Analiza algoritama za izvođenje aritmetičkih operacija

Pirija, Gabrijela

Analiza algoritama za izvođenje aritmetičkih operacija

Pirija, Gabrijela

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Gabrijela Pirija

ANALIZA ALGORITAMA ZA IZVOĐENJE ARITMETIČKIH OPERACIJA

Diplomski rad

Zagreb, srpanj, 2020

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Gabrijela Pirija

ANALIZA ALGORITAMA ZA IZVOĐENJE ARITMETIČKIH OPERACIJA

Diplomski rad

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

Zagreb, srpanj, 2020.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Veliko hvala mojoj obitelji na potpori i pomoći koju su mi pružali za vrijeme mojeg cjelokupnog školovanja. Hvala i mom mentoru izv. prof. dr. sc. Saši Singeru koji mi je uvelike pomogao u pisanju ovog rada i bez njega bi to trajalo puno duže. Na kraju se želim zahvaliti svim prijateljima, a najviše onima koje sam upoznala na fakultetu. Zbog njih su svi teški trenuci bili izdržljivi, a oni lijepi još ljepši.

Sadržaj

Uvod

Postoje razni algoritmi za izvršavanje aritmetičkih operacija zbrajanja, oduzimanja, množenja i dijeljenja, na prirodnim brojevima. Izbor algoritma koji će se koristiti u konkretnim situacijama ovisi o njegovoj složenosti, a najčešće je to vremenska složenost. Algoritmi za zbrajanje i oduzimanje brojeva imaju linearnu vremensku složenost, pa su samim time optimalni. Klasični algoritmi za množenje i dijeljenje brojeva, koji su u ovom radu prvi obrađeni, imaju kvadratnu vremensku složenost. To su algoritmi koji najbliže opisuju postupak za ručno množenje, odnosno, dijeljenje prirodnih brojeva. Klasični algoritam za množenje prirodnih brojeva nije optimalan zato što postoje brži algoritmi za množenje. Jedan od tih algoritama je Karatsubin algoritam, koji je detaljnije opisan u radu. Dijeljenje prirodnih brojeva m i n množemo promatrati kao množenje broja m s recipročnom vrijednosti broja n. U tom slučaju koristimo algoritam za traženje recipročne vrijednosti prirodnog broja i, na kraju, iskoristimo neki od brzih algoritama za množenje. Algoritam za traženje recipročne vrijednosti prirodnog broja, čiji detalji se mogu pronaći u ovom radu, temelji se na Newtonovoj metodi za rješavanje nelinearnih jednadžbi. Svi do sada navedeni algoritmi zahtijevaju prikaz ulaznih brojeva u pozicijskom zapisu. Baza $b \geq 2$ na kojoj se temelji pozicijski prikaz brojeva može biti proizvoljna. Modularni zapis je alternativni način za prikaz prirodnih brojeva. Postoje brzi algoritmi za množenje brojeva, kod kojih je ključno da su brojevi prikazani u modularnom zapisu. Modularna aritmetika može imati linearnu složenost ukoliko se moduli za modularni zapis brojeva dobro izaberu. U radu je opisana Schönhageova metoda za množenje prirodnih brojeva, koja koristi modularnu aritmetiku. Također, jedan od najbržih algoritama za množenje prirodnih brojeva, Schönhage–Strassenov algoritam, između ostalog koristi i modularni prikaz brojeva.

U ovom radu su prvo detaljnije opisani klasični algoritmi za množenje i dijeljenje brojeva. Opisan je postupak po kojem algoritmi rade te njihova složenost. Nakon toga slijedi analiza algoritma za računanje recipročne vrijednosti prirodnog broja, na kojem se temelji brzo dijeljenje brojeva. Potom navodimo Karatsubin algoritam. Na kraju definiramo modularnu aritmetiku te Schönhageovu metodu za množenje prirodnih brojeva.

Poglavlje 1

Klasični algoritmi za množenje i dijeljenje

1.1 Pozicijski prikaz prirodnih brojeva

Za analizu aritmetičkih operacija nad prirodnim brojevima potrebno je definirati zapis brojeva nad kojima izvršavamo navedene operacije. Pozicijski brojevni sustav je skup pravila po kojima jednoznačno možemo zapisati proizvoljni prirodni broj a kao konačnu linearnu kombinaciju potencija izabrane baze b, npr.

$$a = (a_n a_{n-1} \dots a_k \dots a_2 a_1 a_0)_b = a_n b^n + a_{n-1} b^{n-1} + \dots + a_k b^k + \dots + a_2 b^2 + a_1 b + a_0.$$

Tvrdnju ćemo formalizirati u sljedećem teoremu i definiciji.

Teorem 1.1.1. *Neka je $b \in \mathbb{N}, b \geq 2$, proizvoljan prirodni broj. Za svaki prirodni broj $a \in \mathbb{N}$, postoje broj $n \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$ i brojevi $a_0, \dots, a_n$ takvi da je*

$$a = \sum_{i=0}^{n} a_i b^i. \tag{1.1}$$

Brojevi $a_0, \dots, a_n$ mogu, općenito, biti i kompleksni. Uz dodatno ograničenje

$$a_i \in \mathbb{N}_0, \quad i = 0, \dots, n,$$

i takozvanu normalizaciju

$$\begin{aligned} &0 \leq a_i < b, \quad i = 0, \dots, n-1, \\ &0 < a_n < b, \end{aligned} \tag{1.2}$$

prikaz (1.1) *je jedinstven.*

Dokaz teorema može se pronaći u [7].

Definicija 1.1.2. *Broj b zovemo baza, a broj n je najviša potencija baze ili stupanj broja a u bazi b, što označavamo s*

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

Brojevi $a_0, \ldots, a_n$ *su znamenke broja a u bazi b, a znamenku* a_n *zovemo vodeća ili najznačajnija znamenka broja a. Ako vrijedi* (1.2)*, tj. ako su znamenke normalizirane, onda relaciju* (1.1) *zovemo pozicijski zapis broja a u bazi b, i skraćeno ju označavamo s*

$$a = (a_n \ldots a_0)_b$$

ili

$$a = a_n \ldots a_0,$$

ako je iz konteksta jasno u kojoj bazi b se vrši prikaz.

1.2 Klasični algoritam za množenje

Klasični algoritam za množenje prirodnih brojeva je vrlo sličan postupku ručnog množenja na papiru. Prvo ćemo ga analizirati na način da ukratko opišemo postupak, zatim ćemo zapisati algoritam. Na kraju ćemo navesti propoziciju koja kaže da navedeni algoritam za klasično množenje zaista računa umnožak dva proizvoljna prirodna broja.

Neka su $u = (u_m u_{m-1} \ldots u_1 u_0)_b$ i $v = (v_n v_{n-1} \ldots v_1 v_0)_b$ prikazi dva prirodna broja u bazi b. Iz njihovih prikaza možemo zaključiti da su brojevi u i v, redom, stupnja m, odnosno n. Umnožak $u \cdot v$ brojeva u i v označimo s $w = (w_l \ldots w_1 w_0)$, gdje je l stupanj broja w. Za stupanj broja w vrijedi da je $l = m + n$ ili $l = m + n + 1$, što ćemo vidjeti i u samom algoritmu. Algoritam klasičnog množenja radi tako da računamo parcijalne produkte $(u_m u_{m-1} \ldots u_1 u_0)_b \times v_j$, za $j = 0, \ldots, n$, te ih zbrajamo na odgovarajući način.

Slijedi opis algoritma koji kao ulaz prima brojeve $u = (u_m \ldots u_1 u_0)_b$ i $v = (v_n \ldots v_1 v_0)_b$ u pozicijskom zapisu u bazi b, te kao rezultat daje njihov umnožak $w = (w_l \ldots w_1 w_0)_b$. U algoritmu se prvo radi provjera da su brojevi u i v nenegativnog stupnja, odnosno da imaju barem jednu znamenku i da su različiti od 0, kako bi se nastavilo s postupkom množenja. Ukoliko nisu, odnosno ako je $u = 0$ ili $v = 0$, algoritam ne računa njihov umnožak, već u stupanj l broja w zapisuje -1, te vraća $w = 0$.

Algoritam 1: Klasični algoritam za množenje (*MUL*)

Rezultat: $w = (w_l \dots w_1 w_0)$.
if $m \geq 0 \wedge n \geq 0$ **then**
  $l = n + m$;
  $w_j = 0$, za $0 \leq j \leq m$;
  for $j \leftarrow 0$ **to** n **by** 1 **do**
    if $v_j \neq 0$ **then**
      $k = 0$;
      for $i \leftarrow 0$ **to** m **by** 1 **do**
        $t = u_i \times v_j + w_{i+j} + k$;
        $w_{i+j} = t \bmod b$;
        $k = \left\lfloor \frac{t}{b} \right\rfloor$;
      $w_{j+m+1} = k$;
    else
      $w_{j+m} = 0$;
  if $k > 0$ **then**
    $l = m + n + 1$;
  else
    $l = m + n$;
else
  $l = -1$;

Klasični algoritam za množenje brojeva kreće tako da se znamenke rješenja postave na 0, zato što kasnije u algoritmu sudjeluju u računanju pomoćne varijable t. Pomoćna varijabla k, koja služi za spremanje prijenosa, isto se postavlja na 0, jer u prvom koraku ne moramo ništa ”pamtiti”. Vanjska petlja ide po znamenkama drugog broja, tj. broja v, a unutarnja po znamenkama prvog, tj. u. Prvo se množe najmanje značajne znamenke, odnosno jedinice oba broja. Ukoliko je njihov umnožak veći ili jednak bazi, u varijablu k upisujemo ”višak”, odnosno deseticu tog umnoška. Jedinica tog umnoška se upisuje u najmanje značajnu znamenku našeg rezultata. Desetica, koju smo zapamtili u varijabli k, sudjeluje u računanju iduće znamenke rješenja, kada množimo deseticu prvog broja s jedinicom drugog broja. Radimo isto kao u prošlom koraku, deseticu pišemo u znamenku rješenja, a stoticu pamtimo. Tako nastavljamo dok ne dođemo do najznačajnije znamenke broja u.

Nakon toga krećemo s množenjem jedinica prvog broja u s deseticama broja v. Iz tog razloga, kod računanja nove znamenke rješenja koja prikazuje desetice, moramo pamtiti znamenku desetica koju smo prije izračunali. Tu se događa zbrajanje koje, kada množimo na papiru, radimo tek nakon što izmnožimo svaku znamenku sa svakom i zapišemo umnoške s odgovarajućim pomakom. Algoritam tako nastavlja dalje, dok ne dođe do najznačaj-

nijih znamenaka oba broja. Ukoliko u zadnjem koraku imamo "višak" koji pamtimo u varijabli k, tada naše rješenje ima $n + m + 1$ znamenaka.

