{ za} \\ \text{neku konstantu } c < 1 \text{ i za sve dovoljno velike } n. \end{cases}$$

Odmah vidimo da su ispunjeni uvjeti teorema i da je $a = 3$ i $b = 2$. Prema tome je:

$$T(n) = \Theta(n^{log_2 3}) \Rightarrow T(n) = O(n^{log_2 3}) \approx O(n^{1.59}).$$

4.3 Modularno množenje i potenciranje

U većini kriptosustava s javnim ključem, šifriranje i dešifriranje je opisano operacijama u $\mathbb{Z}_m$. Prisjetimo se, skup $\mathbb{Z}_m$ je skup $\{0, 1, ..., m-1\}$, snabdjeven dvjema operacijama $+_m$ i $\cdot_m$ koje zadovoljavaju određena svojstva čini prsten. Zbrajanje i množenje činimo isto kao i u $\mathbb{Z}$, osim što rezultat operacije reduciramo modulo m.

Definicija 4.1. *Pretpostavimo da su a i b cijeli brojevi i da je m pozitivan cijeli broj. Tada pišemo $a \equiv b \pmod m$ ako m dijeli razliku $a - b$. Izraz $a \equiv b \pmod m$ zovemo kongruencija, i čitamo "a je kongruentno b modulo m." Također, cijeli broj m zovemo modul.*

Za x i $y \in \mathbb{Z}_m$ je:

$$x +_m y = \begin{cases} x + y, & \text{ako je } x + y < m, \\ x + y - m, & \text{ako je } x + y \geq m. \end{cases}$$

Dakle vidimo da je zbrajanje modulo m jednostavno, odnosno ne razlikuje se od "običnog" zbrajanja u kontekstu vremenske složenosti.

S druge starne, množenje modulo m je znatno složenije. Ono je složenije u tome, što za razliku od zbrajanja, modularna redukcija nije tako trivijalna. Ukoliko bi produkt računali grubom silom,

prvo bismo izračunali $x \cdot y$, a potom i ostatak r pri dijeljenju $x \cdot y$ s m. Tada je $x \cdot_m y = r$.
Razvijeno je nekoliko poboljšanja ove metode, jedna od njih je i tzv. *Montgomeryjeva redukcija* (iz 1985. godine) čija je glavna ideja izbjegavanje modularne redukcije (dijeljenja).

Neka su R i T prirodni brojevi takvi da je $R > m$, $(m, R) = 1$ i $0 \leq T \leq mR$. Ako je m prikazan u bazi b i u tom prikazu ima n znamenaka, onda se za R obično uzima $R = b^n$. Pokazat ćemo jedan rezultat koji nam govori da se $TR^{-1} \bmod m$ može izračunati bez klasičnog dijeljenja. To jest, dijeljenje s m se zamjenjuje djeljenjem s R, koje zbog posebnog oblika od R predstavlja pomak za n znamenaka.

Lema 4.1. *Neka je* $m' = -m^{-1} \pmod R$, *te* $U = Tm' \bmod R$. *Tada je* $V = (T + Um)/R$ *cijeli broj i* $V \equiv TR^{-1} \pmod m$. *Nadalje,* $TR^{-1} \pmod m = V$ *ili* $TR^{-1} \pmod m = V - m$.

Dokaz. Iz definicije brojeva m' i U slijedi da postoje $k, l \in \mathbb{Z}_m$ takvi da je $mm' = -1 + kR$ i $U = Tm' + lR$. Sada je

$$\frac{T + Um}{R} = \frac{T + Tmm' + lRm}{R} = \frac{T + T(-1 + kR) + lRm}{R} = kT + lm \in \mathbb{Z}.$$

Nadalje je $V \equiv (T + Um)R^{-1} \equiv TR^{-1} + UmR^{-1} \equiv TR^{-1} \pmod{}$m. Kako je prema pretpostavci $T < mR$ i $U < R$ slijedi $0 \leq V < (mR + mR)/R = 2m$, pa iz $V \equiv TR^{-1} \pmod m$ slijedi $V - TR^{-1} \pmod m = 0$ ili m. □

Izraz TR^{-1} zove se *Montgomeryjeva redukcija* od T modulo m u odnosu na R, dok se $xR \bmod M$ naziva *Montgomeryjev prikaz* od x. Također definiramo i *Montgomeryjev produkt* brojeva x i y kao broj $Mont(x, y) = xyR^{-1} \bmod m$. Taj produkt je dobro definiran jer je $xy < m^2 < mR$. Vrijedi:

$$Mont(xR \bmod m, yR \bmod m) = (xR)(yR)R^{-1} = xyR \bmod m.$$

Dakle, za brojeve u Montgomeryjevom prikazu, modularno množenje se može provesti bez modularne redukcije. Vidi se da pri računanju Montgomeryjevog prikaza koristimo modularnu redukciju. No, ukoliko isti broj više puta množimo, tj. potenciramo, Montgomeryjeva redukcija je znatno efikasnija od običnog modularnog potenciranja.

Još jedan način za pojednostavljivanje modularne redukcije je izbor modula pogodnog oblika. Tu ponovno koristimo činjenicu da je dijeljenje brojevima oblika b^n jednostavno. Zato, ukoliko je to moguće, module biramo u obliku $m = b^n - a$, gdje je a mali prirodan broj. Tada $x \pmod m$ možemo računati na sljedeći način:

$q_0 = \lfloor x/b^n \rfloor;\ r_0 = x - q_0 b^n;\ r = r_0;\ i = 0$
while $(q_i > 0)$ {
    $q_{i+1} = \lfloor q_i a/b^n \rfloor;\ r_{i+1} = q_i a - q_{i+1} b^n$
    $i = i + 1;\ r = r + r_i$ }
while $(r \geq p)$ r = r − p

Kao što smo naveli u uvodu, najpoznatiji kriptosustavi s javnim ključem koriste modularno potenciranje prilikom šifriranja, tj. funkcije šifriranja su oblika $x^n \pmod m$.

Modularno potenciranje predstavlja specijalan slučaj potenciranja. Ukoliko potenciranju pristupimo na trivijalan način, tj. računamo x^n kao $x \cdot x \cdots x$, vidimo da imamo $O(n)$ množenja i dodatnu redukciju, pa prema tome ovaj pristup nije efikasan.

Najstarija efikasnija metoda je *binarna metoda* ili *metoda uzastopnog kvadriranja* koja koristi binarni zapis broja n da bi potenciranje svela na niz kvadriranja i zbrajanja.

