UNIVERZITET CRNE GORE

Prirodno-matematički fakultet Podgorica

Marko Pejović

Algoritmi za množenje i dijeljenje velikih brojeva

SPECIJALISTIČKI RAD

Podgorica, 2018.

UNIVERZITET CRNE GORE

Prirodno-matematički fakultet Podgorica

Algoritmi za množenje i dijeljenje velikih brojeva

SPECIJALISTIČKI RAD

Kriptografija

Mentor: Vladimir Božović

Marko Pejović

Studijski program: Matematika i računarske nauke

Podgorica, Jul 2018.

Posveta

Zahvaljujem se porodici, prijateljima i mentoru

Apstrakt

Množenje i dijeljenje su neke od osnovnih matematičkih operacija koje smo naučili još u osnovnoj školi. Pomoću njih možemo relativno lako pomnožiti i podijeliti dva trocifrena, četvorocifrena broja. Međutim, šta kad imamo recimo dva osmocifrena broja? Koliko će nam biti potrebno vremena da dođemo do rešenja? U tu svrhu su matematičari razvili algoritme pomoću kojih, ne samo da možemo pomnožiti dva osmocifrena broja, nego i brojeve sa mnogo više cifara. U nastavku ćemo se osvrnuti na najvažnije algoritme za množenje i dijeljenje velikih brojeva.

Abstract

Multiplication and division are some of the basic mathematical operations we've learned in elementary school. With them we can relatively easily multiply and split two three-digit, four-numbered numbers. However, what if we have, say, two eight-digit numbers? To this purpose, mathematicians have developed algorithms with which, not only can we multiply two eight-digit numbers, but also numbers with much more digits. Below we will look at the most important algorithms for multiplying and dividing large numbers.

Sadržaj

Glava 1

Uvod

Brza realizacija osnovnih aritmetičkih operacija za velike prirodne brojeve ima velike primjene u računarskoj algebri i kriptografiji. Velike prirodne brojeve možemo prikazati nizom brojeva u nekoj izabranoj bazi, tako da se osnovne operacije nad ciframa mogu egzaktno i brzo izvesti u aritmetici računara. Algoritmi za sabiranje i oduzimanje velikih brojeva su jednostavni i oponašaju ručno računanje.

Za razliku od toga, slični algoritmi za množenje i cjelobrojno dijeljenje sa ostatkom imaju kvadratnu složenost i nisu optimalni, jer postoje i mnogo brži algoritmi (poput Karatsubinog i dr.). Na samom početku, opisaćemo klasični algoritam za množenje i dijeljenje velikih brojeva. Potom ćemo se osvrnuti na algoritme za brzo množenje velikih brojeva.

Nakon toga, opisaćemo i algoritam za brzo dijeljenje velikih brojeva koji se, kako će se ispostaviti, svodi na brzo množenje.

Glava 2

Pozicioni zapis prirodnih brojeva

U pozicionim brojnim sistemima upotrebljava se ograničeni broj cifara, s tim da njihova vrijednost zavisi od položaja u zapisanom broju. Otuda su ti sistemi dobili svoj naziv. Svaki pozicioni brojni sistem ima svoju osnovu (bazu) i cifre. Dekadni zapis brojeva koristimo svakodnevno, a to je upravo takav način zapisivanja u bazi b=10. Naredna teorema, bez dokaza, pokazuje kako se zapisuje neki prirodan broj u proizvoljnoj bazi.

Teorema 2.1. *Neka je $b \in \mathbb{N}$, $b \geq 2$ bilo koji prirodan broj. Za svaki prirodan broj $u \in \mathbb{N}$ postoji broj $n \in \mathbb{N}$, i brojevi $u_0, u_1, \ldots, u_n$ takvi da je*

$$\sum_{i=0}^{n} u_i \cdot b^i = u. \quad (2.1)$$

Ovakav prikaz prirodnog broja je jedinstven.

Definicija 2.1. *Broj b zovemo baza, a broj n nazivamo najveći stepen broja u u bazi b. Označavamo sa:*

$$\deg_b(u) = n.$$

Brojeve $u_0, u_1, \ldots, u_n$ nazivamo cifre broja u u bazi b, dok se cifra u_n naziva vodećom.

Ukoliko važi $0 \leq u_i < b$ za $i = 0, 1, \ldots, n-1$ i $0 < u_n < b$ tada izraz (2.1) nazivamo pozicionim zapisom broja u u bazi b. Koristimo sledeću oznaku $u = (u_0, u_1, \ldots, u_n)_b$. Navedimo još par stvari vezanih za pozicioni zapis nekog broja u izabranoj bazi. Zanimljivo je na koji način možemo dobiti dužinu pozicionog zapisa broja.

Definicija 2.2. *Dužina broja u u bazi b, u oznaci $l_b(u)$ je broj cifara u pozicionom zapisu broja u u bazi b i računa se po formuli:*

$$l_b(u) = \deg_b(u) + 1 = [\log_b(u) + 1]$$

Kako primjećujemo, dužina broja zavisi od izabrane baze. Što je broj b veći, to je dužina broja manja i obratno.
Upravo ta dužina brojeva je bitan ulazni podatak. Zbog toga što se složenost algoritama izražava kao funkcija dužine ulaznih brojeva. Kako dužina zavisi od baze, tako zavisi i složenost. Takođe bitan podatak je i veza između dužina brojeva u različitim bazama. Shodno tome, navešćemo teoremu koja definiše taj odnos.

Teorema 2.2. *Dužine prirodnih brojeva u dvije različite baze su proporcionalne. Naime, ako su $b_1, b_2 \geq 2$, bilo koje dvije baze, tada važi relacija*

$$l_{b_2}(u) \sim \log_{b_2} b_1 \cdot l_{b_1}(u),$$

Dakle kao zaključak možemo reći da složenost aritmetičkih algoritama ne zavisi bitno od baze u kojoj su prikazani brojevi. U nastavku ćemo se upoznati sa klasičnim algoritmom za množenje brojeva.

Glava 3

Klasično množenje

3.1 Klasični algoritam množenja

U ovom poglavlju ćemo se upoznati sa klasičnim algoritmom za množenje. Najprije, analiziraćemo algoritam koji računa množenje broja u pozicionom zapisu sa brojnom konstantom. Ovaj algoritam je prilično jednostavan, ali i istovremeno koristan, jer će nam poslužiti za razvoj opšteg algoritma za množenje prirodnih brojeva.