Vidimo da algoritam množi svaku znamenku sa svakom, pamti prijenose i zbraja ih na odgovarajući način. Ovaj algoritam je najbrži algoritam za množenje relativno malih brojeva, ali za veće brojeve, koji imaju ≥ 5 znamenaka, postoje algoritmi koji brže izračunaju rješenje. Više o njima obradit ćemo kasnije u ovom radu.

Propozicija 1.2.1. *Ako su brojevi u i v dani pozicijskim zapisima u bazi b*

$$u = (u_m u_{m-1} \dots u_1 u_0)_b, \quad v = (v_n v_{n-1} \dots v_1 v_0)_b,$$

onda klasični algoritam za množenje daje niz znamenki pozicijskog zapisa broja $u \cdot v$

$$u \cdot v = w = (w_l \dots w_1 w_0)_b,$$

gdje je

$$l = \deg_b(w) \in \{n + m, n + m + 1\}.$$

Dokaz propozicije može se pronaći u [7].

Primjer 1.2.2. *Izračunajmo umnožak brojeva* 874 *i* 96 *navedenim algoritmom. U sljedećoj tablici navedeni su koraci algoritma za dani primjer.*

i	j	u_i	v_j	t	k	w_4	w_3	w_2	w_1	w_0
					0			0	0	0
0	0	4	6	24	2			0	0	4
1	0	7	6	44	4			0	4	4
2	0	8	6	52	5		5	2	4	4
0	1	4	9	40	4		5	2	0	4
1	1	7	9	69	6		5	9	0	4
2	1	8	9	83	8	**8**	**3**	**9**	**0**	**4**

Iz tablice vidimo da je $874 \cdot 96 = 83904$.

1.3 Klasični algoritam za dijeljenje

Prije analize klasičnog algoritma za dijeljenje brojeva, uvodimo potrebne definicije i teoreme. Prvo ćemo definirati što znači da je broj djeljiv nekim drugim brojem. Zatim ćemo dokazati Euklidov teorem, na kojem se temelji klasični algoritam za dijeljenje prirodnih brojeva. Nakon toga ćemo opisati pomoćne algoritme, kojima se koristimo u klasičnom algoritmu za dijeljenje prirodnih brojeva.

Iako promatramo dijeljenje prirodnih brojeva, iskazat ćemo i dokazati jače tvrdnje od onih koje su nama potrebne. Definirat ćemo relaciju "biti djeljiv" za cijele brojeve i Euklidov teorem za cijele brojeve. Navedeno se može pronaći u [3].

Definicija 1.3.1. *Neka su $a \neq 0$ i b cijeli brojevi. Kažemo da je b djeljiv s a, odnosno da a dijeli b, ako postoji cijeli broj x takav da je $b = a \cdot x$. To zapisujemo s $a \mid b$. Tada kažemo da je a djelitelj od b, te da je b višekratnik od a. Ako b nije djeljiv s a, odnosno ako a ne dijeli b, onda pišemo $a \nmid b$.*

Teorem 1.3.2. *(Euklidov teorem) Za proizvoljan prirodni broj a i cijeli broj b, postoje jedinstveni cijeli brojevi q i r takvi da je $b = a \cdot q + r$, i vrijedi $0 \leq r < a$.*

Dokaz. Prvo dokazujemo egzistenciju. Promatramo skup $\{b - am \mid m \in \mathbb{Z}\}$. Najmanji element tog skupa koji nije negativan označimo s r. Kako je r najmanji element skupa, očito je da tada vrijedi $0 \leq r < a$. Broj r dobili smo za točno određeni broj (označimo ga s q) za koji vrijedi $b - aq = r$, odnosno $b = aq + r$.

Dokažimo sada jedinstvenost brojeva q i r. Pretpostavimo suprotno, tj. postoje i brojevi q_1 i r_1 takvi da je $b = aq_1 + r_1$. Pokažimo da mora vrijediti $r = r_1$. Pretpostavimo da je $r < r_1$. Tada je $0 < r_1 - r < a$, te je $r_1 - r = (b - aq_1) - (b - aq) = aq - aq_1 = a(q - q_1) \geq a$. Došli smo do kontradikcije. Analogno dokazujemo da ne može biti ni $r_1 < r$. Možemo zaključiti da je $r_1 = r$, stoga je i $q_1 = q$. □

Kod algoritma za klasično dijeljenje brojeva s ostatkom krećemo od najznačajnijih znamenki. Kada dijelimo broj u s brojem v "na ruke", možemo reći da koristimo pogađanje i iskustvo onoga tko računa. Ako takav postupak želimo egzaktno prikazati nekim algoritmom, morat ćemo računati približne vrijednosti traženog broja te korigirati rezultat dok ne zadovoljimo tražene uvjete. Također ćemo glavni problem rastaviti na manje korake. Ako pretpostavimo da je $u > v$, onda broj u dijelimo s potencijom baze kako bismo dobili broj u', koji, kada ga podijelimo s v, daje jednoznamenkasti kvocijent. Iz tog razloga ćemo prvo obraditi algoritme za dijeljenje s potencijom baze i algoritam za dobivanje jednoznamenkastog kvocijenta.

Slijedi opis algoritma za dijeljenje proizvoljnog prirodnog broja $u = (u_n \dots u_1 u_0)_b$ zapisanog u bazi b, s nekom potencijom p baze, gdje je $p > 0$.

Algoritam 2: Dijeljenje potencijom baze (*DIVB*)

Rezultat: $q = (q_k \dots q_1 q_0)_b$, $r = (r_l \dots r_1 r_0)_b$, gdje je $q = \left\lfloor \frac{u}{b^p} \right\rfloor$, a $r = u \bmod b^p$, uz $k = \max\{-1, n-p\}$.

if $n \geq p$ **then**
  $k = n - p$;
  $l = p - 1$;
  while $(u_l = 0) \wedge (l \geq 0)$ **do**
    $l = l - 1$;
  if $u_l > 0$ **then**
    for $i \leftarrow 0$ **to** l **by** 1 **do**
      $r_i = u_i$;
  else
    $l = -1$;
  for $i \leftarrow 0$ **to** k **by** *1* **do**
    $q_i = u_{i+p}$;
else
  $k = -1$;
  $l = n$;
  for $i \leftarrow 0$ **to** l **by** 1 **do**
    $r_i = u_i$;

Ako je $n \geq p$, algoritam radi na način da u broj q zapisuje znamenke od u, počevši od p-te pa do najznačajnije, a za ostatak r preskače one znamenke od $(p-1)$-og do 0-tog mjesta koje su 0, te sprema one koje nisu. Točnije,

$$q = \sum_{i=0}^{n-p} u_{i+p} b^i, \quad r = \sum_{i=0}^{p-1} u_i b^i, \quad \text{za } n \geq p.$$

Ako je $n < p$, očito je $q = 0$ i $r = u$, jer je $b^p > u$.

Kod algoritma za traženje jednoznamenkastog kvocijenta, samo ime govori da tražimo broj $q = \left\lfloor \frac{u}{v} \right\rfloor$ takav da vrijedi $0 \leq \left\lfloor \frac{u}{v} \right\rfloor < b$. Zbog toga možemo zaključiti da broj u ima uvijek najviše jednu znamenku više od v, tj. vrijedi $n \leq m + 1$, gdje je $n = \deg(u)$ i $m = \deg(v)$. Taj uvjet nam govori da je

$$q = \left\lfloor \frac{u}{v} \right\rfloor \leq b - 1, \text{ odnosno } \frac{u}{v} < b, \text{ pa vrijedi } \frac{u}{b} < v, \ \left\lfloor \frac{u}{b} \right\rfloor < v.$$

To znači da je

$$(u_{m+1} u_m \dots u_1)_b < (v_m v_{m-1} \dots v_0)_b.$$

Ovome problemu pristupamo tako da pronalazimo približnu vrijednost kvocijenta q temeljem najznačajnijih znamenaka brojeva u i v. Postavimo

$$\hat{q} = \min\left(\left\lfloor \frac{u_n b + u_{n-1}}{v_{n-1}} \right\rfloor, b-1\right),$$

što znači da $\hat{q}$ računamo tako da dvije najznačajnije znamenke broja u podijelimo najznačajnijom znamenkom broja v. Ukoliko je rezultat $> b$, onda ga zamijenimo s $b-1$. Sljedeći teorem, dostupan u [5], govori kako je ovakva procjena relativno dobra, odnosno da pogreška neće biti veća od 2.

Teorem 1.3.3. *Ako je* $v_{n-1} \geq \left\lfloor \frac{b}{2} \right\rfloor$, *onda vrijedi* $\hat{q} - 2 \leq q \leq \hat{q}$.

Dokaz. Pokažimo prvo drugu nejednakost, odnosno $q \leq \hat{q}$. Kako je $q \leq b-1$, onda je teorem sigurno točan za $\hat{q} = b-1$. Inače, ako je $\hat{q} = \left\lfloor \frac{u_n b + u_{n-1}}{v_{n-1}} \right\rfloor$, onda vrijedi

$$\hat{q} > \frac{u_n b + u_{n-1}}{v_{n-1}} - 1.$$

Pomnožimo nejednadžbu s v_{n-1}

$$\hat{q} v_{n-1} > u_n b + u_{n-1} - v_{n-1}.$$

Kako radimo s cijelim brojevima, slijedi

$$\hat{q} v_{n-1} \geq u_n b + u_{n-1} - v_{n-1} + 1.$$

Sada imamo

$$\begin{aligned}
u - \hat{q}v &\leq \left(\text{jer je } \hat{q}v \geq \hat{q} v_{n-1} b^{n-1}\right) \leq u - \hat{q} v_{n-1} b^{n-1} \\
&\leq u_n b^n + \cdots + u_0 - (u_n b + u_{n-1} - v_{n-1} + 1) b^{n-1} \\
&\leq u_n b^n + \cdots + u_0 - u_n b^n - u_{n-1} b^{n-1} + v_{n-1} b^{n-1} - b^{n-1} \\
&= u_{n-2} b^{n-2} + \cdots + u_0 + v_{n-1} b^{n-1} - b^{n-1} \\
&< (\text{zbog } b^{n-1} > u_{n-2} b^{n-2} + \cdots + u_0) \\
&< v_{n-1} b^{n-1} \leq v.
\end{aligned}$$