Primjer 3. *Recimo da želimo izračunati* 3^{218} (*mod* 1000). *U prvom koraku eksponent zapišemo kao binaran broj,*

$$218 = 2^7 + 2^6 + 2^4 + 2^3 + 2 = 11011010_2.$$

Tada 3^{218} *postaje*

$$3^{218} = 3^{2+2^3+2^4+2^6+2^7} = 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7}.$$

Vidimo da gornji zapis odgovara čitanju binarnog broja s desna na lijevo. Isto tako možemo čitati s lijeva na desno, pri čemu dobivamo

$$3^{218} = (3 \cdot 3^2)^{2^6} \cdot (3 \cdot 3^2)^{2^3} \cdot 3^2.$$

Primijetimo da je relativno lako izračunati niz vrijednosti

$$3, 3^2, 3^{2^2}, 3^{2^3}, ...$$

zbog toga što je svaki element niza kvadrat prethodnog, pa se operacije svedu na množenje. Nadalje, kako trebamo rezultat modulo 1000, ni u kojem koraku ne moramo spremati više od 3 znamenke.

Dakle, imamo dvije rutine za računanje x^n, gdje je $n = (n_d, ..., n_0)_2$.

Binarna metoda (s desna na lijevo):

```
z = 1; y = x
for (0 ≤ i ≤ d − 1) {
    if (n_i = 1) then z = z · y, y = y^2 }
z = z · y
return z
```

Binarna metoda (s lijeva na desno):

```
z = x
for (d − 1 ≥ i ≥ 0) {
    z = z^2
    if (n_i = 1) then z = x · z }
return z
```

Oba algoritma imaju isti broj operacija: d kvadriranja, te množenja onoliko koliko ima jedinica u binarnom zapisu od n. Znamo da je broj znamenaka u binarnom zapisu broja reda veličine $O(\ln n)$ i ako za složenost množenja i dijeljenja brojeva manjih od m uzmemo da je $O(\ln^2 m)$ dobivamo da je ukupna složenost $O(\ln n \ln^2 m)$.
Jedina prednost drugog algoritma (s lijeva na desno) je da u pretposljednjem koraku algoritma uvijek množimo istim brojem x koji je najčešće mali prost broj, pa je u tom slučaju ta operacija brza. Prednost drugog algoritma je veća ukoliko broj n u svom zapisu sadrži puno jedinica. Ukoliko su ili baza x ili eksponent n fiksni moguća su dodatna poboljšanja algoritama.

U slučaju da je fiksna baza x, unaprijed izračunamo $x_i = x^{2^i}$, pa onda x^n računamo kao $x^n = \prod_{i=0}^{d} x_i^{n_i}$. Slično se dobije ako umjesto baze 2 koristimo proizvoljnu bazu b. Vidimo da u ovom slučaju samo jednom računamo x_i, čime ostvarujemo značajnu uštedu.

U slučaju fiksnog eksponenta, koristimo tzv. lance zbrojeva. To je niz $u_0, u_1, ..., u_s$ s pridruženim nizom $w_1, w_2, ..., w_s$ parova $w_i = (i_1, i_2)$ koji zadovoljavaju

$$u_0 = 1, u_s = u, u_i = u_{i_1} + u_{i_2},\ za\ 1 \le i \le s.$$

Npr. za $n = 31$, $v = (1, 2, 3, 6, 12, 24, 30, 31)$ je lanac zbrojeva duljine 7, jer je

$$\begin{aligned} 2 &= 1 + 1 \\ 3 &= 2 + 1 \\ 6 &= 3 + 3 \\ 12 &= 6 + 6 \\ 24 &= 12 + 12 \\ 30 &= 24 + 6 \\ 31 &= 30 + 1. \end{aligned}$$

Ukoliko je poznat lanac zbrojeva za n duljine s, onda se x^n može izračunati uz s množenja: $x_0 = x, x_i = x_{i_1} \cdot x_{i_2}$, za $i = 1, ..., s$ pa je $x_s = x^n$. Dakle vidimo da nam imamo s množenja, gdje je s duljina lanca zbrojeva.

Primjer 4. *Izračunajmo* 5^{31}. *Kako nam je poznat lanac zbrojeva od 31, prema gore navedenom* 5^{31} *možemo izračunati kao:*

$$5^2 = 5^1 \cdot 5^1$$

$$5^3 = 5^2 \cdot 5^1$$

$$5^6 = 5^3 \cdot 5^3$$

$$5^{12} = 5^6 \cdot 5^6$$

$$5^{24} = 5^{12} \cdot 5^{12}$$

$$5^{30} = 5^{24} \cdot 5^6$$

$$5^{31} = 5^{30} \cdot 5^1.$$

Dakle, izračunali smo 5^{31} *koristeći 7 množenja.*

5 Euklidov algoritam

Kao što smo naveli u uvodu, pozitivan cijeli broj može biti djelitelj dvaju prirodnih brojeva a i b. Najveći pozitivni cijeli broj koji dijeli a i b, od kojih je barem jedan različit od 0, zovemo najveći zajednički djelitelj i označavamo s NZD(a,b) ili samo (a,b). Prvi od načina na koji bi mogli izračunati najveći zajednički djelitelj je faktorizacija brojeva a i b na proste faktore. Prema Osnovnom teoremu aritmetike a i b možemo zapisati u obliku $a = \prod_p p^{\alpha(p)}$ i $b = \prod_p p^{\beta(p)}$, te je onda

$$(a, b) = \prod_p p^{min(\alpha(p),\beta(p))}.$$

Međutim taj pristup nije dobar, zbog činjenice da je faktorizacija cijelih brojeva $\mathbb{NP}$ težak problem. Prema tome trebamo efikasniju metodu za računanje najvećeg zajedničkog djelitelja.

Jedna od efikasnijih procedura za računanje najvećeg zajedničkog djelitelja dvaju brojeva je Euklidov algoritam. Sljedeći teorem, koji navodimo bez dokaza, nam daje teorijski osnovu za primjenu Euklidovog algoritma. (dokaz se može pronaći u [6] str. 166)

Teorem 5.1. *Neka su a i b pozitivni cijeli brojevi, i neka je r ostatak pri djeljenju a s b, tj. $a \equiv r \pmod b$ tada je $(a, b) = (b, r)$.*

Kako je $(a, b) = (|a|, |b|)$, bez smanjenja općenitosti pretpostavimo da je $a > b \geq 0$.