Propozicija 3.1. *Neka je $u \in \mathbb{N}$ broj koji je predstavljen u pozicionom zapisu u proizvoljnoj bazi b, $u = (u_0, u_1, \ldots, u_n)_b$. Ukoliko važi $v_0 \in \mathbb{N}$ i $0 \leq v_0 \leq b$ tada algoritam ALG1(prikazan na slici ispod) daje niz brojeva u istoj bazi koji predstavljaju proizvod $u \cdot v_0$. Dakle,*

$$u \cdot v_0 = (w_0, w_1, \ldots, w_k)_b.$$

Pri tome je $k = \deg_b(w) = n$ ili $k = \deg_b(w) = n+1$ za $v_0 > 0$. Dok za $v_0 = 0$ algoritam vraća $k = -1$.

Za prvi ulazni parametar navodimo pozicioni zapis broja u, u nekoj prozivoljnoj bazi b. Kao drugi parametar je broj v_0.
Kao rezultat rada ovog algoritma dobija broj w u svom pozicionom zapisu u bazi b.

Dakle rezultat rada algoritma je $w = (w_0, w_1, \ldots, w_k)_b$.

```
pocetak:
if(v_0 > 0){
    alg = 0;
    for(int i=0;i<deg(u);i++){
        temp = u_i * v_0 + alg;
        w_i = temp%b;
        alg = temp / b;
    }
    if(alg > 0){
        deg(w) = deg(u) + 1;
        w_deg(w) = alg;

    }
    else{
        deg(w)=deg(u);

    }
else{
    deg(w) = -1;
}

}krak
```

Uvedimo pojam $maxInt$, koji će nam predstavljati najveći cio broj koji je moguće prikazati na računaru, u nekom izabranom tipu cijelih brojeva kojeg podržava arhitektura računara. Tada važi sledeće:

$$u_i \leq maxInt, \quad i = 1, \ldots, n$$

Kako važi da je $0 \leq u_i \leq b$, $i = 0, 1, \ldots, n$ i $0 < u_n <$ b imamo da je $b \leq maxInt + 1$. Potreban uslov da algoritam radi korektno u aritmetici računara, a ne samo u teoriji,

je

$$b^2 - b - 1 \leq maxInt.$$

U nastavku pogledajmo kolika je složenost ovog algoritma. Prostorna složenost je

$$\underset{S}{\text{Compl}}(ALG1) = 2 \cdot l(u) + c,$$

s tim da je $c \leq 6$, što se može i umanjiti. Operacije "/"i "%"brojimo svaku za sebe, zbog toga što se u višim programskim jezicima te operacije vrše odvojeno i podjednako traju. Najbolja, najgora i prosječna aritmetička složenost su jednake budući da algoritam uvijek obavlja isti broj operacija. Dakle, aritmetička složenost Algoritma1 je

$$\underset{A}{\text{Compl}}(ALG1) = 4 \cdot l(u).$$

Naravno, vremenska složenost zavisi od dužine broja u, pa važi:

$$\underset{T}{\text{Compl}}(ALG1) \sim l(u) \sim l(w).$$

U nastavku, obradićemo i slučaj množenja potencijom baze. Upravo, ovaj sledeći i algoritam koji smo prethodno odradili, u kombinaciji, omogućavaju direktnu realizaciju algoritma za množenje dva broja zapisanih u pozicionom zapisu. Kako se množenje svodi na pomjeranje cifara, možemo sa sigurnošću tvrditi da je algoritam korektan. Označimo ovaj algoritam sa ALG2. Njegova prostorna složenost je

$$\underset{S}{\text{Compl}}(ALG2) = 2 \cdot l(u) + p + c,$$

gdje je $c \leq 4$. Aritmetička složenost je konstantna zbog sabiranja stepena, dakle

$$\underset{A}{\text{Compl}}(ALG2) = 1.$$

Što se tiče vremenske složenosti, ona je zavisna od izbora strukture podataka za prikazivanje brojeva u računaru. U najgorem slučaju, vremenska složenost je

$$\underset{T}{\text{Compl}}(ALG2) \sim l(w) = l(u) + p.$$

U nastavku je prikazana šema algoritma. Dakle, za ulazne parametre imamo pozicioni zapis broja u, tj. $(u_n, u_{n-1}, \ldots, u_0)$ i potencija baze $p \in \mathbb{N}$ u nekoj odabranoj bazi b. Kao rezultat rada ovog algoritma dobijamo broj $w = u \cdot b^p$ prikazan u pozicionom zapisu u bazi b.

```
pocetak:
  if(0<=deg(u)){
    
    deg(w)=deg(u) + p;
    for(int i=deg(u);i=>0;i--){
      
      w_i+p = u_i;
      
    }
    
    for(i=0;i<p;i++){
      w_i = 0;
      
    }
  else{
      deg(w) = -1;
    }
  
kraj
```

ALG1 i ALG2 zajedno omogućavaju direktnu realizaciju klasičnog algoritma za množenje velikih brojeva. Naime,

$$w = u \cdot v = \sum_{i=0}^{n} u_i \cdot b^i \cdot \sum_{j=0}^{m} v_j \cdot b^j$$

možemo zapisati kao

$$w = \sum_{j=0}^{m} (u \cdot v_j) \cdot b^j$$

i na njega primijeniti algoritam sličan Hornerovoj shemi, što u stvari odgovara ručnom množenju.

Kod ovog algoritma operacije se obavljaju nad nizom cifara odgovarajućih brojeva. Proizvod $w \cdot b$ se zamjenjuje pozivom algoritma $ALG2(w, 1, t_1)$, pri čemu t_1 predstavlja rezultat algoritma ALG2. Dok proizvod $u \cdot v_j$ treba zamijeniti pozivom $ALG1(u, v_j, t_2)$, pri čemu je t_2 rezultat rada ALG1. Na samom kraju treba pozvati algoritam sabiranja na t_1 i t_2.

Brojevi t_1 i t_2 su dodatni trošak memorije i s obzirom da ne koristimo neka svojstva koja bi smanjila broj računskih operacija, u nastavku ćemo upoznati efikasniji algoritam za množenje.

Propozicija 3.2. *Neka su brojevi $u, v \in \mathbb{N}$ dati u pozicionim zapisima u bazi b,*
$u = (u_n, u_{n-1}, \ldots, u_0)_b$, $v=(v_m, v_{m-1}, \ldots, v_0)_b$,
onda ALG3 daje niz cifara u pozicionom zapisu koje predstavljaju $u \cdot v$. Odnosno,
$u \cdot v = w = (u_k, u_{k-1}, \ldots, u_0)_b$,
tako da je $k = \deg_b(w) = n+m$ ili $k = \deg_b(w) = n+m+1$.