Dobili smo da je $u - \hat{q}v < v$. Nakon dijeljenja s v dobivamo

$$\begin{aligned}
\frac{u}{v} - \hat{q} &< 1 \\
q \leq \frac{u}{v} &< \hat{q} + 1 \\
q &\leq \hat{q}.
\end{aligned}$$

Dokažimo sada prvu nejednakost, odnosno da $\hat{q}$ nije veće od $q + 2$. Pretpostavimo suprotno, odnosno da vrijedi $\hat{q} \geq q + 3$. Slijedi

$$\hat{q} \leq \frac{u_n b + u_{n-1}}{v_{n-1}} = \frac{u_n b^n + u_{n-1} b^{n-1}}{v_{n-1} b^{n-1}} \leq \frac{u}{v_{n-1} b^{n-1}} < \frac{u}{v - b^{n-1}}.$$

Znamo da $v \neq b^{n-1}$, tj. $v \neq (10 \ldots 0)_b$, jer je u tom slučaju $q = \hat{q}$. Onda je

$$\begin{aligned} 3 \leq \hat{q} - q &< \left(\text{jer je } q > \frac{u}{v} - 1\right) < \frac{u}{v - b^{n-1}} - \frac{u}{v} + 1 \\ &= \frac{uv - u(v - b^{n-1})}{v(v - b^{n-1})} + 1 = \frac{u}{v}\left(\frac{v - v + b^{n-1}}{v - b^{n-1}}\right) + 1 \\ &= \frac{u}{v}\left(\frac{b^{n-1}}{v - b^{n-1}}\right) + 1. \end{aligned}$$

Sada slijedi

$$\begin{aligned} 2 &< \frac{u}{v}\left(\frac{b^{n-1}}{v - b^{n-1}}\right) \\ \frac{u}{v} &> 2\left(\frac{v - b^{n-1}}{b^{n-1}}\right) = 2\left(\frac{v}{b_{n-1}} - 1\right) \geq 2(v_{n-1} - 1). \end{aligned}$$

Kako je $b - 1 \geq \hat{q}$, onda je $b - 4 \geq \hat{q} - 3 \geq q = \left\lfloor \frac{u}{v} \right\rfloor \geq 2(v_{n-1} - 1)$, pa imamo, redom,

$$\begin{aligned} b - 4 &\geq 2v_{n-1} - 2 \\ b - 2 &\geq 2v_{n-1} \\ \frac{b}{2} - 1 &\geq v_{n-1} \\ \left\lfloor \frac{b}{2} \right\rfloor &> v_{n-1}. \end{aligned}$$

Vidimo da u slučaju kada je $\hat{q} \geq q + 3$, onda je $v_{n-1} < \left\lfloor \frac{b}{2} \right\rfloor$. Iz toga možemo zaključiti da je $\hat{q} - 2 \leq q$, ako je $v_{n-1} \geq \left\lfloor \frac{b}{2} \right\rfloor$. Time smo dokazali i prvu nejednakost teorema. □

Definicija 1.3.4. *Divizor v koji zadovoljava uvjet* $v_{n-1} \geq \left\lfloor \frac{b}{2} \right\rfloor$ *zovemo* ***normalizirani divizor****.*

Algoritam za nalaženje jednoznamenkastog kvocijenta radi na način da prvo nađe približnu vrijednost kvocijenta $\hat{q}$. Nakon toga, ukoliko je $u \geq \hat{q}v$ i ostatak $\hat{r}$ (dobiven pomoću $\hat{q}$) veći od v, korigiramo $\hat{q}$ i $\hat{r}$ tako da $\hat{q}$ povećavamo za 1, a $\hat{r}$ smanjujemo za v, sve dok

ostatak nije manji od v. Ukoliko je $u < \hat{q}v$, smanjujemo $\hat{q}$ za 1, sve dok vrijedi nejednakost. Na kraju je traženi $q = \hat{q}$ i $r = \hat{r}$. Složenost ovog algoritma ovisi o tome koliko je dobra procjena kvocijenta $\hat{q}$. Teorem 1.3.3 dokazuje da, za normalizirani divizor, u algoritmu nećemo imati više od 2 oduzimanja.

Sada možemo definirati algoritam za dijeljenje prirodnih brojeva s ostatkom, u kojem ćemo procjenu za kvocijent računati na način opisan gore. Prije dijeljenja brojeva u i v napravit ćemo normalizaciju divizora, tako da brojeve u i v pomnožimo s odgovarajućim faktorom d. Cilj normalizacije divizora je što manja složenost algoritma, odnosno što bolja procjena kvocijenta q. Normalizacija ne smije promijeniti kvocijent, a ostatak r koji dobijemo, ćemo u zadnjem koraku podijeliti s d. Teorem 1.3.5 nam govori da je faktor d, koji tražimo, oblika $d = \left\lfloor \frac{b}{v_m+1} \right\rfloor$. Dokaz tog teorema možete naći u [7].

Teorem 1.3.5. *Neka je $v \in \mathbb{N}$ bilo koji broj, dan u pozicijskom zapisu $v = (v_m \dots v_0)_b$ u bazi b. Ako definiramo*

$$d_1 = \left\lfloor \frac{b}{v_m + 1} \right\rfloor \quad i \quad v' = d_1 \cdot v,$$

onda je $\deg_b(v') = \deg_b(v) = m$ *i za vodeću znamenku broja v' vrijedi*

$$\left\lfloor \frac{b}{2} \right\rfloor \leq v'_m \leq b - 1.$$

Algoritam kao ulaz prima dva broja $u = (u_n \dots u_0)_b$ i $v = (v_m \dots v_0)_b$ zapisana u bazi b. Algoritam završava s greškom ukoliko je $v = 0$, zato pretpostavljamo da je $v > 0$.

Algoritam 3: Dijeljenje prirodnih brojeva s ostatkom (*DIV*)

Rezultat: $q = (q_k \dots q_1 q_0)_b$, $r = (r_l \dots r_1 r_0)_b$, gdje je $q = \left\lfloor \frac{u}{v} \right\rfloor$, a $r = u \bmod v$.

if $m \geq 0$ **then**
  if $n \geq m$ **then**
    $v_{poc} = v_m + 1$;
    if $v_m < b \div 2$ **then**
      $d = b \div v_{poc}$;
      $(v_m \dots v_0) = d \cdot (v_m \dots v_0)$;
      $(u_{n+1} u_n \dots u_0) = d \cdot (u_{n+1} u_n \dots u_0)$;
    else
      $u_{n+1} = 0$;
    $k = n - m$;
    for $i \leftarrow k$ **to** 0 **by** -1 **do**
      $nu = m + i$;
        $\{u' = (u_{i+m+1} u_{i+m} \dots u_i)\}$
      $q_i = (u_{nu+1} \cdot b + u_{nu}) \div v_{poc}$;
        $\{r = u' - q_i \cdot v, \quad r = (u_{i+m+1} u_{i+m} \dots u_i)\}$
      if $q_i > 0$ **then**
        $(u_{i+m+1} \dots u_i) = (u_{i+m+1} \dots u_i) - q_i \cdot (v_m \dots v_0)$;
      while $(u_{m+i+1} \dots u_i) \geq (v_m \dots v_0)$ **do**
        $q_i = q_i - 1$;
        $(u_{i+m+1} \dots u_i) = (u_{i+m+1} \dots u_i) - (v_m \dots v_0)$;
    if $q_k = 0$ **then**
      $k = k - 1$;
    if $d > 1$ **then**
      $(r_m \dots r_0) = (u_{m+1} \dots u_0)/d$;
    else
      $(r_m \dots r_0) = (u_m \dots u_0)$;
    $l = m$;
    while $(r_l = 0) \wedge (l > 0)$ **do**
      $l = l - 1$;
    if $r_l = 0$ **then**
      $l = -1$;
  else
    $k = -1$;
    $l = m$;
    $(r_m \dots r_0) = (v_m \dots v_0)$;
else
  ERROR;

Poglavlje 2

Složenost klasičnih algoritama za množenje i dijeljenje

Za jednu zadaću može postojati više algoritama koji daju rješenje. Potreban nam je način na koji možemo uspoređivati algoritme, kako bismo odlučili koji je bolji. Složenost algoritama nam govori o tome koliko resursa algoritam troši za računanje. Ona ovisi o veličini zadaće koju algoritam rješava pa ćemo, iz tog razloga, složenost izražavati kao funkciju veličine zadaće. Mjera veličine zadaće za aritmetičke algoritme je duljina ulaznih brojeva. Prvo ćemo definirati potrebne pojmove kako bismo precizno mogli navesti složenosti algoritama.

2.1 Duljina broja

Definicija 2.1.1. *Duljina broja u u bazi b, u oznaci $\ell_b(u)$, je broj znamenki u pozicijskom zapisu broja u u bazi b*

$$\ell_b(u) = \deg_b(u) + 1 = \lfloor \log_b u + 1 \rfloor.$$

Možemo primijetiti da su duljina i stupanj broja povezani. Posebno ćemo definirati prikaz broja nula u bazi b i njezinu duljinu, tako da veza među duljinom i stupnjem ostane vrijediti.

Definicija 2.1.2. *Broj $0 \in \mathbb{N}_0$ prikazujemo praznim nizom znamenki $0 = (\,)_b$ za bilo koju bazu $b \geq 2$. Duljina broja 0 je prirodno definirana s $\ell_b(0) = 0$, a najvišu potenciju baze (stupanj) definiramo s $\deg_b(0) = -1$.*

Kako u ovom poglavlju govorimo o složenosti algoritama, uvest ćemo i sljedeće oznake.

Definicija 2.1.3. *Neka su $f, g : \mathcal{D} \to \mathbb{R}$ dvije funkcije na odozgo neograničenom podskupu $\mathcal{D} \subseteq \mathbb{R}$.*

1. *f je istog reda veličine kao i g, ili f raste istom brzinom kao g, u oznaci*

$$f(x) \in \Theta(g(x)) \quad za\ (x \to \infty),$$

ako $\exists c_1, c_2 \in \mathbb{R}$, $c_1, c_2 > 0$ i $\exists x_0 \in \mathcal{D}$ takvi da je

$$\forall x > x_0, \quad c_1|g(x)| < |f(x)| < c_2|g(x)|.$$

2. *f i g su asimptotski jednake (konstante rasta su iste), u oznaci*

$$f(x) \sim g(x) \quad za\ (x \to \infty),$$

ako postoji $\lim\limits_{x\to\infty} \dfrac{f(x)}{g(x)}$ i vrijedi

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1.$$

Definicija 2.1.4. *Ako su f i g nenegativne funkcije i ako je*

$$f(x) \in \Theta(g(x))$$

onda kažemo da su f i g kodominantne.