Euklidov algoritam:

while $(b > 0)$ $(a, b) = (b, a \bmod b)$
return a

Kako je $a \bmod b < b$ vidimo da se u svakom koraku algoritma b smanji barem za 1, odnosno da algoritam radi konačko koraka. Zbog lakšeg analiziranja, raspišimo algoritam po koracima:

$$\begin{aligned} a &= q_1 b + r_1 \\ b &= q_2 r_1 + r_2 \\ &\vdots \\ r_{n-3} &= q_{n-1} r_{n-2} + r_{n-1} \\ r_{n-2} &= q_n r_{n-1} \end{aligned} \tag{9}$$

Neka su a i b prirodni brojevi takvi da je za računanje (a, b) Euklidovim algoritmom potrebno n koraka. Tvrdimo da je tada $a \geq F_{n+2}$ i $b \geq F_{n+1}$, gdje F_k označava k-ti Fibonaccijev broj. Fibonaccijevi brojevi tvore niz, zvan Fibonaccijev niz, tako da je svaki sljedeći član, osim prva dva, jednak sumi prethodna dva člana, odnoso $F_0 = 0$, $F_1 = 1$ i $F_n = F_{n-1} + F_{n-2}$ za $n \geq 2$. Dokažimo tvrdnju matematičkom indukcijom po n.

Baza indukcije: $n = 1$
$b \geq 1 = F_2$ i $a \geq 2 = F_3$, što je točno zbog pretpostavke na a i b,
Pretpostavka indukcije: tvrdnja vrijedi za $n - 1$ koraka
Korak indukcije: dokažimo da tvrdnja vrijedi za n

Očito je da za brojeve b i r_1 Euklidov algoritam treba $n-1$ koraka.
Prema tome, po pretpostavci indukcije je $b \geq F_{n+1}$, $r_1 \geq F_n$.
Pa je $a = q_1 b + r_1 \geq b + r_1 \geq F_{n+1} + F_n = F_{n+2}$.

Ako primjenimo Euklidov algoritam na F_{n+2} i F_{n+1} imamo

$$\begin{aligned} F_{n+2} &= F_{n+1} \cdot 1 + F_n \\ F_{n+1} &= F_n \cdot 1 + F_{n-1} \\ &\vdots \\ F_4 &= F_3 \cdot 1 + F_2 \\ F_3 &= F_2 \cdot 2 + F_0 \end{aligned}$$

Kako je $F_0 = 0$ slijedi da je $(F_{n+2}, F_{n+1}) = F_2 = 1$ pa imamo točno n koraka. Ako sada iskoristimo Binetovu formulu za Fibonaccijeve brojeve

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

dobivamo

Teorem 5.2. *Neka su $a, b \leq N$. Tada je broj koraka u Euklidovom algoritmu za računanje (a,b) manji ili jednak*

$$\left\lfloor \frac{ln(\sqrt{5}N)}{ln((1+\sqrt{5})/2)} \right\rfloor - 2 \approx 2.078\, ln\,(N) + 1.672.$$

Također, može se pokazati da je prosječan broj koraka algoritma približno jednak

$$\frac{12 ln\, 2}{\pi^2}\, ln\, N + 0.14 \approx 0.843\, ln\, N + 0.14.$$

Dakle, vidimo da je potreban broj koraka reda veličine $O(lnN)$. U svakom koraku algoritma vrši se jedno dijeljenje brojeva $\leq$ N, pa je ukupna složenost Euklidovog algoritma $O(ln^3\, N)$.

Problem koji je usko vezan problemu određivanja najvećeg zajedničkog djelitelja jest problem računanja modularnog inverza $x^{-1} \pmod{m}$, to jest broja $y \in \{1, ..., m-1\}$ takvog da je $xy \equiv 1 \pmod{m}$. Taj problem smo susreli kod RSA kriptosustava iz čega proizlazi druga primjena Euklidovog algoritma. Sljedeći teorem nam daje vezu između ova dva problema.

Teorem 5.3. *Postoje cijeli brojevi x,y takvi da je $ax + by = (a, b)$.*

Dokaz. Neka je g najmanji prirodan broj oblika $ax + by, x, y \in \mathbb{Z}$. Tvrdimo da je $g = (a, b)$. Očito je da svaki zajednički djelitelj od a i b dijeli $ax + by = g$. Prema tome $(a, b)|g$. Pretpostavimo $g \nmid a$. Tada je $a = qg + r$, $0 < r < g$. No, $r = (1 - qx)a - qyb$, pa smo dobili kontradikciju s pretpostavkom da je g minimalan, iz čega zaključujemo da je $g = (a, b)$. □

Euklidov algoritam možemo koristit i za rješavanje linearnih Diofantskih jednadžbi odlika $ax+by=(a,b)$. Tada govorimo o proširenom Euklidovom algoritmu.

Prošireni Euklidov algoritam:
$(x,y,g,u,v,w)=(1,0,a,0,1,b)$;
`while` $w>0$ {
    $q=\lfloor g/w \rfloor$;
    $(x,y,g,u,v,w)=(u,v,w,x-qu,y-qv,g-qw)$}
`return` (x , y , g)

Primjer 5. *Odredimo x,y tako da je $99x+78y=(99,78)$. U prvom koraku algoritma inicijaliziramo varijable:*
$(x,y,g,u,v,w)=(1,0,99,0,1,78)$.

q	x	y	g	u	v	w
1	*0*	*1*	*78*	*1*	*-1*	*21*
3	*1*	*-1*	*21*	*-3*	*4*	*15*
1	*-3*	*4*	*15*	*4*	*-5*	*6*
2	*4*	*-5*	*6*	*-11*	*14*	*3*
2	*-11*	*14*	*3*	*26*	*-33*	*0*

Vidimo da algoritam vraća $(-11,14,3)$ pa je $99\cdot(-11)+78\cdot 14=3$.

Uz Euklidov algoritam, za računanje najvećeg zajedničkog djelitelja vrlo često se koristi i tzv. "binarni gcd[4] algoritam". Radi tako da se umjesto dijeljenja koriste oduzimanje i dijeljenje s 2 (što u binarnoj aritmetici odgovara pomaku). Rezultat toga je da algoritam ima veći broj koraka i vrši više operacija, ali su ti koraci jednostavniji i operacije brže. Korištenje oduzimanje umjesto dijeljenja ima uporište u tome da su vrlo često kvocijenti mali prosti brojevi. Može se pokazati da je vjerojatnost da je Euklidov kvocijent jednak q:

$$P(q)=\log_2\left(1-\frac{1}{(q+1)^2-1}\right).$$

Tako imamo $P(1)\approx 0.415$, $P(2)\approx 0.170$, ... Dakle, u 41.5% slučajeva kvocijent je jednak 1. Iako binarni gcd algoritam ima istu vremensku složenost kao i obični Euklidov algoritam, Akhavi i Vallée [1] dokazali su da binarni algoritam može biti i do 60% brži. Mana binarnog gcd algoritma je što je kompleksniji za implementaciju.