Za ulazne podatke ALG3 imamo pozicioni zapis brojeva u i v u nekoj odabranoj

bazi b. Kao rezulatat imamo $w = u \cdot b$, pri čemu je $k = \deg_b(w)$.

```
pocetak:
if(0<=deg(u) && 0<=deg(v)){
    
    deg(w) = deg(u)+deg(v);
    
    for(int i=0;i<deg(u)+1;i++){
        w[i] = 0;
        
    }
    for(int j=0;j<deg(v)+1;j++){
    
    int alg = 0;
    for(int i=0;i<=deg(u);i++){
        int temp = w[i+j] + u[i] * v[j] + alg;
        w[i+j] = temp%(b);
        alg = temp/b;
    }
    w[deg(u)+j+1] = alg;
    }
    if(alg>0){
    deg(w) = deg(u)+deg(v)+1;
    }
    else{
        deg(w)= deg(u)+deg(v);
    }
    
}
    else{
        deg(w)= -1;
    }
kraj
```

Za prostornu složenost važi:

$$\underset{S}{\text{Compl}}(ALG3) = 2 \cdot (l(u) + l(v)) + c,$$

s tim da je $c \leq 7$, ako računamo mogućnost da je $l(w) = l(u) + l(v)$, te dodamo prostor za dužine brojeva i pomoćne promjenljive.

Aritmetička složenost pomenutog algoritma je

$$\underset{A}{\text{Compl}}(ALG3) = 5 \cdot l(u) \cdot l(v) + 2.$$

Vremenska složenost algoritma je

$$\underset{T}{\text{Compl}}(ALG3) \sim l(u) \cdot l(v).$$

3.2 Složenost klasičnog algoritma

Zanimljiv podatak je koliko aritmetičkih operacija je potrebno da bi se dva broja u pozicionom zapisu u nekoj bazi b pomnožila. U sledećoj definiciji uvešćemo oznaku za taj broj u zavisnosti od dužine brojeva koje množimo.

Definicija 3.1. *Neka je ALG bilo koji algoritam za množenje prirodnih brojeva u i v u pozicionom zapisu u nekoj bazi b. Aritmetička složenost algoritma je zavisna od dužine brojeva u i v, i označavamo je sa:*

$$Mul(n, m) = \underset{A}{\text{Compl}}(ALG)$$

ako je l(u) = n, l(v) = m. Ako je n=m, onda označavamo skraćeno sa:

$$Mul(n) = Mul(n, n).$$

Naravno podrazumijeva se da ALG radi korektno za sve prirodne brojeve u i v, odnosno za sve proizvoljne dužine ovih brojeva. I upravo to je bitno, jer bi u suprotnom ovaj problem bio trivijalan. Naime, u tom slučaju može se konstruisati tablica množenja za datu dužinu, pa se množenje tada svodi na jednostavno čitanje rezultata iz tabele.

Takođe smo u definiciji dozvolili da baza b bude fiksirana. To naravno ne utiče na dužinu brojeva jer je ona i dalje proizvoljna. Zbog toga je trajanje množenja zavisno od dužine brojeva, bar kao broj traženja u tablici. Uočimo da je Mul(n,m) dobra mjera potrebnog vremena za sekvencijalnim arhitekturama.

Broj operacija za množenje prirodnih brojeva koji imaju po n cifara u svom pozicionom zapisu je proporcionalan sa kvadratom dužine brojeva.

Svakako ima prostora za bitno ubrzanje klasičnog algoritma za množenje prirodnih brojeva.

Glava 4

"Podijeli pa pomnoži" algoritam za množenje

4.1 Karatsubin algoritam

Kao što smo vidjeli ALG3 ima dosta prostora za ubrzanje. Klasični algoritam ćemo ubrzati metodom 'podjele', tako što ćemo zadatak veličine n svesti na neki niz zadataka manjih veličina. Dakle, postupak ćemo primjenjivati sve dok problem ne postane lako rješiv.

Pretpostavimo da je $l(u) = l(v) = n$. Neka je n paran broj tj $n = 2k$. Ova pretpostavka na omogućava da brojeve

$$u = \sum_{i=0}^{n} u_i \cdot b^i, \quad v = \sum_{i=0}^{n} v_i \cdot b^i \quad (3.1)$$

možemo zapisati u bazi $b^k = b^{n/2}$, u obliku

$$u = U_1 \cdot b^k + U_0, \quad v = V_1 \cdot b^k + V_0, \quad (3.2)$$

pri čemu U_1,V_1 predstavljaju značajnije polovine brojeva u i v, dok U_0,V_0 predstavljaju donje polovine tih brojeva. Dakle imamo da je:

$$U_1 = (u_{2k-1}, \ldots, u_k)_b = \sum_{i=0}^{k-1} u_{i+k} \cdot b^i, U_0 = (u_{k-1}, \ldots, u_0)_b = \sum_{i=0}^{k-1} u_i \cdot b^i$$

$$V_1 = (v_{2k-1}, \ldots, v_k)_b = \sum_{i=0}^{k-1} v_{i+k} \cdot b^i, U_0 = (v_{k-1}, \ldots, v_0)_b = \sum_{i=0}^{k-1} v_i \cdot b^i$$

Dobijamo da brojevi U_1, U_0, V_1, V_0 predstavljaju cifre brojeva u i v u bazi b^k.

Uz ove oznake, imamo da proizvod $w = u \cdot v$, posmatran u bazi b^k ima oblik

$$w = (U_1 \cdot V_1) \cdot b^{2k} + (U_0 \cdot V_1 + U_1 \cdot V_0) \cdot b^k + U_0 + V_0. \quad (3.3)$$

Ovim smo sveli problem množenja brojeva koji imaju n cifara u svom pozicionom zapisu na problem nalaženja 4 proizvoda brojeva koji imaju po $\frac{n}{2}$ cifara u svom pozicionom zapisu.

Lema 4.1. *Neka je $t \in \mathbb{R}$ neka konstanta. Za bilo koji algoritam množenja prirodnih brojeva u pozicionom zapisu, postoji $C \in \mathbb{N}$, takva da za svako $n \in \mathbb{N}$ važi:*

$$Mul(n + t) \leq Mul(n) + C \cdot n, \quad (3.4)$$

pri čemu C zavisi od t, ali ne i od n.