S obzirom da je duljina broja mjera veličine zadaće za algoritme koje promatramo, želimo pokazati da baza u kojoj prikazujemo ulazne brojeve nema znatni utjecaj na složenost tih algoritama. Sljedeći teorem, preuzet iz [7], to i dokazuje.

Teorem 2.1.5. *Duljine prirodnih brojeva u bilo koje dvije baze su kodominantne. Vrijedi i jače, ako su $b_1, b_2 \geq 2$ proizvoljne dvije baze, onda je*

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

tj. duljine brojeva su asimptotski proporcionalne za velike brojeve.

Dokaz. Iz definicije duljine prirodnog broja $\ell_b(u)$ vidimo da se radi o nenegativnoj rastućoj funkciji za bilo koju bazu b. Omjer duljina istog broja u u različitim bazama b_1 i b_2 definiramo kao sljedeću funkciju

$$f(u) = \frac{\ell_{b_2}(u)}{\ell_{b_1}(u)} = \frac{\deg_{b_2}(u)}{\deg_{b_1}(u)} = \frac{\lfloor \log_{b_2} u + 1 \rfloor}{\lfloor \log_{b_1} u + 1 \rfloor}.$$

Za $u \geq b_1 > 1$ vrijedi $\log_{b_1} u \neq 0$. Koristeći relaciju $x - 1 < \lfloor x \rfloor \leq x, \forall x \in \mathbb{R}$, dobivamo

$$f(u) < \frac{\log_{b_2} u + 1}{\log_{b_1} u} = \log_{b_2} b_1 + \frac{1}{\log_{b_1} u}. \tag{2.1}$$

Iz $u \geq b_1$ slijedi $\log_{b_1} u \geq 1$, pa je

$$f(u) < \log_{b_2} b_1 + 1. \tag{2.2}$$

Relacija (2.2) vrijedi i za $u < b_1$, jer je u tom slučaju $\ell_{b_1}(u) = 1$, pa je

$$f(u) = \ell_{b_2}(u) \leq \log_{b_2} u + 1 < \log_{b_2} b_1 + 1.$$

S druge strane imamo

$$f(u) > \frac{\log_{b_2} u}{\log_{b_1} u + 1} = \log_{b_2} b_1 \cdot \frac{\log_{b_1} u}{\log_{b_1} u + 1}. \tag{2.3}$$

Za $u \geq b_1$ je

$$\frac{\log_{b_1} u}{\log_{b_1} u + 1} \geq \frac{1}{2},$$

što s (2.2) daje

$$\frac{1}{2} \log_{b_2} b_1 < f(u) < \log_{b_2} b_1 + 1,$$

za $u \geq b_1$. To znači da je

$$\ell_{b_2}(u) = \Theta(\ell_{b_1}(u)),$$

pa su $\ell_{b_1}(u)$ i $\ell_{b_2}(u)$ kodominantne. Iz relacija (2.1) i (2.3) dobivamo

$$\log_{b_2} b_1 \cdot \frac{\log_{b_1} u}{\log_{b_1} u + 1} < f(u) < \log_{b_2} b_1 + \frac{1}{\log_{b_1} u}.$$

Zbog $\log_{b_1} u \to \infty$ kada $u \to \infty$, izlazi

$$\lim_{u \to \infty} f(u) = \log_{b_2} b_1,$$

što dokazuje tvrdnju teorema. □

2.2 Složenosti algoritama

Složenosti koje promatramo su vremenska, prostorna i aritmetička složenost. Prostorna složenost nam govori koliko memorije je potrebno algoritmu za spremanje vrijednosti varijabli s kojima radi, bez da navodimo jedinice u kojima mjerimo memoriju. Aritmetička složenost nam daje broj elementarnih operacija zbrajanja, množenja i dijeljenja, koji ovise o ulaznim podacima. Vremensku složenost izražavamo redom veličine, bez navođenja jedinica u kojima ju mjerimo. Mjerimo ju na način da odaberemo elementarne operacije te njima dodijelimo neku osnovnu složenost. Razlog za to je što neka računala mogu računati brže od drugih, a nama je cilj uspoređivati algoritme, a ne računala koja ih izvode. Također, dovoljna nam je informacija o vodećem, tj. dominantnom članu funkcije koji utječe na njezino ponašanje. Za algoritme koje smo opisali ranije, navest ćemo navedene složenosti.

Klasični algoritam za množenje *MUL* ima prostornu složenost

$$\mathrm{Compl}_S(MUL) = 2(\ell(u) + \ell(v)) + c.$$

Naime, treba nam prostor za $\ell(u)$ znamenki da spremimo broj u, $\ell(v)$ znamenki da spremimo broj v, $\ell(u)+\ell(v)$ znamenki da spremimo rezultat, te c mjesta za spremanje pomoćnih varijabli. Aritmetička složenost, odnosno broj elementarnih operacija koje algoritam izvede je

$$\mathrm{Compl}_A(MUL) = \begin{cases} 2\ell(u)\cdot\ell(v) + 2 & \text{zbrajanja,} \\ \ell(u)\cdot\ell(v) & \text{množenja,} \\ 2\ell(u)\cdot\ell(v) & \text{dijeljenja,} \end{cases}$$

što je sveukupno $\mathrm{Compl}_A(MUL) = 5\ell(u)\cdot\ell(v) + 2$. Iz ovoga slijedi da je vremenska složenost klasičnog algoritma za množenje prirodnih brojeva jednaka

$$\mathrm{Compl}_T(MUL) \sim \ell(u)\cdot\ell(v).$$

Algoritam dijeljenja potencijom baze *DIVB* ima prostornu složenost

$$\mathrm{Compl}_S(DIVB) = \ell(u) + \ell(r) + \ell(q) + c,$$

gdje je c konstanta koja označava broj mjesta potrebnih za spremanje pomoćnih varijabli. Aritmetička složenost algoritma je konstanta, jer samo jednom oduzimamo stupnjeve pa vrijedi

$$\mathrm{Compl}_A(DIVB) = 1.$$

Vremenska složenost u najgorem slučaju je

$$\mathrm{Compl}_T(DIVB) \sim \ell(u).$$

Prostorna složenost klasičnog algoritma za dijeljenje prirodnih brojeva s ostatkom *DIV* je

$$\mathrm{Compl}_S(DIV) = 2 \cdot \ell(u) + \ell(v) + c,$$

jer nam treba prostor za spremanje znamenki ulaznih brojeva u i v. Zatim smo uračunali prostor od $\ell(u)-\ell(v)+1$ znamenki za spremanje kvocijenta q te $\ell(v)$ znamenki za spremanje ostatka r. Konstanta c služi za količinu mjesta potrebnu za spremanje pomoćnih varijabli. Aritmetičku složenost možemo promatrati kao zbroj aritmetičkih složenosti manjih faza algoritma

$$\begin{aligned}\mathrm{Compl}_A(DIV) = {} & \mathrm{Compl}_A(normalizacija) \\ & + \mathrm{Compl}_A(dijeljenje) \\ & + \mathrm{Compl}_A(ostatak).\end{aligned}$$

Ukoliko normalizacija nije potrebna, onda su

$$\mathrm{Compl}_A(normalizacija) = \mathrm{Compl}_A(ostatak) = 0.$$

Inače je

$$\mathrm{Compl}_A(normalizacija) = \begin{cases} \ell(v) - 1 & \text{zbrajanja,} \\ \ell(v) & \text{množenja,} \\ 2\ell(v) - 1 & \text{dijeljenja,} \end{cases}$$

što je ukupno $\mathrm{Compl}_A(normalizacija) = 4\ell(v) - 2$. Za računanje ostatka vrijedi

$$\mathrm{Compl}_A(ostatak) = \begin{cases} \ell(v) - 1 & \text{zbrajanja,} \\ \ell(v) - 1 & \text{množenja,} \\ 2\ell(v) - 1 & \text{dijeljenja,} \end{cases}$$

što je ukupno $\mathrm{Compl}_A(ostatak) = 4\ell(v) - 3$. Aritmetička složenost faze dijeljenja je

$$\mathrm{Compl}_A(dijeljenje) = \sum_{i=0}^{\deg(q)} (\mathrm{Compl}_A(q_i) + \mathrm{Compl}_A(r) + \mathrm{Compl}_A(korekcija)),$$

uz oznake q_i = nalaženje procjene q_i, r = nalaženje odgovarajućeg ostatka. Za nalaženje procjene imamo po jedno zbrajanje, množenje i dijeljenje pa vrijedi

$$\mathrm{Compl}_A(q_i) = 3.$$

Za ostatak imamo

$$\mathrm{Compl}_A(r) = \begin{cases} 4\ell(v) & \text{zbrajanja,} \\ \ell(v) & \text{množenja,} \\ 2\ell(v) & \text{dijeljenja,} \end{cases}$$

odnosno $\mathrm{Compl}_A(r) = 7\ell(v)$. Za korekciju kvocijenta i ostatka imamo

$$\mathrm{Compl}_A(\textit{korekcija}) = 3\ell(v) + 1 \text{ zbrajanja.}$$

Kako je $\ell(q) \leq \ell(u) - \ell(v) + 1$, slijedi

$$\mathit{Compl}_A(\textit{dijeljenje}) = (\ell(u) - \ell(v) + 1) \cdot \begin{cases} 10\ell(v) + 5 & \text{zbrajanja,} \\ \ell(v) & \text{množenja,} \\ 2\ell(v) & \text{dijeljenja,} \end{cases}$$

što je ukupno $\mathrm{Compl}_A(\textit{dijeljenje}) = (\ell(u) - \ell(v) + 1) \cdot (13\ell(v) + 5)$. Konačno, za klasični algoritam dijeljenja s ostatkom vrijedi

$$\mathrm{Compl}_A(\mathit{DIV}) = (\ell(u) - \ell(v) + 1)(13\ell(v) + 5) + 8\ell(v) - 4,$$

jer smo dodali još jedno zbrajanje za v_{poc}. Za vremensku složenost vrijedi

$$\mathrm{Compl}_T(\mathit{DIV}) \sim (\ell(u) - \ell(v) + 1) \cdot \ell(v).$$

Klasični algoritmi za množenje, odnosno dijeljenje prirodnih brojeva su vrlo dobri za brojeve relativno male duljine. Za brojeve koji imaju veću duljinu postoje drugi, puno brži algoritmi za množenje, odnosno dijeljenje brojeva. U nastavku ovog rada obradit ćemo neke od njih.