Neka je $v_2(k)$ najveća potencija broja 2 koja dijeli k.

Binarni gcd algoritam:
$\beta=\min\{v_2(a),v_2(b)\}$
$a=a/2^{v_2(a)}$; $b=b/2^{v_2(b)}$
`while` $(a\neq b)$
    $(a,b)=(\min\{a,b\},|b-a|/2^{v_2(b-a)})$
`return` $2^\beta a$

[4]gcd - eng. greatest common divisor, najveći zajednički djelitelj

6 Kineski teorem o ostacima

Kineski teorem o ostacima govori nam o egzistenciji i obliku rješenja sustava linearnih kongruencija. Najranija tvrdnja ovog teorema pronađena je u knjizi "Sunzi Suanjing"[5]. Rad nije sadržavao dokaz niti potpuni algoritam. Teorem je korišten za prebrojavanje vojnika. Recimo da želimo prebrojiti grupu od približno 1000 vojnika. Prvo vojnike rasporedimo u npr. 3, 4, 5 i 7 kolona, te zabilježimo koliko je vojnika svaki put ostalo neraspoređeno. Tako dobivamo sustav od četiri kongruencije s modulima 3, 4, 5 i 7. Pitanje je imali taj sustav uopće rješenje, i koje je ono. Odgovor nam daje upravo kineski teorem o ostacima.

Teorem 6.1. *Neka su $m_1, ..., m_k$ u parovima relativno prosti prirodni brojevi, tj. $(m_i, m_j) = 1$ za $i, j = 1, ..., k$, $i \neq j$. Tada za proizvoljne cijele brojeve $x_1, ..., x_k$ postoji cijeli broj x takav da vrijedi*

$$x \equiv x_i \pmod{m_i},\ i = 1, ..., k.$$

Broj x je jedinstven modulo $M = m_1 \cdots m_k$.

Broj x iz teorema možemo pronaći na sljedeći način. Neka je $M_i = \frac{M}{m_i}$. Kako je $(M_i, m_i) = 1$, pomoću proširenog Euklidovog algoritma možemo pronaći a_i takav da je $a_i M_i \equiv 1 \pmod{m_i}$. Sada vidimo da je

$$x = \sum_{i=1}^{k} a_i M_i x_i \pmod{M} \tag{10}$$

jedinstveno rješenje sustava.
Zaista, za svaki $i = 1, 2, ..., k$ imamo

$$\begin{aligned} x &= (a_1 M_1 x_1 + a_2 M_2 x_2 + \cdots a_k M_k x_k) \pmod{m_i} \\ &= a_i M_i x_i \pmod{m_i} \\ &= x_i \pmod{m_i}, \end{aligned}$$

gdje druga jednakost slijedi iz činenice da je $M_i \equiv 0 \pmod{m_i}$, za svaki $j \neq i$, a treća jer je $a_i M_i \equiv 1 \pmod{m_i}$. Nadalje, pretpostavimo da postoje dva rješenja u i v sustava kongruencija. Tada $m_1|(u-v), m_2|(u-v), ..., m_k|(u-v)$, a kako su $m_1, ..., m_k$ relativno prosti, tada $m_1 m_2 \cdots m_k$ dijeli $u - v$ odnosno

$$u \equiv v \pmod{m_1 m_2 \cdots m_k}.$$

Dakle, rješenje je jedinstveno modulo M.
Radi lakše analize, podijelimo algoritam na tri glavne cjeline:

1. Računanje M.
2. Računanje pribrojnika u (10).
3. Računanje sume svih pribrojnika.

[5] "The Mathematical Classic of Master Sun, 3-5 stoljeće

Može se pokazati da je složenost gore opisanog algoritma $O(ln^2 M)$. (dokaz se može naći u [2]).

Jednan od razloga široke primjene kineskog teorema o ostatcima je to da računanje po velikom modulu zamjenjuje s nekoliko neovisnih računanja po puno manjim modulima, što je od velike koristi u paraleliziranju računanja.

U primjenama su često m_i fiksni, dok se x_i mijenjaju. U tom slučaju dio algoritma koji ne ovisi o x_i možemo izračunati unaprijed. Dajemo sljedeći algoritam koji koristi tu ideju:

Garnerov algoritam za CRT[6]:

for $1 \leq i \leq k-1$ {
$\mu_i = \prod_{j=1}^{i} m_j$;
$c_i = \mu_i^{-1} \ mod \ m_{i+1}$ }
$M = \mu_{k-1} m_k$

$x = x_k$
for $1 \leq i \leq k-1$ {
$y = ((x_{i+1} - x)c_i) \ mod \ m_{i+1}$;
$x = x + y\mu_i$ }
$x = x \ mod \ M$
return x

Ovaj algoritam rješava module sekvencijalno, tako da nakon $i-te$ iteracije, x zadovoljava $x \equiv x_j \ (mod \ m_j)$ za $j = 1, 2, ..., i+1$. Ovaj pristup ima smisla ako rješavamo više problema sa istim m_i, ukoliko problem rješavamo jednokratno, preporuča se induktivna uporaba kineskog teorema o ostatcima za sustav od dvije kongruencije.
Ukoliko želimo riješiti sustav

$$x \equiv x_1 \ (mod \ m_1), \ x \equiv x_2 \ (mod \ m_2),$$

tada primjenimo Euklidov algoritam na $um_1 + vm_2 = 1$, te je iz toga $x = um_1x_2 + vm_2x_1 \ mod \ m_1m_2$ rješenje sustava.

Induktivni algoritam za CRT:

$m = m_1; x = x_1$
for $2 \leq i \leq k$ {
odredi u,v t.d je $um + vm_i = 1$;
$x = umx_i + vm_ix$;
$m = mm_i$;
$x = x \ mod \ m$ }
return x

Spomenimo da je problem odluke "Za danih k parova $(x_1, m_1), ..., (x_k, m_k)$, odrediti postoji li x takav da je $x \not\equiv x_i \ (mod \ m_i)$ " **NP**-potpun.