Uzevši u obzir prethodnu lemu pokazuje se da relacija (3.3) ne poboljšava klasični

algoritam, budući da je:

$$Mul(n) = 4 \cdot Mul(n/2) + c_1 \cdot n + c_2,$$

iz čega dobijamo:

$$Mul(n) = O(n^2).$$

Relaciju (3.3) možemo napisati u sledećem obliku:

$$U_0 \cdot V_1 + U_1 \cdot V_0 = (U_1 + U_0) \cdot (V_1 + V_0) - U_0 \cdot V_0 - U_1 \cdot V_1. \quad (3.5)$$

Iz poslednje relacije vidimo da je sada potrebno naći tri proizvoda za razliku od prethodne gdje su bila potrebna 4. Uz pomoć relacija (3.3) i (3.5) dobijamo algoritam koji je otkrio Anatolij Karatsuba (Anatoly Karatsuba) 1960. godine.
U nastavku je prikazan kratak izgled algoritma:

```
pocetak:
t_1=(U[1]+U[0])*(V[1]+V[0]);
t_2=U[0]*V[0];
t_3=U[1]*V[1];
w=t_3*b^(2k) + (t_1-t_2-t_3)* b^(k) + t_2;
kraj
```

Uz pretpotavku da potrebna tri množenja obavljamo istim algoritmom, za aritmetičku složenost dobijamo:

$$Mul(n) \leq 2Mul(n/2) + Mul(n/2 + 1) + c_1 \cdot n. \quad (3.6)$$

U ovoj sumi, srednji član predstavlja složenost množenja $(U_1 + U_0) \cdot (V_1 + V_0)$,

budući da ta dva broja mogu imati n/2 + 1 cifara. Ukoliko se insistira na parnoj dužini brojeva, prethodna lema garantuje da se relacija (3.6) ne mijenja bitno ako bismo dodali vodeću nulu i zamijenili $Mul(n/2+1)Mul(n/2+2)$. Primjenom iste leme za $t = 1$ na relaciju (3.6) dobijamo složenost Karatsubinog algoritma

$$Mul(n) \leq 3Mul(n/2) + c_0 \cdot n, \quad (3.7)$$

pri čemu je c_0 neka konstanta koja ne zavisi od n. Ako algoritam upotrebljavamo rekurzivno, onda relacija (3.7) važi za svako $n > 1$, a kraj je kada dobijemo pojedinačne cifre. Tada koristimo aritmetiku računara u prethodnom algoritmu pa imamo:

$$Mul(1) \leq c_0'. \quad (3.8)$$

Neka je $c = \max\{c_0 \ , \ c_0'\}$. Konačno, imamo rekurzivne relacije

$$Mul(1) \leq c, \quad Mul(n) \leq 3Mul(n/2) + c \cdot n.$$

Ove rekurzivne relacije ćemo riješiti pomoću naredne leme.

Lema 4.2. *Neka je T: N $\rightarrow$R monotono rastuća funkcija i neka su a, d, c iz R^+ konstante takve da je a>d. Ako funkcija T zadovoljava nejednačine*

$$T(1) \leq c, \quad T(n) \leq a \cdot T(n/d) + c \cdot n, \quad n = d^k, \quad k > 0, \quad (3.9)$$

onda imamo da je

$$T(n) = \frac{c}{a-d} \cdot [\, a \cdot n^{\log_d a} - d \cdot n] \quad (3.10)$$

za $n = d^k$, $k > 0$, *i postoji konstanta* $C > 0$ *takva da je*

$$T(n) \leq c \cdot n^{\log_d a}, \quad za \quad \forall n \in \mathbb{N}. \quad (3.11)$$

Dokaz: Iz(3.9) se pokazuje indukcijom da važi

$$T(d^k) = \frac{c}{a-d}[\, a \cdot a^k - d \cdot d^k]\,.$$

Relaciju (3.10) dobijemo za $n = d^k$ uz sledeće jednakosti

$$a^k = d^{k \log_d a} = (d^k)^{\log_d a}.$$

Iz monotonosti funkcije i pretpostavke da je $a > d$, slijedi (3.11).

Funkcija Mul zadovoljava uslove funkcije T iz leme 3.2, a budući da brojeve dužine manje od n možemo dopuniti vodećim nulama do dužine n. Sada iz relacije (3.7) slijedi da za Karatsubin algoritam važi

$$Mul(n) < C \cdot n^{\log_2 3}, n \in \mathbb{N}.$$

Ovo predstavlja značajno ubrzanje klasičnog algoritma.

4.2 Generalizacija Karatsubinog algoritma

U Karatsubinovom algoritmu smo brojeve u i v rastavljali na dva dijela. U ovoj sekciji ćemo brojeve u i v rastavljati na $r+1$ djelova, za neki fiksirani broj $r \in \mathbb{N}$. Pretpostavimo da je $n = (r+1) \cdot k$, za neko $r \in \mathbb{N}$. Brojeve u i v zapisujemo u bazi

b^k na sledeći način:

$$u = \sum_{i=0}^{r} U_i \cdot b^{ki}, \quad v = \sum_{i=0}^{r} V_i \cdot b^{ki} \quad (3.2.1)$$

pri čemu su U_i, V_i k-cifreni brojevi u bazi b, za $i = 0, 1, \ldots, r$. Sada, definišimo polinome $U(x)$, $V(x)$ na sledeći način:

$$U(x) = \sum_{i=0}^{r} U_i \cdot x^i, \quad V(x) = \sum_{i=0}^{r} V_i \cdot x^i,$$

a sa $W(x)$ definišemo proizvod ovih polinoma. Dakle,

$$W(x) = U(x) \cdot V(x) = \sum_{i=0}^{2r} W_i \cdot x^i.$$

Kako je $u = U(b^k), v = V(b^k)$, slijedi da je

$$w = u \cdot v = W(b^k).$$

Za izračunavanje

$$w = \sum_{i=0}^{2r} W_i \cdot b^{ki}, \quad (3.2.3)$$

dovoljno je odrediti koeficijente W_i za svako $i = 0, 1, \ldots, 2r$. Kada nađemo te vrjednosti izraz (3.2.3) se svodi na operacije sabiranja i množenja potencijom baze. To zahtijeva broj operacija proporcionalan sa $n = (r + 1) \cdot k$.

Polinom $W(x)$ je određen sa koeficijentima ili vrijednostima u $2r + 1$ tačaka. Ako izaberemo tačke $0, 1, \ldots, 2r$ tada imamo

$$W(i) = U(i) \cdot V(i), \quad i = 0, 1, \ldots, 2r.$$

Koeficijente W_i možemo izračunati Hornerovom shemom i polinomijalnom aritmetikom, koristeći Njutnov oblik interpolacije polinoma

$$W(x) = \sum_{i=0}^{2r} \frac{1}{i!} \Delta^i W(0) \cdot \prod_{j=0}^{i-1} (x - j) \quad (3.2.4).$$

Odavde je lako izračunati $w = W(b^k)$.

Aritmetička složenost algoritma je

$$Mul(n) \leq C \cdot n^{1+\log_{r+1} 2},$$

dokaz možete naći u [5]. Kada se dužina n može prikazati u računaru, pokazalo se da je algoritam efikasan kada je $r \leq 5$, ali tada n mora biti veliko.

4.3 Odnos Karatsubinog algoritma i algoritma za klasično množenje

U ovoj sekciji uporedićemo klasični i Karatsubin algoritam za množenje. Kod je predstavljen u programskom jeziku Java.