Poglavlje 3

Asimptotski brzi algoritmi za množenje i dijeljenje

Ne postoji točno jedan algoritam za množenje, odnosno dijeljenje brojeva za koji možemo tvrditi da je najbolji i najbrži. Postoje razni algoritmi za izvođenje aritmetičkih operacija, od kojih su neki bolji za manje veličine zadaće, a neki za jako velike. Ovisno o veličini zadaće treba znati odabrati pravi algoritam. Klasični algoritmi za množenje, odnosno dijeljenje prirodnih brojeva efikasni su za male duljine ulaznih podataka. U ovom poglavlju navest ćemo algoritme koji su asimptotski brži od algoritama koje smo do sada naveli.

3.1 Računanje recipročne vrijednosti prirodnog broja

Asimptotski brzo dijeljenje prirodnih brojeva u i v možemo izvršiti tako da prvo izračunamo recipročnu vrijednost djelitelja $1/v$, a zatim, nekim asimptotski brzim algoritmom za množenje brojeva, pomnožimo u i $1/v$. Time ćemo dobiti traženi kvocijent u/v. Kako recipročna vrijednost broja v više ne mora biti prirodan broj, tražimo dovoljno točnu aproksimaciju $\hat{v}$ za v^{-1}.

Dovoljno točna aproksimacija znači da zadamo traženu točnost koju će recipročna vrijednost $\hat{v}$ zadovoljiti, kako bi krajnji kvocijent $\hat{q}$, također, bio dovoljno blizu pravog kvocijenta q. Ukoliko zadamo točnost $|\epsilon| \leq b^{-n}$, gdje je b odabrana baza $b \geq 2$, a $n = \ell_b(u)$, imamo

$$\frac{1}{v} = \hat{v} + \epsilon,$$

pa je onda

$$\left| \frac{u}{v} - u \cdot \hat{v} \right| \leq u|\epsilon| \leq 1.$$

Sada stavimo

$$\hat{q} = \lfloor u \cdot \hat{v} \rfloor$$

pa vrijedi

$$|q - \hat{q}| \leq 1.$$

Očito je tražena recipročna vrijednost v^{-1} rješenje jednadžbe

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

Do njezinog rješenja doći ćemo Newtonovom metodom, koja je definirana u [8]. Metodu još nazivamo i Metoda tangente, jer je ideja metode povući tangentu na graf funkcije u točki $(x_0, f(x_0))$, za neku početnu točku x_0. Novu aproksimaciju za nultočku definiramo u točki x_1 gdje povučena tangenta siječe os x. Slijedi geometrijski izvod. Znamo da je jednadžba tangente u točki x_n dana formulom

$$y - f(x_n) = f'(x_n)(x - x_n),$$

iz čega izlazi da je nova aproksimacija x_{n+1} za nultočku dana s

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3.1}$$

Neka je v_i trenutna aproksimacija za v^{-1}. Uvrštavajući našu funkciju f i oznake u (3.1), dobijemo da je iduća aproksimacija v_{i+1} jednaka

$$v_{i+1} = v_i \cdot (2 - v \cdot v_i).$$

Ukoliko s $e_{(i)}$ označimo pogrešku $i-$te aproksimacije za traženu recipročnu vrijednost $1/v$

$$e_{(i)} = \frac{1}{v} - v_i, \quad i \geq 0,$$

uvrštavanjem u formulu za iduću aproksimaciju slijedi da je

$$e_{(i+1)} = v \cdot e_{(i)}^2.$$

Ako postavimo da je $|e_{(i)}| \leq b^{-(m+d)}$, gdje je $m = \ell_b(v)$, tada za $e_{(i+1)}$ izlazi

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

Kako je $b^{m-1} \leq v < b^m$, za v^{-1} vrijedi

$$\frac{1}{b^m} < \frac{1}{v} \leq \frac{1}{b^{m-1}},$$

pa zaključujemo da se broj točnih znamenki udvostručuje svakim korakom, ako se operacije za računanje aproksimacija izvode egzaktno.

Pozicijski prikaz realnog broja $a = (a_k \cdots a_1 a_0 . a_{-1} a_{-2} \cdots)_b$ u bazi b ima sljedeći zapis

$$a = \sum_{i=1}^{k} a_i \cdot b^i + a_0 + \sum_{i=-1}^{-\infty} a_i \cdot b^i.$$

Pozicijski prikaz broja $1/v$ u bazi b prikazat ćemo na način da započne sa znamenkom 0 i da prva znamenka iza decimalne točke bude različita od 0. Razlog za to je što želimo pojednostaviti zapis, tako da izbjegnemo pisanje početnih 0 iza decimalne točke, s obzirom da ih ima puno za velike brojeve v.

$$\frac{1}{v} = b^{-(m-1)} \cdot \sum_{i=0}^{+\infty} v_i' b^{-i}, \quad \text{gdje je} \quad 0 \leq v_i' < b, \quad i \in \mathbb{N}_0,$$

i vrijedi

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

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

Sada ćemo opisati način na koji radi algoritam za nalaženje recipročne vrijednosti prirodnog broja. Neka je zadana točnost p takva da za aproksimaciju $\hat{v}$ broja $1/v$ vrijedi

$$b^{m-1+p} \cdot \hat{v} = \left\lfloor \frac{b^{m-1+p}}{v} \right\rfloor = \sum_{i=0}^{p} v_i' b^{p-i} = \sum_{i=0}^{p} \hat{v}_i b^i,$$

odnosno prvih p znamenaka iza decimalne točke u zapisu broja $1/v$ smo točno odredili, s time da smo napravili pomak znamenki ulijevo kako bismo radili s prirodnim brojevima. Zbog toga, operacije kod računanja iteracija u Newtonovoj metodi možemo računati cjelobrojnom aritmetikom. Preciznost p duplirat ćemo u svakoj iteraciji, pa ćemo u iteracijama koristiti samo dio vodećih znamenki broja v. Neka je baza u kojoj prikazujemo brojeve $b = 2$, te neka vrijedi $\ell_2(v) = m = 2^l$, odnosno broj v ima parnu duljinu. To znači da ćemo u $k-$toj iteraciji koristiti preciznost $p_k = 2^k$ i promatrat ćemo vodećih p_k znamenki ulaznog broja v.

Algoritam 4: Recipročni prirodni broj (*REC*)

Rezultat: Broj $\hat{v}$ takav da $\hat{v} = \lfloor 2^{2m-1}/v \rfloor$, gdje je $\ell(\hat{v}) \in \{m, m+1\}$.

$m = \deg(v) + 1$;
$(\hat{v}_1, \hat{v}_0) = (1, 0)$;
$p = 1$;
while $p < m$ **do**
  $\{p = 2^k\}$
  $p = 2 \cdot p$;
  $\{\hat{v} = \lfloor 2^{2p-1}/\lfloor v/2^{m-p} \rfloor \rfloor\}$
  $(t_{2p-1} \cdots t_0) = (\hat{v}_{p/2} \cdots \hat{v}_0) \cdot 2^{3 \cdot p/2} - (\hat{v}_{p/2} \cdots \hat{v}_0)^2 \cdot (v_{m-1} \cdots v_{m-p})$;
  $(\hat{v}_p \cdots \hat{v}_0) = (t_{2p-1} \cdots t_{p-1})$;
  for $i \leftarrow 2$ **to** 0 **by** -1 **do**
    if $((\hat{v}_p \cdots \hat{v}_0) + 2^i) \cdot (v_{m-1} \cdots v_{m-p}) \leq 2^{2p-1}$ **then**
      $(\hat{v}_p \cdots \hat{v}_0) = (\hat{v}_p \cdots \hat{v}_0) + 2^i$;
if $\hat{v}_m = 0$ **then**
  $\deg(\hat{v}) = \deg(v)$;
else
  $\deg(\hat{v}) = \deg(v) + 1$;

Sada ćemo navesti iskaz i dokaz teorema iz [7], koji potvrđuje da algoritam nalazi zadovoljavajući recipročni broj.

Teorem 3.1.1. *Algoritam REC za traženje recipročnog prirodnog broja nalazi broj* $\hat{v} \in \mathbb{N}$ *takav da je*

$$\hat{v} = \lfloor 2^{2m-1}/v \rfloor, \tag{3.2}$$

tj. vrijedi

$$v \cdot \hat{v} = 2^{2m-1} - r, \quad 0 \leq r < v.$$

Dokaz. Dokazat ćemo da algoritam nalazi niz brojeva

$$\hat{v}_{[k]} = \left\lfloor \frac{2^{2p-1}}{\lfloor v/2^{m-p} \rfloor} \right\rfloor, \quad k = 0, \ldots, l, \tag{3.3}$$

gdje je $\hat{v} = \hat{v}_{[0]}$ na samom početku algoritma i $\hat{v} = \hat{v}_{[k]}$ na dnu petlje u kojoj je $p = 2^k$, za $k = 1, \ldots, l$.

Kako je $m = 2^l$ i zbog toga što algoritam vraća $\hat{v} = \hat{v}_{[l]}$, uvrštavanjem u (3.3) direktno dobijemo da vrijedi (3.2).

Radi preglednosti, u nastavku dokaza ispuštamo uglate zagrade u indeksu, uz dogovor da slovo k u indeksu označava broj, poput $\hat{v}_k$ ili v_k, a ne znamenku broja.

Indukcijom po k dokazat ćemo relaciju (3.3). Za $k = 0$ vrijedi da je $p = 1$. Onda je

$$\left\lfloor \frac{v}{2^{m-1}} \right\rfloor = v_{m-1} = 1,$$

jer je $v_{m-1} > 0$, a baza u kojoj prikazujemo brojeve je 2. Sada imamo $\hat{v}_0 = \left\lfloor \frac{2^{2-1}}{1} \right\rfloor = 2$. Algoritam inicijalizira $\hat{v} = (10)_2 = 2$ pa vrijedi (3.3).