[6]CRT - chinesse remainer theorem, kineski teorem o ostatcima

7 Verižni razlomci

Pogledajmo još jednu primjenu Euklidovog algoritma. Iz prvog koraka u (9) slijedi da je

$$\frac{a}{b} = q_1 + \frac{1}{b/r_1}.$$

U sljedećem koraku imamo

$$\frac{a}{b} = q_1 + \frac{1}{q_2 + \frac{1}{r_1/r_2}}.$$

Ponavljanjem postupka racionalni broj a/b prikazujemo u obliku

$$\frac{a}{b} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \ddots + \cfrac{1}{q_n}}}.$$

Gornji prikaz se naziva *razvoj broja a/b u jednostavni verižni razlomak*. Općenito, za $\alpha \in \mathbb{R}$ prikaz broja α u obliku

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots}},$$

gdje je $a_0 \in \mathbb{Z}$, a $a_1, a_2, ... \in \mathbb{N}$ zovemo *razvoj broja α u jednostavni verižni (neprekidni) razlomak.* Verižni razlomak kraće zapisujemo kao $[a_0; a_1, a_2, ...]$. Brojeve $\alpha_0, \alpha_1, ...$ zovemo *parcijalni kvocijenti*, a definiramo ih na sljedeći način:

$$a_0 = \lfloor \alpha \rfloor,\ \alpha = a_0 + \frac{1}{\alpha_1}, a_1 = \lfloor \alpha_1 \rfloor,\ \alpha_1 = a_1 + \frac{1}{\alpha_2},\ a_2 = \lfloor \alpha_2 \rfloor, ... \tag{11}$$

Postupak ponavljamo sve dok je $a_k \neq \alpha_k$. Može se pokazati da je razvoj u jednostavni verižni razlomak broja α konačan ako i samo ako je α racionalan broj. Prilikom računanja s iracionalnim brojevima, najčešće znamo samo njihovu racionalnu aproksimaciju, to znači da će u razvoju biti točno samo nekoliko početnih q_j.

Primjer 6. *Izračunati razvoj u jednostavni verižni razlomak broja* $\sqrt{101}$, *ali tako da u svim računima koristimo samo 10 najznačajnijih znamenki.*

$$\sqrt{101} = 10.04987562.$$

Iz (11) dobivamo:

$$a_0 = 10, \alpha_1 = \frac{1}{\alpha - a_0} = 20.04987607 \text{ pa je } a_1 = 20.$$

Postupak provodimo analogno i za $a_2, a_3, \ldots$ *te dobivamo*

$$\sqrt{101} = [10; 20, 20, 20, 8, 6, 1, \ldots].$$

Kako smo u računu koristili aproksimaciju zanima nas koliko smo prvih a_i točno izračunali. Uskoro ćemo pokazati da je

$$\sqrt{101} = [10; \overline{20}]$$

pa vidimo da smo dobro izračunali prva četiri parcijalna kvocijenta. Sljedeći algoritam za razvoj broja u verižni razlomak ima implementiran zaustavni kriterij. Neka je dan $\alpha \in \mathbb{R}$, te racionalni brojevi a/b i a'/b' takvi da je

$$\frac{a}{b} \leq \alpha \leq \frac{a'}{b'}.$$

Algoritam računa razvoj od α i staje točno onda kada više nije moguće odrediti sljedeći a_i iz a/b i a'/b', odnosno kada se pripadni a_i u razvojima od a/b i a'/b' ne podudaraju, te daje gornju i donju ogradu za taj a_i.

Razvoj u verižni razlomak:

$i = 0$
$q = \lfloor a/b \rfloor; r = a - bq; r' = a' - b'q$
while $(0 \leq r' < b' \text{ and } b \neq 0)$ {
    $a_i = q$
    $i = i + 1$
    $a = b; b = r; a' = b'; b' = r';$
    $q = \lfloor a/b \rfloor; r = a - bq; r' = a' - b'q$
if (b = 0 and b' = 0) then return $[a_0; a_1, \ldots, a_i]$
if (b ≠ 0 and b' = 0) then return $[a_0; a_1, \ldots, a_{i-1}]$, $a_i \geq q$
$q' = \lfloor a'/b' \rfloor$
if (b = 0 and b' ≠ 0) then return $[a_0; a_1, \ldots, a_{i-1}]$, $a_i \geq q'$
if (bb' ≠ 0) then return $[a_0; a_1, \ldots, a_{i-1}]$, $\min(q, q') \leq a_i \leq \max(q, q')$}

Ako je α kvadratna iracionalnost, tj. iracionalan broj koji je rješenje neke kvadratne jednadžbe s racionalnim koeficijentima, tada je njegov razvoj u jednostavni verižni razlomak periodičan. Kako

je α kvadratna iracionalnost, može se prikazati u obliku

$$\alpha = \alpha_0 = \frac{s_0 + \sqrt{d}}{t_0},$$

pri čemu su $d, s_0, t_0 \in \mathbf{Z}, t_0 \neq 0$, d nije potpun kvadrat i $t_0|(d - s_0^2)$. Sada parcijalne kvocijente a_i računamo rekurzivno:

$$a_i = \lfloor \alpha_i \rfloor, s_{i+1} = a_i t_i + s_i, t_{i+1} = \frac{d - s_{i+1}^2}{t_i}, \alpha_{i+1} = \frac{s_{i+1} + a_0}{t_{i+1}}. \tag{12}$$

Primijetimo da iako je α iracionalan broj, algoritam radi samo s cijelim brojevima. Može se pokazati da su s_i i t_i ograničeni, pa prema tome mogu poprimiti samo konačno mnogo vrijednosti. To znači da postoje indeksi j i k, $j < k$, takvi da je $s : j = s_k$ i $t_j = t_k$, ali tada je i $\alpha_j = \alpha_l$ pa je

$$\alpha = [a_0, a_1, ..., a_{j-1}, \overline{a_j, a_{j+1}, ..., a_{k-1}}],$$

gdje povlaka označava niz brojeva koji se periodički ponavljanja. U slučaju da je $\alpha = \sqrt{d}$ može se i preciznije dati kako izgleda razvoj u verižni razlomak.

$$\sqrt{d} = [a_0, \overline{a_1, a_2, ..., a_{r-1}, 2a_0}],$$

gdje je $a_0 = \lfloor \sqrt{d} \rfloor$, a za $a_1, a_2, ..., a_{r-1}$ vrijedi $a_i = a_{r-i},\ i = 1, 2, ..., r-1$. Može se pokazati da za duljinu perioda vrijedi $r = O(\sqrt{d} \log d)$.
Racionalne brojeve oblika

$$\frac{p_k}{q_k} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots + \cfrac{1}{a_k}}},$$

zovemo *konvergente verižnog razlomka*. Vidimo da brojnici i nazivnici konvergenti zadovoljavaju sljedeće rekurzije:

$$\begin{aligned} p_0 = a_0, p_1 = a_0 a_1 + 1, p_{k+2} = a_{k+2} p_{k+1} + p_k, \\ q_0 = 1, q_1 = a_1, q_{k+2} = a_{k+2} q_{k+1} + q_k. \end{aligned} \tag{13}$$