Prvo pogledajmo implementaciju algoritma za klasično množenje.

```java
public class KlasicniAlgoritam {

 public static int [] klasicnoMnozenje(int[] u, int[] v,int b) {

     int deg_u = u.length;
     int deg_v = v.length;
     int[] w;

     if(deg_u >= 0 && deg_v >= 0) {

         w = new int[deg_u + deg_v +1];
         int alg;
         int deg_w = w.length;

         for(int i=deg_v - 1; i >= 0;i--) {
             alg = 0;
             for(int j=deg_u - 1;j >= 0;j--) {
                 int temp = w[i+j+2] + v[i]*u[j] + alg;
                 w[i+j+2] = temp%b;
                 alg = temp/b;

             }
             w[i+1] = alg;

         }
     }
     else {

         w = new int[1];
         w[0] = -1;
     }

     return w;
      }}
```

Brojeve prikazujemo kao nizove cijelih brojeva na sledeći način: Na primjer za broj u, vodeća cifra u pozicionom zapisu je na poziciji $u[0]$ dok je cifra najmanjeg značaja na poziciji $u[n]$.

Na narednoj slici je kod Karatsubinog agoritma.

```java
import java.math.BigInteger;
public class Karatsuba {

private static BigInteger Zero = new BigInteger("0");

public static BigInteger karatsuba(BigInteger x,BigInteger y) {

    int N = Math.max(x.bitLength(), y.bitLength());
    if(N <= 3) {
        return x.multiply(y);
    }

    N = (N/2) + (N%2);

    BigInteger b = x.shiftRight(N);
    BigInteger a = x.subtract(b.shiftLeft(N));
    BigInteger d = y.shiftRight(N);
    BigInteger c = y.subtract(d.shiftLeft(N));

    BigInteger ac = karatsuba(a, c);
    BigInteger bd = karatsuba(b, d);
    BigInteger abcd = karatsuba(a.add(b), c.add(d));

    return
ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(N)).add(bd.shiftLeft(2*N));
            }

}
```

U tablici ispod napravljeno je poređenje klasičnog algoritma i Karatsubinog. Vidi se da je Karatsubin algoritam brži. Za male brojeve je razlika zanemarljiva.

--

Performanse	Klasično množenje	Karatsubin algoritam
Intel Core i7-6700HQ 2.6GHz Turbo Boost up to 3.5GHz	5.103 s	3.300 s
Intel Core i3-5005U CPU @ 2.0 GHz	8.160 s	6.482 s
AMD E-350 APU 1.5GHz	21.700 s	15.260 s

Glava 5

Šenhage-Štrasenov algoritam

U ovom poglavlju ćemo se najprije upoznati sa Furijeovom transformacijom za množenje cijelih brojeva. Takođe, proučićemo i inverznu transformaciju, a nakon toga vidjeti kako su to Štrasen i Šenhage iskoristili za pravljenje do sada najbržeg algoritma za množenje velikih brojeva.

5.1 Diskretna i inverzna Furijeova transformacija

Obično Furijeova transformacija je definisana nad poljem kompleksnih brojeva. Mi ćemo međutim, definisati Furijeovu tranfsormaciju nad komutativnim prstenom $(R, +, \cdot)$.

Definicija 5.1. *Za element $w \in \mathbb{R}$ kažemo da je n-ti korijen iz jedinice ako su zadovoljeni sledeći uslovi:*

1. $w \neq 1$,

2. $w^n = 1$,

3. $\sum_{i=0}^{n-1} w^{ip} = 0$, za $1 \leq p < n$. Elementi $w^0, \ldots, w^{n-1}$ zovu se n-ti korijeni iz jedinice

Uzmimo sada da je $a = [a_0, \ldots, a_{n-1}]^T$ vektor kolone čiji su elementi iz R. Pretpostavimo da n ima multiplikativni inverz u R i da R ima glavni n-ti korijen iz jedinice. Uzmimo da je A $n \times n$ matrica za koju važi:

$$A[i, j] = w^{ij},$$

pri čemu je $0 \leq i, j < n$.

Tada vektor $F(a) = A \cdot a$, čija je i-ta komponenta jednaka $\sum_{k=0}^{n-1} a^k \cdot w^{ik}$, naziva se diskretna Furijeova transformacija od a. Kako je A regularna matrica , tada postoji A^{-1}. Element matrice A u i-toj vrsti i j-toj koloni je w^{ij}, tada na istoj poziciji u matrici A^{-1} je element $(1/\mathrm{n})w^{-ij}$. Vektor $F^{-1}(a) = A^{-1} \cdot a$, čija je i-ta komponenta jednaka:

$$\frac{1}{n} \sum_{k=0}^{n-1} a_k \cdot w^{-ik},$$

zovemo inverzna diskretna Furijeova transformacija od a.

Pogledajmo sada vezu između Furijeove transformacije i polinomne evaluacije i interpolacije. Naime, neka je

$$p(x) = \sum_{i=0}^{n-1} a_i \cdot x^i$$

polinom stepena $n - 1$. Polinom možemo prikazati na dva načina. Jedan je kao niz njegovih koeficijenata, a drugi kao niz njegovih vrijednosti u n različitih tačaka. Sada računanje Furijeove transformacije vektora $[a_0, \ldots, a_{n-1}]^T$ je ekvivalentno konvertovanju prikaza polinoma pomoću koeficijenata $a_0, \ldots, a_{n-1}$ u njegove vrijednosti u tačkama $w_0, \ldots, w_{n-1}$. U suprotnom smjeru , inverzna Furijeova transformacija je ekvivalentna interpolaciji polinomom za date vrijednosti u n-tim korjenima iz jedinice.

Konvolucijski proizvod dva vektora se može uraditi primjenom Furijeovih transformacija. Neka su sada $a = [a_0, \ldots, a_{n-1}]^T$ i $b = [b_0, \ldots, b_{n-1}]^T$ dva vektora. Konvolucijski proizvod dva vektora je vektor $c = [c_0, \ldots, c_{2n-1}]^T$, pri čemu je $c_i = \sum_{j=0}^{n-1} a_j \cdot b_{i-j}$. Ako je $k < 0$ ili $k > n$ uzimamo da je a_k=b_k=0. Dodatno važi da je c_{2n-1}=0.

Razlog za uvođenje konvolucijskog proizvoda je upravo množenje polinoma. Ako imamo dva polinoma

$$p(x) = \sum_{i=0}^{n-1} a_i \cdot x^i \quad i \quad q(x) = \sum_{i=0}^{n-1} b_i \cdot x^i$$

tada je prozivod ova dva polinoma jednak:

$$p(x)q(x) = \sum_{i=0}^{2n-2} (\sum_{j=0}^{i} a_j \cdot b_{i-j}) \cdot x^i.$$

Unutrašnja suma je upravo konvolucijski proizvod vektora koeficijenata polinoma a i b.