Pretpostavimo da (3.3) vrijedi za neki $k < l$, te označimo $p = 2^{k+1}$. Označimo s

$$v_k = \lfloor v/2^{m-\frac{p}{2}} \rfloor = (v_{m-1} \cdots v_{m-\frac{p}{2}}),$$
$$v_{k+1} = \lfloor v/2^{m-p} \rfloor = (v_{m-1} \cdots v_{m-p}),$$

pa je $v_{k+1} = v_k \cdot 2^{\frac{p}{2}} + v'_k$, gdje smo s v'_k označili dodatne znamenke u v_{k+1}, odnosno

$$v'_k = (v_{m-\frac{p}{2}-1} \cdots v_{m-p}).$$

Po pretpostavci indukcije, na početku petlje s $p = 2^{k+1}$, vrijedi

$$\hat{v}_k = \left\lfloor \frac{2^{p-1}}{\lfloor v/2^{m-\frac{p}{2}} \rfloor} \right\rfloor,$$

odnosno

$$\hat{v}_k = \left\lfloor \frac{2^{p-1}}{v_k} \right\rfloor = (\hat{v}_{\frac{p}{2}} \cdots \hat{v}_0),$$

što znači da je

$$v_k \cdot \hat{v}_k = 2^{p-1} - r_k, \quad 0 \leq r_k < v_k. \tag{3.4}$$

Prva naredba u petlji računa broj

$$t = 2^{3 \cdot p/2} \cdot \hat{v}_k - v_{k+1} \cdot \hat{v}_k^2. \tag{3.5}$$

S obzirom da je vodeća znamenka $v_{m-1} = 1$, uvrštavanjem u oznaku za v_k, vidimo da je $v_k \geq 2^{\frac{p}{2}-1}$, pa je $\hat{v}_k \leq 2^{\frac{p}{2}}$, iz čega slijedi

$$t \leq 2^{3 \cdot p/2} \cdot 2^{p/2} - v_{k+1} \cdot \hat{v}_k^2 < 2^{2p}.$$

Dakle, algoritam egzaktno računa broj $t = (t_{2p-1} \cdots t_0)$.

Promatramo umnožak $t \cdot v_{k+1}$.

$$\begin{aligned} t \cdot v_{k+1} &= t \cdot (v_k \cdot 2^{\frac{p}{2}} + v'_k) = (\text{prema (3.5)}) = \\ &= v_k \hat{v}_k 2^{2p} + v'_k \hat{v}_k 2^{3 \cdot p/2} - (v_k \hat{v}_k 2^{p/2} + v'_k \hat{v}_k)^2. \end{aligned}$$

Uvrštavanjem relacije (3.4) dobijemo

$$t \cdot v_{k+1} = 2^{3p-2} - (2^{\frac{p}{2}} r_k - v'_k \hat{v}_k)^2.$$

Podijelimo dobiveno s 2^{p-1}, te označimo s $T = (2^{\frac{p}{2}} r_k - v'_k \hat{v}_k)^2 \cdot 2^{-(p-1)}$, pa imamo

$$2^{2p-1} = \frac{t v_{k+1}}{2^{p-1}} + T. \tag{3.6}$$

Kako je $v_k < 2^{\frac{p}{2}}$, onda je i $r_k < 2^{\frac{p}{2}}$. Vrijedi $\hat{v}_k < 2^{\frac{p}{2}}$ i $v'_k < 2^{\frac{p}{2}}$, pa je

$$|2^{p/2} r_k - v'_k \hat{v}_k| < 2^p,$$

iz čega slijedi

$$0 \leq T < 2^{p+1}.$$

Algoritam postavlja da je $\hat{v} = \lfloor t/2^{p-1} \rfloor$. Kako je $T \geq 0$, iz (3.6) slijedi

$$2^{2p-1} \geq \frac{t v_{k+1}}{2^{p-1}} \geq v_{k+1} \cdot \left\lfloor \frac{t}{2^{p-1}} \right\rfloor > v_{k+1} \cdot \left(\frac{t}{2^{p-1}} - 1 \right) = 2^{2p-1} - v_{k+1} - T.$$

Za brojeve na desnoj strani vrijedi $v_{k+1} < 2^p$ i $T < 2^{p+1}$, pa dobivamo ocjenu za $v_{k+1}\hat{v}$

$$2^{2p-1} \geq v_{k+1}\hat{v} > 2^{2p-1} - 2^{p+1} - 2^p.$$

To znači da je

$$v_{k+1} \cdot \hat{v} = 2^{2p-1} - r',$$

gdje je

$$0 \leq r' < 2^{p+1} + 2^p.$$

Kako je $v_{k+1} \geq 2^{p-1}$, slijedi

$$0 \leq r' < 6 v_{k+1}.$$

Iz ovoga slijedi da korekcijom

$$\hat{v}_{k+1} = \hat{v} + c,$$

koju algoritam izvršava na kraju, gdje je $c < 6$, možemo postići da je

$$v_{k+1} \cdot \hat{v}_{k+1} = 2^{2p-1} - r_{k+1}, \quad 0 \leq r_{k+1} < v_{k+1}.$$

Time smo dokazali tvrdnju teorema. □

Napomena 3.1.2. *U algoritmu smo pretpostavili da je $\ell(v) = m$ parne duljine, odnosno $m = 2^l$, i da je tražena preciznost $p = m$. U slučaju da duljina m nije potencija broja 2, onda nađemo $m_1 = 2^l$ takav da je*

$$\frac{m_1}{2} < m \leq m_1.$$

Zatim definiramo $v_1 = 2^{m_1 - m} \cdot v$ i nađemo aproksimaciju $\hat{v}_1$ za $1/v_1$ navedenim algoritmom. Broj

$$\hat{v} = \left\lfloor \frac{\hat{v}_1}{2^{m_1 - m}} \right\rfloor$$

je tražena aproksimacija za $1/v$ s preciznošću $p = m$.

Ukoliko je tražena točnost $p \neq m$, onda definiramo

$$v = \lfloor v \cdot 2^{m_2 - m} \rfloor,$$

gdje je m_2 potencija broja 2, takva da je

$$\frac{m_2}{2} < p \leq m_2.$$

Sada nađemo aproksimaciju $\hat{v}_2$ za $1/v_2$ navedenim algoritmom pa je broj

$$\hat{v} = \left\lfloor \frac{\hat{v}_2}{2^{m_2 - p}} \right\rfloor$$

tražena aproksimacija s preciznošću p.

Aritmetička složenost ovog algoritma, u oznaci $\mathrm{Rec}(m)$, ovisi o aritmetičkoj složenosti algoritma za množenje koji izaberemo, u oznaci $\mathrm{Mul}(m)$. Algoritam je rekurzivan pa, kada tražimo $\hat{v}_l$, moramo pronaći $\hat{v}_{l-1}$ te zadnji put proći kroz petlju s $p = m = 2^l$.

Kako bismo našli broj t, moramo kvadrirati broj $\hat{v}_{l-1}$, pomnožiti ga s v te napraviti pomake i zbrajanja, pa slijedi

$$\mathrm{Compl}_A(t) \leq \mathrm{Mul}\left(\frac{m}{2} + 1\right) + \mathrm{Mul}(m + 2) + c_1 m.$$

Pretpostavimo da za $\mathrm{Mul}(m)$ vrijedi

$$\mathrm{Mul}\left(\frac{m}{2}\right) \leq \frac{1}{2}\,\mathrm{Mul}(m), \quad \forall m \geq 2,$$

odnosno da $\mathrm{Mul}(m)$ raste barem linearno. Iz toga slijedi da je

$$\mathrm{Mul}\left(\frac{m}{2} + 1\right) \leq \frac{1}{2}\,\mathrm{Mul}(m + 2).$$

Kako je

$$\mathrm{Mul}(m+2) - \mathrm{Mul}(m) \in \Theta(m),$$

onda je

$$\mathrm{Compl}_A(t) \leq \frac{3}{2}\,\mathrm{Mul}(m) + c_2 m.$$

Broj $\hat{v}$ iz t dobijemo pomakom, a korekcija zahtijeva najviše 3 množenja, uz zbrajanje i uspoređivanje. Moguće je korekciju odraditi sa samo jednim množenjem, uz zbrajanje i oduzimanje. Zbog toga slijedi

$$\mathrm{Rec}(m) \leq \mathrm{Rec}\left(\frac{m}{2}\right) + \frac{5}{2}\,\mathrm{Mul}(m) + c'm.$$

Želimo pokazati da postoji konstanta c takva da vrijedi $\mathrm{Rec}(m) \leq c \cdot \mathrm{Mul}(m)$, gdje je $m = 2^l$ potencija broja 2. Izaberemo c, kao što je navedeno u [1], tako da je

$$c \geq \frac{2\,\mathrm{Rec}(1)}{\mathrm{Mul}(1)} \quad \text{i} \quad c \geq 5 + 2c'.$$

Dokaz provodimo indukcijom po eksponentu l. Tvrdnja očito vrijedi za bazu indukcije, $l = 1$. Pretpostavimo da tvrdnja vrijedi za $l - 1$, tj. za $m/2$. Sada je

$$\mathrm{Rec}(m) \leq c \cdot \mathrm{Mul}\left(\frac{m}{2}\right) + \frac{5}{2}\,\mathrm{Mul}(m) + c'm.$$

S obzirom da smo pretpostavili da za $\mathrm{Mul}(m)$ vrijedi

$$\mathrm{Mul}\left(\frac{m}{2}\right) \leq \frac{1}{2}\,\mathrm{Mul}(m), \quad \forall m \geq 2,$$

onda možemo pisati

$$\mathrm{Rec}(m) \leq \left(\frac{c}{2} + \frac{5}{2} + c'\right)\mathrm{Mul}(m).$$

Kako smo konstantu c izabrali tako da je $c \geq 5 + 2c'$, slijedi da je

$$\mathrm{Rec}(m) \leq c \cdot \mathrm{Mul}(m).$$

Time smo dokazali tvrdnju, odnosno

$$\mathrm{Rec}(m) \in \Theta(\mathrm{Mul}(m)).$$

Složenost cjelobrojnog dijeljenja linearno ovisi o složenosti algoritama za množenje koje koristimo, dakle složenost dijeljenja i množenja je podjednaka. Ukoliko u algoritmu za cjelobrojno dijeljenje koristimo asimptotski brze algoritme za množenje, imat ćemo i asimptotski brz algoritam za dijeljenje. U nastavku ćemo obraditi jedan takav algoritam za množenje prirodnih brojeva.

3.2 Karatsubin algoritam za množenje brojeva

Anatoly Karatsuba bio je ruski matematičar koji je 1963. godine objavio novi algoritam za množenje brojeva. Karatsubin algoritam temelji se na metodi "podijeli-pa-vladaj" te je asimtotski brži od klasičnog algoritma za množenje brojeva.