Pokažimo indukcijom da vrijedi:

$$q_k p_{k-1} - p_k q_{k-1} = (-1)^k. \tag{14}$$

Baza indukcije: k = 1

Prema (13) vrijedi $q_1p_0 - p_1q_0 = a_1a_0 - (a_0a_1 + 1) \cdot 1 = -1 = (-1)^1$

Pretpostavka indukcije: tvrdnja vrijedi za k koraka

Korak indukcije: pokažimo da tvrdnja vrijedi za k+1

$q_{k+1}p_k - p_{k+1}q_k = (a_{k+1}q_k + q_{k-1})p_k - (a_{k+1}p_k + p_{k-1})q_k$
$= q_{k-1}p_k - p_{k-1}q_k = -(q_kp_{k-1} - p_kq_{k-1}) = -(-1)^k = (-1)^{k+1})$.

Gdje treća jednakost od kraja slijedi iz pretpostavke indukcije.

Jednakost (13) povlači da je $\frac{p_{2k}}{q_{2k}} \leq \alpha$ i $\alpha \leq \frac{p_{2k+1}}{q_{2k+1}}$ za svaki k. Također, ako je α iracionalan broj vrijedi $\lim_{k\to\infty} \frac{p_k}{q_k} = \alpha$. Konvergente verižnog razlomka dobro aproksimiraju α. Može se pokazati sljedeća ocjena

$$\left|\alpha - \frac{p_k}{q_k}\right| < \frac{1}{q_k^2}.$$

Vrijedi i obrat gornje tvrdnje. To jest, ako je $\frac{p}{q}$ racionalan broj koji zadovoljava nejednakost

$$\left|\alpha - \frac{p}{q}\right| < \frac{1}{2q^2},$$

tada je $\frac{p}{q} = \frac{p_k}{q_k}$ za neki k. Verižne razlomke i konvergente koristimo kao alat za rješavanje nekih diofantskih jednadžbi. Posebno se ističu linearne diofantske jednadžbe, kao i *Pellove jednadžbe. Pellove jednadžbe* su jednadžbe oblika $x^2 - dy^2 = 1$, pri čemu je d prirodan broj koji nije potpun kvadrat. Vrlo često promatramo i jednadžbu oblika $x^2 - dy^2 = -1$.

Pokazat ćemo još jednu primjenu verižnih razlomaka. Prema Fermatovom teoremu o zbroju kvadrata, neparan prost broj p može se prikazati kao zbroj kvadrata dva cijela broja ako i samo ako je $p \equiv 1 \pmod 4$. Dakle za dani prost broj p vrlo je jednostavno provjeriti može li se on zapisati kao suma kvadrata dva cijela broja. Međutim, problem nalaženja brojeva x, y takvih da je $x^2 + y^2 = p$ je vrlo težak. Konstruirat ćemo dva algoritma koristeći verižne razlomke.

1. konstrukcija (Hermite[7])

Pretpostavimo da znamo neko rješenje kongruencije $z^2 \equiv -1 \pmod p$. Prema tome, $z^2 + 1$ je neki višekratnik od p kojeg znamo prikazati kao zbroj dva kvadrata. Ideja je da pomoću njega broj p također prikažemo kao zbroj dva kvadrata. Pogledajmo verižni razlomak

$$\frac{z}{p} = [a_0, a_1, ..., a_m].$$

Postoji jedinstveni cijeli broj n takav da je $q_n < \sqrt{p} < q_{n+1}$. Prema (13) vrijedi

$$\frac{p_n}{q_n} < \frac{z}{p} < \frac{p_{n+1}}{q_{n+1}}$$

[7]Charles Hermite, francuski matematičar

što povlači

$$\left|\frac{z}{p}-\frac{p_n}{q_n}\right|<\left|\frac{p_n}{q_n}-\frac{p_{n+1}}{q_{n+1}}\right|=\frac{1}{q_nq_{n+1}}.$$

Dakle, $\frac{z}{p}=\frac{p_n}{q_n}+\frac{\varepsilon}{q_nq_{n+1}}$, gdje je $|\varepsilon|<1$. Odavde je $zq_n-pp_n=\frac{\varepsilon p}{q_{n+1}}$, pa je $(zq_n-pp_n)^2<\frac{p^2}{q_{n+1}^2}$.

Prema tome, $(zq_n-pp_n)^2+q_n^2\equiv q_n^2(z^2+1)\equiv 0 \pmod p$ i $0<(zq_n-pp_n)^2+q_n^2<2p$, što daje $(zq_n-pp_n)^2+q_n^2=p$.

2. konstruncija (Legendre[8])

Promotrimo Pellovu jednadžbu $x^2-py^2=1$. Može se pokazati da ona ima beskonačno mnogo rješenja u prirodnim brojevima. Neka je (X,Y) najmanje takvo rješenje. Iz $(X+1)(X-1)=pY^2$ slijedi da je $X+1=ab^2p$, $X+1=ab^2$ ili $X-1=ac^2$, $X-1=ac^2p$, gdje je $a=(X+1,X-1)=1$ ili 2, dok su b,c meki prirodni brojevi. Oduzimanjem jednadžbi dobivamo $c^2-pb^2=-\frac{2}{a}$ ili $b^2-pc^2=\frac{2}{a}$. Kako smo pretpostavili da je (X,Y) minimalno rješenje, to povlači da je $a\neq 2$ u drugoj jednadžbi. Prema tome je

$$c^2-pb^2=-1,\ c^2-pb^2=-2 \text{ ili } b^2-pc^2=2.$$

Ako je $p\equiv 1 \pmod 4$, onda je $c^2-pb^2\equiv 0,1$ ili $3 \pmod 4$, pa mora vrijediti $c^2-pb^2=-1$. Nužan i dovoljan uvjet da bi Pellova jednadžba ovog oblika imala rješenja je da period u razvoju u verižni razlomak od $\sqrt{p}$ bude neparan. Dakle, razvoj mora biti ovog oblika:

$$\sqrt{p}=[a_0;\overline{a_1,...,a_n,a_n,...,a_1,2a_0}].$$

Broj $\alpha_{n+1}=\frac{s_{n+1}+\sqrt{p}}{t_{n+1}}$ je čisto periodičan (nema pretperioda) i period mu je palindrom. Neka je $\alpha'_{n+1}=\frac{s_{n+1}-\sqrt{p}}{t_{n+1}}$ njegov konjugat. Može se pokazati da je razvoj od $-\frac{1}{\alpha'_{n+1}}$ također čisto periodičan, s tim da se kvocijenti u periodu ponavljaju obrnutim redosljedom nego oni u α_{n+1}. Kako je period od α_{n+1} palindrom, zaključujemo da je $-\frac{1}{\alpha'_{n+1}}=\alpha_{n+1}$. Prema tome je

$$\alpha'_{n+1}\alpha_{n+1}=\frac{s_{n+1}^2+p}{t_{n+1}^2}=-1,$$

iz čega konačno slijedi $p=s_{n+1}^2+t_{n+1}^2$.