Ako imamo dva polinoma stepena $n-1$ predstavljeni vrijednostima u tačkama n-tih korijena iz jedinice, da bi izračunali njihov proizvod možemo jednostavno množiti parove vrijednosti u odgovarajućim tačkama. Ovo sugeriše da je konvolucijski proizvod vektora a i b zapravo inverzna transformacija od proizvoda po komponentama od transformacije tih vektora. Označićemo rečeno na sledeći način:

$$a \star b = F^{-1}(F(a) \cdot F(b)). \quad (4.1.1)$$

Dakle, konvolucijski proizvod može se izračunati tako da se napravi Furijeova transformacija, izračuna proizvod po parovima, i onda se sprovede inverzna transformacija. Sledeća teorema rješava problem predstavljanja proizvoda dva polinoma stepena $n-1$, za koji nam je potrebno $2n-2$ tačaka. Dokaz možete naći u [1].

Teorema 5.1. *Neka su data dva vektora dužine 2n*

$a = [a_0, \ldots, a_{n-1}, 0, \ldots 0]^T$ *i* $b = [b_0, \ldots, b_{n-1}, 0, \ldots, 0]^T$.

Neka su $F(a) = [a'_0, \ldots, a'_{2n-1}]^T$ *i* $F(b) = [b'_0, \ldots, b'_{2n-1}]^T$ *njihove Furijeove transformacije. Tada važi*

$a \star b = F^{-1}(F(a) \cdot F(b))$.

Definicija 5.2. *Neka su dati vektori*

$$a = [a_0, \ldots, a_{n-1}]^T, \quad b = [b_0, \ldots, b_{n-1}]^T$$

dva vektora dužine n. Pozitivna ciklična konvolucija vektora a i b je vektor $c = [c_0, \ldots, c_{n-1}]^T$ *, s tim da važi sledeće:*

$$c_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} + \sum_{j=i+1}^{n-1} a_j \cdot b_{n+i-j}$$

Na sličan način se definiše i negativna ciklična konvolucija s tim što se umjesto znaka '+' između poslednje dvije sume pise znak '-'.

5.2 Šenhage-Štrasenov algoritam

Šenhage i Štrasen su 1970. godine pronašli jedan efikasan algoritam za množenje velikih brojeva. Vremenska složenost algoritma je $O(n \log n \log \log n)$ ako množimo n-bitne brojeve. Možemo zamijeniti n sa 2^n i naći proceduru koja množi (2^n) bitne brojeve u $O(2^n n \log n)$ koraka. Ključna ideja algoritma je množenje brojeva po modulu $2^{2^n} + 1$.

Petpostavimo da je $N = 2^n$ i želimo da nađemo proizvod brojeva u i v po modulu 2^N+1. Ako je na primjer $u = -1$ tada imamo da je $u \cdot v \equiv (2^N+1-v) \pmod{2^N+1}$. Razložimo naše N bitne brojeve na blokove. Neka je $k + l = n, \quad K = 2^k, \quad L = 2^l$, $u = U_{K-1}2^{(K-1)L} + U_1 2^L + U_0$ i $v = V_{K-1}2^{(K-1)L} + V_1 2^L + V_0$.

Odgovarajuće vrijednosti za k i l odredićemo kasnije. Ispostaviće se da je potrebno da bude $k \leq l + 1$, a zatim da je najpogodnije uzeti $l = \lfloor n/2 \rfloor$. Proizvod u i v je zadat sa

$$u \cdot v = y_{2K-2}2^{(2K-2)L} + \ldots + y_1 2^L + y_0, \quad (5.1.1)$$

pri čemu je $y_i = \sum_{j=0}^{K-1} U_j \cdot V_{i-j}$, $0 \leq i < 2K$. Za $j < 0$ ili $j > K - 1$ uzimamo da je $U_j = V_j$=0. Član y_{2k-1}=0 i on se uzima samo zbog simetričnosti.
U nastavku izlaganja izraz 'DFT' označavaće 'Diskretnu Furijeovu transformaciju'. Proizvod u i v može biti izračunat pomoću konvolucione teoreme. Množenje odgovarajućih parova DFT bi zahtijevalo 2K množenja. Međutim upotrebom ciklične konvolucije broj množenja smanjujemo na K. Zbog ovoga se množenje izvodi po modulu $2^N + 1$. Kako je $K \cdot L = N$, imamo da je $2^{KL} \equiv -1 \pmod{2^N + 1}$. Tada iz (5.1.1) dobijamo

$$uv \equiv (w_{(K-1)}2^{(K-1)L} + \ldots + w_1 2^L + w_0) \pmod{2^N + 1},$$

gdje je $w_i = y_i - y_{K+i}, 0 \leq i < K$.
Kako je proizvod dva broja od L bitova manji od 2^{2L} i kako su y_i i y_{K+i} sume $i + 1$ i $K - (i + 1)$ takvih proizvoda respektivno, tada $w_i = y_i - y_{k+i}$ mora biti u intervalu $-(K - 1 - i)2^{2L} < w_i < (i + 1)2^{2L}$. Tada postoji najviše $K2^{2L}$ mogućih vrijednosti koje w_i može uzeti. Ukoliko je moguće izračunati w_i po modulu $K \cdot 2^{2L}$ onda možemo i $uv \pmod{2^N + 1}$ sa $O(K \log(K2^{2L}))$ dodatnih koraka.

Da bismo izračunali $w_i \pmod{K2^{2L}}$ mi računamo w_i dva puta, jedan po modulu K a drugi po modulu $2^{2L}+1$. Ovo je druga ključna ideja našeg algoritma. Neka je dalje $w_i' \equiv w_i \pmod{K}$ $w_i'' \equiv w_i \pmod{2^{2L}+1}$. Kako je K stepen dvojke, a 2^{2L}+1 neparan, to su i K i 2^{2L}+1 uzajamno prosti brojevi. Zbog toga w_i može se izračunati kao

$$w_i = (2^{2L})((w_i' - w_i'') \pmod{K} + w_i'',$$

i važi da je w_i između $-(K-1-i)2^{2L}$ i $(i+1)2^{2L}$. Rad potreban da se izračuna w_i iz w_i' i w_i'' je $O(L+\log(K))$ za jedno w_i, dok za sve iznosi $O(N)$.