Algoritam prima dva prirodna broja u i v te računa njihov umnožak tako da prvo podijeli oba ulazna broja na dva jednaka dijela. Duljine ulaznih brojeva $\ell(u)$ i $\ell(v)$ trebaju biti parne i jednake, pa ukoliko nisu, dodajemo koliko je potrebno 0 na mjesta vodećih znamenki. Označimo duljine brojeva nakon dodanih 0 s n. Razdvajanjem brojeva u i v na dva jednaka dijela dobijemo

$$u = U_1 b^{\frac{n}{2}} + U_0 \quad \text{i} \quad v = V_1 b^{\frac{n}{2}} + V_0,$$

gdje je $b \geq 2$ proizvoljna baza u kojoj zapisujemo zadane brojeve. Umnožak $w = u \cdot v$ računamo formulom

$$w = u \cdot v = U_0 V_0 + (U_0 V_1 + U_1 V_0) \cdot b^{\frac{n}{2}} + U_1 V_1 \cdot b^n.$$

Vidimo da umjesto jednog umnoška od dva n-znamenkasta broja, imamo četiri umnoška od dva $\frac{n}{2}$-znamenkasta broja te nekoliko zbrajanja i pomaka. Ukoliko dio u zagradi zapišemo kao

$$U_0 V_1 + U_1 V_0 = (U_0 + U_1)(V_0 + V_1) - U_0 V_0 - U_1 V_1,$$

imat ćemo tri umnoška od dva $\frac{n}{2}$-znamenkasta broja, ili dva umnoška od dva $\frac{n}{2}$-znamenkasta broja i jedan umnožak od dva $(\frac{n}{2} + 1)$-znamenkasta broja, umjesto četiri. Algoritam se rekurzivno ponavlja sve dok $\frac{n}{2}$ nije dovoljno "malen" za klasično množenje.

Kako bismo dokazali da je aritmetička složenost Karatsubinog algoritma manja od aritmetičke složenosti klasičnog algoritma za množenje prirodnih brojeva, uvest ćemo sljedeće definicije i propozicije.

Definicija 3.2.1. *Neka je MUL bilo koji opći algoritam za množenje prirodnih brojeva $u, v \in \mathbb{N}$, u pozicijskom zapisu u nekoj bazi b. Aritmetičku složenost algoritma, u ovisnosti o duljinama brojeva u i v, gdje su $\ell(u) = m$ i $\ell(v) = n$, pišemo s*

$$\mathrm{Mul}(n, m) = \mathrm{Compl}_A(MUL).$$

Ako je $n = m$, pišemo skraćeno

$$\mathrm{Mul}(n) = \mathrm{Compl}_A(MUL).$$

Propozicija 3.2.2. *Neka je $t \in \mathbb{N}$ bilo koja konstanta. Za bilo koji algoritam množenja prirodnih brojeva u pozicijskom zapisu, postoji konstanta $C \in \mathbb{N}$, takva da za svaki $n \in \mathbb{N}$ vrijedi*

$$\mathrm{Mul}(n + t) \leq \mathrm{Mul}(n) + Cn,$$

gdje konstanta C ne ovisi o n.

Dokaz propozicije može se naći u [7]. Slijedi relacija za aritmetičku složenost Karatsubinog algoritma

$$\mathrm{Mul}(n) \leq 2\,\mathrm{Mul}\left(\frac{n}{2}\right) + \mathrm{Mul}\left(\frac{n}{2}+1\right) + c_1 n. \tag{3.7}$$

Primjenom propozicije 3.2.2 slijedi da postoji $c_2 \in \mathbb{N}$ takav da je

$$\mathrm{Mul}\left(\frac{n}{2}+1\right) \leq \mathrm{Mul}(n) + c_2 n.$$

Označimo s $c_0 = c_1 + c_2$, pa uvrštavanjem u (3.7) dobijemo

$$\mathrm{Mul}(n) \leq 3\,\mathrm{Mul}\left(\frac{n}{2}\right) + c_0 n.$$

Za $n = 1$ koristimo aritmetiku računala za množenje brojeva. U tom slučaju je $\mathrm{Mul}(1) = \hat{c}$, gdje je $\hat{c}$ neka konstanta neovisna o n. Radi jednostavnosti, uzmemo da je $c = \max\{c_0, \hat{c}\}$, pa dobivamo sljedeće rekurzivne relacije za aritmetičku složenost Karatsubinog algoritma, gdje je $n > 1$

$$\begin{aligned} &Mul(1) \leq c, \\ &Mul(n) \leq 3Mul\left(\frac{n}{2}\right) + cn. \end{aligned} \tag{3.8}$$

Definicija 3.2.3. *Neka je $f : \mathcal{D} \to \mathbb{R}_0^+$ nenegativna funkcija na odozgo neograničenom podskupu $\mathcal{D} \subseteq \mathbb{R}_0^+$. Funkcija f je asimptotski rastuća (neopadajuća) ako je f rastuća za dovoljno velike argumente, tj. $\exists M \in \mathcal{D}$ takav da $\forall x, y \in \mathcal{D}$ vrijedi*

$$x, y \geq M \; i \; x < y \implies f(x) \leq f(y).$$

Sljedeća propozicija, dostupna u [4], daje rješenje naših rekurzivnih relacija (3.8), odnosno daje aritmetičku složenost Karatsubinog algoritma.

Propozicija 3.2.4. *Neka je $T : \mathbb{N} \to \mathbb{R}^+$ asimptotski rastuća funkcija za koju vrijedi rekurzivna relacija*

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k, \quad za\ n > n_0,$$

gdje su $n_0 \geq 1$, $b \geq 2$, $k \geq 0$ cijeli brojevi, te $a, c > 0$ realni brojevi. Za red veličine od T vrijedi

$$T(n) \in \begin{cases} \Theta(n^k) & ako\ je\ a < b^k, \\ \Theta(n^k \log_b n) & ako\ je\ a = b^k, \\ \Theta(n^{\log_b a}) & ako\ je\ a > b^k. \end{cases}$$

Dokaz. Na prvom nivou rekurzije imamo jednu zadaću veličine n, koja zahtijeva $c \cdot n^k$ operacija. Na drugom nivou imamo a zadaća veličine $\frac{n}{b}$, od kojih svaka zahtijeva $c \cdot \left(\frac{n}{b}\right)^k$ operacija. Stoga, na drugom nivou imamo sveukupno $\left(\frac{a}{b^k} \cdot cn^k\right)$ operacija. Na sljedećem nivou imamo a^2 zadaća veličine $\frac{n}{b^2}$, koje troše $c \cdot \left(\frac{n}{b^2}\right)^k$ operacija, što ukupno iznosi $\left(\frac{a^2}{b^{2k}} \cdot cn^k\right)$ operacija. Analogno zaključujemo da u j-tom koraku rekurzije imamo ukupno $\left(\frac{a^j}{b^{kj}} \cdot cn^k\right)$ operacija. Imamo $\log_b n$ nivoa rekurzije pa za ukupni broj operacija na svim nivoima vrijedi

$$T(n) \leq \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^j \cdot cn^k = cn^k \cdot \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^j.$$

Primijetimo da je $\sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^j$ geometrijska suma s faktorom $r = \frac{a}{b^k}$. Ukoliko je $a = b^k$, slijedi da je $r = 1$, pa je

$$T(n) \leq cn^k \cdot \log_b n,$$

odnosno

$$T(n) \in \Theta(n^k \log_b n).$$

Inače, uvrštavajući formulu za geometrijsku sumu, imamo

$$T(n) \leq cn^k \cdot \frac{r^{\log_b n} - 1}{r - 1}.$$

Ako je $a < b^k$, onda je $r < 1$. S obzirom da promatramo asimptotski rastuću funkciju, tada za sve veće n, izraz $\left(\frac{a}{b^k}\right)^{\log_b n}$ teži u 0. Time dobijemo da je

$$T(n) \leq cn^k \cdot 1,$$

odnosno vrijedi da je

$$T(n) \in \Theta(n^k).$$

Inače, ako je $a > b^k$, tada vrijedi

$$T(n) \leq cn^k \cdot \frac{r^{\log_b n} - 1}{r - 1} \leq cn^k \cdot \frac{r^{\log_b n}}{r - 1} \leq \frac{c}{r - 1} \cdot n^k r^{\log_b n}.$$

Znamo da za $\forall x > 0$ i $\forall y > 0$ vrijedi $x^{\log y} = y^{\log x}$, stoga je

$$r^{\log_b n} = n^{\log_b r} = n^{\log_b (a/b^k)} = n^{\log_b a - k \log_b b} = n^{\log_b a - k}.$$

Sada imamo

$$T(n) \leq \left(\frac{c}{r - 1}\right) \cdot n^k n^{\log_b a} n^{-k} = \left(\frac{c}{r - 1}\right) \cdot n^{\log_b a},$$

odnosno

$$T(n) \in \Theta(n^{\log_b a}).$$

Time smo dokazali propoziciju. □

S obzirom da je funkcija aritmetičke složenosti rastuća funkcija, pa je samim time i asimptotski rastuća, tada na nju možemo primijeniti ovu propoziciju. U našem slučaju je $a = 3$, $b = 2$ te $k = 0$. Kako je $3 > 2^0 = 1$, po propoziciji 3.2.4 slijedi da je

$$\mathrm{Mul}(n) \in \Theta(n^{\log_2 3}).$$

Približno je $\log_2 3 = 1.58496$ pa vidimo da je aritmetička složenost Karatsubinog algoritma znatno manja od aritmetičke složenosti klasičnog algoritma za množenje prirodnih brojeva.

3.3 Modularna aritmetika

Modularna aritmetika je alternativni način na koji možemo izvoditi aritmetičke operacije nad velikim prirodnim brojevima. Ideja je imati nekoliko relativno prostih modula $m_1, \ldots, m_r$ te izvoditi aritmetičke operacije nad ostacima $u \pmod{m_1}, \ldots, u \pmod{m_r}$, umjesto direktno na broju u. Kineski teorem o ostacima nam omogućuje jedinstveni zapis broja u pomoću modula.

Teorem 3.3.1. *Neka su $m_1, \ldots, m_r$ prirodni brojevi, u parovima relativno prosti. Neka je $m = m_1 m_2 \ldots m_r$ i neka su $a, u_1, \ldots, u_r$ prirodni brojevi. Tada postoji jedinstveni prirodni broj u koji zadovoljava sljedeće*

$$a \leq u < a + m \quad i \quad u \equiv u_j \pmod{m_j}, \ \ za\ 1 \leq j \leq r.$$

Dokaz teorema možete pronaći u [5] te u [2], gdje autori jedan od dokaza nazivaju Newtonovom metodom za rješavanje Kineskog problema ostataka.