[8] Adrien-Marie Legendre, francuski matematičar

8 Testiranje prostosti

Vidjeli smo da je izbor parametara iznimno važan za sigurnost RSA kriptosustava. Prvi zadatak je odabrati dva velika prosta broja p i q, od na primjer 100 znamenki. U praksi to činimo tako da odaberemo neki broj m koji ima isto 100 znamenki, pa pomoću nekog testa prostosti tražimo prvi prost broj veći od m. Detalje o izboru parametara pronaći u [6]

Najjednostavniji test prostosti sastoji se od niza probnih dijeljenja zadanog broja svim brojevima manjim od njega samog. Primijetimo da je moguće poboljšati spomenuti pristup. Naime, dijeljenje je dovoljno provesti za brojeve do $\sqrt{n}$, zbog toga što ako je $n = ab$ ne mogu istovremeno oba faktora biti veća od $\sqrt{n}$.

Školski algoritam za testiranje prostosti:
for $i \leq \lceil \sqrt{n} \rceil$ {
    if n mod i = 0
        return False
    else
        return True }

Vidimo da gornji algoritam radi $O(\sqrt{n})$ dijeljenja, i znamo da je duljina ulaza $O(ln\ n)$. Vrijedi da je $O(\sqrt{n}) = O((e^{ln\ n})^{\frac{1}{2}})$ što je eksponencijalno veće od broja bitova potrebnih za prikaz broja. Dakle, algoritam je eksponencijalne složenosti, odnosno nije dovoljno efikasan za primjenu na većim brojevima. Jedno rješenje ovog problema je razvoj brzih i jednostavnih algoritama, pri čemu žrtvujemo nešto točnosti. Takvi algoritmi imaju pozitivnu vjerojatnost pogreške, koja se može proizvoljno smanjiti. Valja napomenuti da ti algoritmi griješe tako da složene brojeve proglase prostim, a nikad obrnuto. Nekad nam i informacija o tom je li broj složen dovoljna. Tada govorimo o testiranju složenosti.

8.1 Fermatov test

Sljedeći teorem daje teorijsku osnovu nekih testova prostosti i složenosti.

Teorem 8.1. ***(Mali Fermatov teorem).*** *Ako je p prost i a relativno prost s p, tada vrijedi $a^{p-1} \equiv 1\ (mod\ p)$.*

Odmah je vidljivo da se *Teorem* 8.1 (točnije, njegov obrat) može koristiti kao test složenosti. Za broj n tražimo s njim relativno prost broj a tako da je $a^{n-1} \not\equiv 1 \pmod{n}$). Ukoliko nađemo takav a, dokazali smo da je n složen. No, možemo li taj teorem iskoristiti kao test prostosti? Odgovor nažalost nije potvrdan. Naime, postoje parovi brojeva a i n koji zadovoljavaju kriterij iz teorema, a da je pri tome n složen.

Definicija 8.1. *Za neparan složen broj n kažemo da je (Fermat)* ***pseudoprost u bazu a*** *ako vrijedi*

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

Primjer 7. *Broj 33 je pseudoprost za bazu 10. Zaista,* $33 = 3 \cdot 11$, *pa je 33 složen, ali vrijedi* $10^{32} = (10^2)^{16} \equiv 1^{16} \equiv 1 \pmod{3)3}$.

Dakle, ako broj prođe Fermatov test za neku bazu, ne možemo zaključiti da je prost. Štoviše, pokazano je da pseudoprostih brojeva za fiksnu bazu ima beskonačno mnogo. Međutim, ukoliko provodimo Fermatov test za različite baze, možemo smanjiti vjerojatnost da broj bude lažno prost.

Npr., broj iz prošlog primjera je pseudoprost za bazu 10, ali nije za bazu 2: $2^{3}2 = 2^{2} \cdot (2^{5})^{4} \equiv 2^{2} \cdot 1 \equiv 4 \pmod{33}$, pa saznajemo da je 33 složen.

Preostaje odrediti složenost Fermatovog testa. Kao što vidimo, trebamo izračunati $a^{d} \pmod{n}$ što je efikasnije moguće. Dakle, to je problem modularnom potenciranja, koji smo obradili u poglavlju 5.3., iz čega slijedi da je to polinomijalan algoritam.