Za izračunavanje w_i' ne treba mnogo vremena jer je k mnogo manje od $2L$. Prvo možemo izračunati $u_i' = u_i \pmod{K}$ i $v_i' = v_i \pmod{K}$ i $y_i' = \sum_{j=0}^{2K-1} u_j' \cdot v_{i-j}'$. Slažemo u_i-ove i v_i-ove zajedno, razdvajajući ih sa $2\log K$ nula. Na taj način dobijamo brojeve

$$\overline{\mathrm{u}} = \sum_{i=0}^{K-1} u_i' \cdot 2^{(3\log K)i} \quad i \quad \overline{\mathrm{v}} = \sum_{i=0}^{K-1} v_i' \cdot 2^{(3\log K)i}.$$

Nakon toga izvršimo množenje brojeva $\overline{\mathrm{u}}$ i $\overline{\mathrm{v}}$ pomoću Karakuba-Hofmanovog algoritma koji zahtijeva manje od O(N) koraka. Vidimo da je $\overline{\mathrm{u}}\cdot\overline{\mathrm{v}} = \sum_{i=0}^{2K-1} y_i' \cdot 2^{(3\log K)i}$ i $y_i' < 2^{3\log K}$. Sada w_i' računamo kao $w_i' \equiv (y_i' - y_{K+i}') \pmod{K}$.

I na kraju treća ključna ideja ovog algoritma je računanje w_i'' koristeći pojam ciklične konvolucije. Ona zahtijeva izvršavanje DFT-a, i množenje odgovarajućih parova i inverznu DFT. Neka je dalje $m = 2^{2L}+1$, $r = 2^{l+1-k}$, a $\psi = 2^r$ i $w = \psi^2 = 2^{4L/K}$. Ispostavlja se da w i K imaju multiplikativne inverze po modulu m, a w je primitivni K-ti korjen iz jedinice. Tada je negativna ciklična konvolucija vektora $[u_0, \psi u_1, \ldots, \psi^{K-1}u_{K-1}]$ i $[v_0, \psi v_1, \ldots, \psi^{K-1}v_{K-1}]$ jednaka:

$$[y_0 - y_k, \psi(y_1 - y_{K-1}), \ldots, \psi^{K-1}(y_{K-1} - y_{2K-1})],$$

pri čemu je $y_i = \sum_{j=0}^{K-1} u_j \cdot v_{i-j}$ za $0 < i < 2K-1$.

5.3 'Korak po korak' - Kompletan Šenhage-Štrasenov algoritam

Kao ulazne podatke imamo dva $N = 2^n$ bitna cijela broja u i v. Kao izlaz dobijamo $N+1$ bitni proizvod u i v po modulu $2^N + 1$.

Ako je n malo, množimo u i v po modulu $2^N + 1$ sa nekim pogodnijim algoritmom. Za veliko n izračunamo $l = \lfloor n/2 \rfloor$, $k = n - l$, $L = 2^l$, $\psi = 2^{l+1-k}$, $w = \psi^2$. Dalje, izrazimo $u = \sum_{i=0}^{K-1} u_i \cdot 2^{Li}$ i $v = \sum_{i=0}^{K-1} v_i \cdot 2^{Li}$, pri čemu su u_i i v_i brojevi između 0 i $2^L - 1$. Nakon toga slijede koraci:

1. Izračunati DFT, po modulu $2^{2L} + 1$, nizova

$$[u_0, \psi u_1, \ldots, \psi^{K-1} u_{K-1}] \quad i \quad [v_0, \psi v_1, \ldots, \psi^{K-1} v_{K-1}]$$

koristeći w kao primitivni korjen.

2. Izračunati zatim proizvode odgovarajućih parova DFT dobijenih u prethodnom koraku po modulu $2^{2L}+1$. Ovo se ostvaruje rekurzivnim pozivom Šenhage-Štrasenovog algoritma.

3. Izračunati inverznu DFT po modulu $2^{2L}+1$ vektora proizvoda parova iz prethodnog koraka. Kao rezultat dobijamo $[w_0, \psi w_1, \ldots, \psi^{K-1} w_{K-1}]$ po modulu $2^{2L}+1$, gdje je w_i član negativne-ciklične konvolucije vektora $[u_0, u_1, \ldots, u_{K-1}]$ i $[v_0, v_1, \ldots, v_{K-1}]$. Zatim odrediti $w_i'' \equiv w_i \pmod{2^{2L}+1}$ množenjem $\psi^i w_i$ sa ψ^{-i} po modulu $2^{2L}+1$.

4. a) Izračunati $u_i' = u_i \pmod{K}$, $v_i' = v_i \pmod{K}$, za $0 \leq i < K$.

b) Izračunati $\overline{u} = \sum_{i=0}^{K-1} u_i' \cdot 2^{(3 \log K)i}$ i $\overline{v} = \sum_{i=0}^{K-1} v_i' \cdot 2^{(3 \log K)i}$. Ovo ostvarujemo nižući

u'_i-ove i v'_i-ove zajedno sa $2\log K$ nula između njih.

c) Proizvod $\overline{u}\cdot\overline{v}$ je oblika $\sum_{i=0}^{2K-1} y'_i \cdot 2^{(3\log K)i}$, gdje je $y'_i = \sum_{j=0}^{2K-1} u'_j v'_{i-j}$ i $y'_i < 2^{3\log K}$.

Izdvojiti y'_i i izračunati $w'_i \equiv (y'_i - y'_{K+i}) \pmod{K}$, za $0 \leq i < K$.

5. Izračunati

$w'''_i = (2^{2L}+1)((w'_i - w''_i) \pmod{K} + w''_i$ i

$w_i = w'''_i$ ako je $w'''_i < (i+1)2^{2L}$ ili $w_i = w'''_i - K(2^{2L}+1)$ ako je $w'''_i \geq (i+1)2^{2L}$.

6. Izračunati $\sum_{i=0}^{K-1} w_i \cdot 2^{Li}$ po modulu $2^N + 1$. Dobili smo traženi rezultat.

Teorema 5.2. *Vrijeme izvršavanja Šenhage-Štrasenovog algoritma je*

$$O(N \log N \log\log N)$$

koraka.

Glava 6

Dijeljenje velikih brojeva

Kao što smo pomenuli na samom početku dijeljenje brojeva u i v predstavlja zapravo množenje broja u sa recipročnom vrijednošću broja v. Dakle, može se pretpostaviti da se dijeljenje može podjednako brzo obaviti kao i množenje. Međutim, jedan od problema je što recimo, recipročna vrijednost broja v može imati beskonačan zapis u bazi b. Naš cilj biće naći dovoljno dobru aproksimaciju $1/v$.

6.1 Aproksimacija recipročne vrijednosti

Kako nas zanima cjelobrojni koeficijent

$$q = \lfloor \frac{u}{v} \rfloor$$

aproksimacija $\frac{1}{v}$ mora biti toliko tačna da nam omogući nalaženje broja q ili njegove dobre ocjene $\overline{\mathrm{q}}$.

Neka je $l(u) = n$ i $l(v) = m$. Kada bismo našli aproksimaciju w za $1/v$ sa tačnošću ϵ

$$1/v = w + \epsilon,$$

gdje je

$$|\epsilon| \leq b^{-n},$$

onda imamo da je

$$|\frac{u}{v} - u \cdot w| = u \cdot |\epsilon| \leq 1$$

Ako uzmemo da je

$$\overline{\mathrm{q}} = \lfloor u \cdot v \rfloor$$

imamo da je

$$|q - \overline{\mathrm{q}}| \leq 1.$$

Za nalaženje w koristićemo Njutnovu metodu za rešavanje jednačine

$$\frac{1}{x} - v = 0$$

Pretpostavimo da možemo naći dovoljno dobru aproksimaciju $w_{(0)}$ za $\frac{1}{v}$. Svaku sledeću aproksimaciju možemo dobiti koristeći rekurzivnu relaciju

$$w_{(k+1)} = w_{(k)} \cdot (2 - v \cdot w_{(k)}) \quad , \quad k \geq 0 \quad (5.3)$$

. Označimo dalje $e_{(k)}$ grešku k-te aproksimacije

$$e_{(k)} = \frac{1}{v} - w_{(k)}, \quad k \geq 0.$$

Iz (5.3) imamo da je

$$e_{(k+1)} = v \cdot (e_{(k)})^2, \quad k \geq 0 \quad (5.4).$$

Ukoliko osiguramo da važi $|e_{(k)}| \leq b^{-(m+d)}$ onda iz (5.4), zbog $v < b^m$ slijedi da je

$$|e_{(k+1)}| \leq b^{-(m+2d)} \quad (5.5).$$

Kako je $b^{m-1} \leq v \leq b^m, za \quad \frac{1}{v}$ imamo da je

$$b^{-m} < \frac{1}{v} \leq b^{-(m-1)} \quad (5.6)$$

pa slijedi da svaki sledeći korak udvostručuje broj tačnih cifara.

Da bi praktično realizovali algoritam potrebno je brojeve $w_{(k)}$ prikazati konačnim nizom cifara u bazi b. Zbog relacije (5.6) broj $\frac{1}{v}$ ima sledeći prikaz u bazi b

$$\frac{1}{v} = b^{-(m-1)} \cdot \sum_{i=0}^{\infty} v_i' b^{-i}$$

, gdje je

$$0 \leq v_i' < b, \quad i \in \mathbb{N}$$

i važi sledeće

$$b^{-m} < \frac{1}{v} < b^{-(m-1)} \iff v_0' = 0 \quad v_1' > 0$$

$$\frac{1}{v} = b^{-(m-1)} \iff v_0' = 1 \quad v_i' = 0, \quad \forall i \in \mathbb{N}.$$

Pretpostavimo da imamo zadatu preciznost p i da želimo izračunati aproksimaciju w' za $\frac{1}{w}$, tako da važi

$$b^{m-1+p} \cdot w' = \lfloor \frac{b^{m-1+p}}{v} \rfloor = \sum_{i=0}^{p} v_i' b^{p-i} = \sum_{i=0}^{p} w_i b^i \quad (5.7)$$

pri čemu je $w_i = v_{p-i}'$ i $l(w') = p$.

Iz ovoga vidimo da u relaciji (5.3) lako možemo preći na cijele brojeve. Ako tu relaciju

pomnožimo sa b^{m-1+p} i definišemo

$$w'_{(k)} = b^{m-1+p} \cdot w_{(k)}, \quad k \geq 0,$$

dobijamo rekurzivnu relaciju

$$w'_{(k+1)} = 2w'_{(k)} - \frac{v}{b^{m-1+p}} \cdot (w'_{(k)})^2, \quad k \geq 0,$$

u kojoj se svaka operacija može obaviti koristeći cjelobrojnu aritmetiku.
Još jedna mogućnost je da povećamo p tokom iteracije, duplirajući p u svakoj sledećoj iteraciji. Pretpostavimo da je $b = 2$ i da je broj m potencija broja 2, odnosno

$$m = 2^l,$$

i da je tražena tačnost $p = m$. Osnovna ideja našeg algoritma DIV je da u k-tom koraku koristimo tačnost $p_k = 2^k$ i da koristimo p_k vodećih cifara broja v.
Ulazni podaci našeg algoritma su: niz cifara $(v_{m-1}, \ldots, v_0)$ prirodnog broja v u bazi 2, uz pretpostavku da je $m = l(v)$ i da je $m = 2^l$ za neko $l \in \mathbb{N}$. Kao rezultat imamo niz cifara $(w_{r-1}, \ldots, w_0)$ broja $w = \lfloor 2^{2m-1}/v \rfloor$ pri čemu je $r = l(w)$. U nastavku pogledajte izgled algoritma.

```
pocetak:
m = deg(v)+1;
(w_1,w_0)=(1,0);
p=1;
if(m>1){
    while(p=m){
        p=2*p;
(s[2p-1],..,s[0])=(w[p/2],..,w[0])*2^(3p/2) - 
(w[p/2],..,w[0])^2 * (v[m-1],..,v[m-p]);
(w[p],..,w[0])=(s[2p-1],..,s[p-1]);
for(i=2;i<=0;i--){
    f((((w[p],..,w[0])+2^i)*(v[m-1],..,v[m-p]))<=2^(2p-1)){
(w[p],..,w[0])=(w[p],..,w[0])+2^i;
            }
        }
    }
    if(w[m]=0){
        deg(w)=deg(v);
    }
    else{
        deg(w)=deg(v)+1;
    }
    kraj.
```

Glava 7

Zaključak

Pokazali smo kako se množenje i dijeljenje velikih brojeva, zapisanih u nekoj proizvoljnoj bazi b, može brzo i efikasno izvesti uz pomoć računara i naravno dobrog algoritma. Na početku smo se upoznali sa klasičnim algoritmom za množenje, a potom smo vidjeli kako je Karatsuba konstruisao algoritam koji je dosta efikasniji od klasičnog. Potom smo pomenuli i opisali rad do sada najbržeg algoritma za množenje velikih brojeva, a to je Šenhage-Štrasenov algoritam. Na samom kraju pokazali smo da se algoritam dijeljenja oslanja u velikom na množenje.

Svakako, razumno je očekivati da će se u budućnosti parametri brzine i efikasnosti popraviti.

Bibliografija

[1] The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, 1976, A. V. Aho, J. E. Hopcroft i Ullman J. D.,

[2] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, 1997

[3] Paralelni algoritmi za množenje velikih brojeva i testovi primalnost- Milenko Mosurović i Žarko Mijajlović

[4] Skripta iz aritmetickih i algebarskih algoritama - Univerzitet u Zagrebu

[5] Predavanja i oblikovanja i analize algoritama - Saša Singer, Univerzitet u Zagrebu