9 Dodatak

Dajemo implementaciju nekih algoritama i jednostavnu implementaciju RSA kriptosustav baziranu na shemi iz uvoda. Pretpostavljamo da znamo velike proste brojeve p i q.

```
import random
from math import log


# funkcija za zbrajanje dva cijela broja u bazi b
# po defaultu je b = 10
# pretpostavljamo da je ulaz ispravno zadan,
# tj. da su sve znamenke manje od d
def zbroj(x, y, b = 10):
    # brojeve zapisujemo u obliku (x_n, ..., x_1, x_0)
    x = [int(d) for d in str(x)]
    y = [int(d) for d in str(y)]
    n = len(x)
    c = 0
    w = []
    # for petlja prolazi listom od desna na lijevo, te rezultat dodaje
    # na pocetak od w
    for i in range(n-1, -1, -1):
        if (x[i] + y[i] + c < b):
            w = [x[i] + y[i] + c] + w
            c = 0
        else:
            w = [x[i] + y[i] + c - b] + w
            c = 1
    if c != 0:
        w = [c] + w
    return w;

# funkcija za umnozak dva cijela broja u bazi b
# dozvoljavamo da faktori budu razlicite duljine

def produkt(x, y, b = 10):
    x = [int(d) for d in str(x)]
    y = [int(d) for d in str(y)]
    x.reverse()
    y.reverse()
    n = len(x)
    m = len(y)
    w = [0]*(n + m)
    for i in range(m):
        c = 0
        for j in range(n):
            pom = w[i+j] + x[j]*y[i] + c
```

```
            w[i+j] = pom%b
            c = pom//b
        w[i+n] += c
    w.reverse()
    w = int("".join(map(str, w)))
    return w


def karatsuba(x,y):

        if len(str(x)) == 1 or len(str(y)) == 1:
                return x*y
        else:
                n = max(len(str(x)),len(str(y)))
                nb = n // 2

                a = x // pow(10,nb)
                b = x % pow(10,nb)
                c = y // pow(10,nb)
                d = y % pow(10,nb)
                ac = karatsuba(a,c)
                bd = karatsuba(b,d)
                ad_plus_bc = karatsuba(a+b,c+d) - ac - bd

                # pisanje n kao 2*nb rjesava parne i neparne slucajeve
                prod = ac * 10**(2*nb) + (ad_plus_bc * 10**nb) + bd

                return prod

    def euklid(a,b):
        while(b > 0):
                (a,b) = (b, a%b)
        return a

    def prosireni_euklid(a,b):
        (x,y,g,u,v,w)  = (1,0,a,0,1,b)
        while( w > 0):
                q = int(g/w)
                (x,y,g,u,v,w) = (u,v,w,x-q*u, y - q*v, g - q*w)
        return (x,y)

    def generiraj_kljuceve(p,q):

        n = p*q
            phi = (p-1)*(q-1)
        #generiramo cijeli broj tako da je (phi, e) = 1
        c = random.randrange(1, phi)
```

```
    #koristenjem Euklidovog algoritma provjeravamo da li je to istina
    t = euklid(e, phi)
    while(t != 1):
            e = random.randrange(1, phi)
            t = t = euklid(e, phi)

    #koristenjem prosirenog euklidovog algoritma generiramo privatan
    d = prosireni_euklid(e, phi)[0]
    if (d < 0): d += phi
    return ((e,n), (d,n))

def encrypt(pk, tekst):
    #otpakiramo kljuc po komponentama
    k, n = pk
    #pretvaramo svako slovo otvorenog teksta u odgovarajuci broj
    sifrat = [pow(ord(znak),k,n) for znak in tekst]

    return sifrat


def decrypt(pk, sifrat):
    #otpakiramo kljuc po komponentama
    k, n = pk
    #pretvaramo svaki broj u slovo pomocu funkcije desifriranja
    otvoreni_tekst = [chr(pow(znak,k,n)) for znak in sifrat]

    return ''.join(otvoreni_tekst)
return tekst


#skripta za pokretanje algoritama
import time
from algoritmi import *

if __name__ == '__main__':
    start_i = time.time()
    p = 10263959282974110577205419657399167590071656780803806680334l9
    q = 10660348838016845482092722036001287867920795857598929152227O6
    (jk, pk) = generiraj_kljuceve(p,q)
    print("Javni kljuc:",jk, "Tajni kljuc",pk)
    poruka = "osnovni algoritmi teorije brojeve"
    print("Otvoreni tekst: ",poruka)
    start = time.time()
    sifrat = encrypt(pk, poruka)
    print("Vrijeme potrebno za sifriranje",time.time() - start)
        #print("Sifrat: ",sifrat)
    start = time.time()
```

```
    otvoreni_tekst = decrypt(jk, sifrat)
    print("Vrijeme potrebno za desifriranje: ",time.time() - start)
    print("Otvoreni tekst: ",otvoreni_tekst)
    print("Ukupno vrijeme trajanja je: ", time.time() - start_i)

Javni kljuc: (544196229423526756558951073880386627817481399638660635
Otvoreni tekst:  osnovni algoritmi teorije brojeve
Vrijeme potrebno za sifriranje 0.14040017127990723
Vrijeme potrebno za desifriranje:  0.15600037574768066
Otvoreni tekst:  osnovni algoritmi teorije brojeve
Ukupno vrijeme trajanja je:  0.35880064964294434
255.8260506587214
```

10 Literatura

[1] Akhavi, Ali, Vallée, Brigitte, *AverageBit-Complexity of Euclidean Algorithms*, Proceedings ICALP'00, Lecture Notes

[2] M. W. Baldoni, C. Ciliberto, G. M. Piacentini Cattaneo, *Elementary Number Theory, Cryptography and Codes* Springer-Verlag, Berlin Heidelberg, 2009.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *Introduction to Algorithms, 3rd Edition* The MIT Press, Chambridge, Massachusetts, 2009.

[4] A. Dujella *Teorija brojeva u kriptografiji*, 2003.

[5] J. Hoffstein, J. Pipher, J. H. Silverman *An Introduction to Mathematical Cryptography* Springer Science+Business Media, LLC, New York, 2008.

[6] T. Koshy *Elementary Number Theory with Applications* Elsevier Inc., London, 2007

[7] A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing 7 (1971), 281–292 str.

[8] D. R. Stinson *Cryptography Theory and Practice* Chapman & Hall/CRC, Boca Raton, Fl, 2006.

[9] https://www.untruth.org/ josh/math/effective-crt.pdf

Sažetak

U radu ćemo se baviti osnovnim algoritmima koji se koriste u teoriji brojeva i kriptografiji. Uvod sadrži motivaciju za razvoj kriptografije, a samim time i algoritama. Također definiramo kriptosustav, te ostale pojmove koje ćemo koristiti u radu. Središnji dio rada bavi se analizom složenosti osnovnih algoritama. Posljednji dio rada sadrži kratak primjer, to jest impelmentaciju RSA sheme.

Ključne riječi

Algoritam, vremenska i prostorna složenost algoritma, Euklidov algoritam, teorija brojeva

Abstract

In this paper we will describe basic algorithms that are used in number theory and criptography. Introduction contains motivation for development of cryptography, thus and algorithms. Also, we define crypto sistem, and other concepts that are used in paper. Central part is about complexity of basic algorithms. Last part contains short example, that is implementation if RSA scheme.

Key words

Algorithms, time and space complexity of an algorithm, Euclidian algorithm, number theory

11 Životopis

Rođen sam 1. srpnja 1993. u Osijeku. Završio sam Osnovnu školu Josipovac u Josipovcu a zatim i III. gimnaziju Osijek. Tijekom osnovnoškolskog i srednjoškolskog obrazovanja sudjelovao sam na brojnim natjecanjima iz matematike i fizike. Nakon završetka gimnazije, 2012. godine upisujem Sveučilišni preddiplomski studij matematike na Odjelu za matematiku u Osijeku. Preddiplomski studij završavam 2017. godine s temom završnog rada Riemann-Stieltjesov integral pod mentorstvom izv. prof. dr. sc. Mihaele Ribičić Penave. Iste godine upisujem Sveučilišni diplomski studij, smjer Matematika i računarstvo.