Prema ovom teoremu, svaki broj u iz navedenog raspona možemo jedinstveno prikazati nizom ostataka $(u_1 \ldots u_r)$. Modularna reprezentacija aritmetičkih operacija je sljedeća

$$\begin{aligned}
(u_1 \ldots u_r) + (v_1 \ldots v_r) &= ((u_1 + v_1) \pmod{m_1} \ldots (u_r + v_r) \pmod{m_r}), \\
(u_1 \ldots u_r) - (v_1 \ldots v_r) &= ((u_1 - v_1) \pmod{m_1} \ldots (u_r - v_r) \pmod{m_r}), \\
(u_1 \ldots u_r) \cdot (v_1 \ldots v_r) &= ((u_1 \cdot v_1) \pmod{m_1} \ldots (u_r \cdot v_r) \pmod{m_r}).
\end{aligned} \tag{3.9}$$

Raspon brojeva koje možemo koristiti u modularnoj aritmetici je $m = m_1 m_2 \cdots m_r$, odnosno produkt modula. Ukoliko radimo s binarnim brojevima, zgodno je module birati na način da je svaki m_j za jedan manji od neke potencije broja 2

$$m_j = 2^{e_j} - 1.$$

Brojevi ovog oblika nazivaju se Mersenneovi brojevi, a prosti brojevi tog oblika nazivaju se Mersenneovi prosti brojevi. Za Mersenneove brojeve $2^e - 1$ i $2^f - 1$ vrijedi da su relativno

prosti ako i samo ako su e i f relativno prosti. Ukoliko je 2^{32} veličina riječi na računalu, tada možemo izabrati $m_1 = 2^{32} - 1, m_2 = 2^{31} - 1, m_3 = 2^{29} - 1, m_4 = 2^{27} - 1, m_5 = 2^{25} - 1$, što omogućuje aritmetičke operacije zbrajanja, oduzimanja i množenja brojeva veličine do $m_1 m_2 m_3 m_4 m_5 > 2^{143}$. S obzirom da su moduli koje smo izabrali blizu veličine riječi u računalu, a aritmetičke operacije iz (3.9) radimo na ostacima u_i, gdje je $u_i < m_i$, za $i \in \{1, \dots r\}$, to znači da zbrajanje, oduzimanje i množenje možemo računati aritmetikom računala. Artimetička složenost zbrajanja, oduzimanja i množenja n-znamenkastih brojeva je, u tom slučaju, reda veličine n, što je znatno ubrzanje za množenje.

Kako bismo koristili modularnu aritmetiku, potrebni su nam i algoritmi konverzije modularnog prikaza broja u pozicijski prikaz i obrnuto. Također, bitno je da je složenost algoritama konverzije dovoljno mala, kako bismo imali poboljšanje u modularnoj verziji algoritma za množenje prirodnih brojeva, u odnosu na klasični algoritam množenja.

Opisat ćemo Newtonovu metodu za konverziju iz modularnog u pozicijski zapis. Koristit ćemo brojeve c_{ij}, $1 \leq i < j \leq r$, za koje vrijedi

$$c_{ij} m_i \equiv 1 \pmod{m_j}.$$

Konstante c_{ij} mogu se izračunati Euklidovim algoritmom za dane i i j, tako da vrijedi

$$c_{ij} m_i + c_{ji} m_j = 1. \tag{3.10}$$

Ako za module uzmemo Mersenneove brojeve, Knuth [5] pokazuje da je za Mersenneov broj $2^e - 1$, njegov inverz modulo $2^f - 1$ dan formulom

$$1 + 2^d + \cdots + 2^{(c-1)d},$$

gdje je $e \pmod{f} = d$ i $ce \pmod{f} = 1$.

Newtonova metoda je rekurzivna te u njoj rješenje $u^{(k)}$ nalazimo pomoću prethodno izračunatog rješenja

$$u^{(k-1)} \equiv u_j \pmod{m_j}, \quad \text{za } j = 1, 2, \dots, k-1, \tag{3.11}$$

počevši od $u^{(1)} = u_1$, a traženi broj u jednak je $u^{(r)}$. S a_k označimo sljedeću razliku brojeva

$$a_k = u_k - u^{(k-1)},$$

te $u^{(k)}$ izračunamo na sljedeći način

$$u^{(k)} = u^{(k-1)} + a_k \prod_{j=1}^{k-1} c_{jk} m_j. \tag{3.12}$$

Vidimo da vrijedi

$$\begin{aligned} u^{(k)} &\equiv u^{(k-1)} \pmod{m_j} \equiv (\text{zbog } (3.11)) \equiv \\ &\equiv u_j \pmod{m_j}, \quad \text{za } j = 1, 2, \ldots, k-1. \end{aligned} \tag{3.13}$$

Iz (3.12) slijedi

$$\begin{aligned} u^{(k)} &= u^{(k-1)} + a_k \prod_{j=1}^{k-1} c_{jk} m_j = (\text{zbog } (3.10)) = \\ &= u^{(k-1)} + a_k \prod_{j=1}^{k-1} (1 - c_{kj} m_k) \\ &\equiv u^{(k-1)} + a_k \cdot 1 \pmod{m_k} \\ &\equiv u^{(k-1)} + (u_k - u^{(k-1)}) \pmod{m_k} \\ &\equiv u_k \pmod{m_k}. \end{aligned} \tag{3.14}$$

Iz (3.13) i (3.14) sada slijedi da je

$$u^{(k)} \equiv u_j \pmod{m_j}, \quad \text{za } j = 1, 2, \ldots, k.$$

Članak [6] prikazuje Newtonovu metodu sljedećim trokutastim sustavom jednadžbi

$$\begin{aligned} d_1 &\equiv u_1 \pmod{m_1} \\ d_1 + d_2 m_1 &\equiv u_2 \pmod{m_2} \\ d_1 + d_2 m_1 + d_3 m_1 m_2 &\equiv u_3 \pmod{m_3} \\ &\vdots \\ d_1 + d_2 m_1 + d_3 m_1 m_2 + \cdots + d_r m_1 m_2 \cdots m_{r-1} &\equiv u_r \pmod{m_r}, \end{aligned}$$

iz kojeg možemo primijetiti da je

$$\begin{aligned} u^{(1)} &= d_1 \\ u^{(2)} &= d_1 + d_2 m_1 \\ u^{(3)} &= d_1 + d_2 m_1 + d_3 m_1 m_2 \\ &\vdots \\ u = u^{(r)} &= d_1 + d_2 m_1 + d_3 m_1 m_2 + \cdots + d_r m_1 m_2 \cdots m_{r-1}. \end{aligned}$$

Rješenje Newtonove interpolacije za Kineski problem ostataka je broj u sljedećeg oblika

$$u = u^{(r)} = \sum_{k=1}^{r} d_k q_k,$$

gdje su

$$u^{(k)} = u^{(k-1)} + d_k \cdot q_k,$$

$$q_k = \prod_{j=1}^{k-1} m_j, \quad d_k = (u_k - u^{(k-1)}) \prod_{j=1}^{k-1} c_{jk}, \quad d_1 = u^{(1)} = u_1.$$

Svaki d_k se u algoritmu računa kao navedeni izraz modulo m_k. S obzirom da se u algoritmu cijelo vrijeme koriste isti moduli, moguće je određene varijable, koje ovise samo o njima, unaprijed izračunati te ih dati algoritmu kao dodatne ulazne podatke. Te varijable su

$$q_k = \prod_{j=1}^{k-1} m_j, \quad \text{za } k = 2, 3, \ldots, r$$

i

$$c_k = \prod_{j=1}^{k-1} c_{jk} \pmod{m_k}, \quad \text{za } k = 2, 3, \ldots, r.$$

Konverzija broja iz binarnog pozicijskog zapisa u modularni je vrlo jednostavna kada su moduli oblika $2^{e_j} - 1$. Knuth [5] to radi na način da se binarna reprezentacija broja u razdvoji na blokove od e_j bitova. Neka tako sveukupno dobijemo $(t + 1)$ blokova. Tada u možemo zapisati kao

$$u = a_t A^t + a_{t-1} A^{t-1} + \cdots + a_1 A + a_0, \tag{3.15}$$

gdje je $A = 2^{e_j}$, a a_k je broj s binarnim zapisom iz k-og bloka, tako da je $0 \leq a_k < 2^{e_j}$, za $0 \leq k \leq t$. Tada vrijedi da je

$$u \equiv a_t + a_{t-1} + \cdots + a_1 + a_0 \pmod{2^{e_j} - 1}, \tag{3.16}$$

jer je $A = 2^{e_j} \equiv 1 \pmod{2^{e_j} - 1}$. Ostatke u_j onda dobijemo modularnim zbrajanjem.

Knuth u [5] navodi modularnu metodu za množenje prirodnih brojeva koju je opisao njemački matematičar Arnold Schönhage. Za početak, na sljedeći način definiramo niz brojeva

$$q_0 = 1, \quad q_{k+1} = 3q_k - 1,$$

odnosno $q_k = 3^k - 3^{k-1} - \cdots - 1 = \frac{1}{2}(3^k + 1)$. Množit ćemo p_k-bitne brojeve, gdje je $p_k = (18q_k + 8)$, koristeći metodu za množenje p_{k-1}-bitnih brojeva. Kako je $p_0 = 26$, slijedi da, ako znamo množiti 26-bitne brojeve, znat ćemo množiti i 44-bitne brojeve, itd. Koristi se šest međusobno relativno prostih modula oblika

$$m_1 = 2^{6q_k - 1} - 1, \quad m_2 = 2^{6q_k + 1} - 1, \quad m_3 = 2^{6q_k + 2} - 1$$
$$m_4 = 2^{6q_k + 3} - 1, \quad m_5 = 2^{6q_k + 5} - 1, \quad m_6 = 2^{6q_k + 7} - 1.$$

Navedeni moduli su relativno prosti i pomoću njih možemo prikazati brojeve veličine do $m = m_1m_2m_3m_4m_5m_6 > 2^{36q_k+16} = 2^{2(18q_k+8)} = 2^{2p_k}$. Zbog toga zaključujemo da ne može doći do overflowa kod množenja dva p_k-bitna broja u i v. Sada navodimo korake metode.

(1) Prvo brojeve u i v iz binarnog pretvorimo u modularni prikaz, odnosno izračunamo

$$\begin{aligned} u_1 &= u \pmod{m_1}, \quad & v_1 &= v \pmod{m_1}, \\ u_2 &= u \pmod{m_2}, \quad & v_2 &= v \pmod{m_2}, \\ u_3 &= u \pmod{m_3}, \quad & v_3 &= v \pmod{m_3}, \\ u_4 &= u \pmod{m_4}, \quad & v_4 &= v \pmod{m_4}, \\ u_5 &= u \pmod{m_5}, \quad & v_5 &= v \pmod{m_5}, \\ u_6 &= u \pmod{m_6}, \quad & v_6 &= v \pmod{m_6}. \end{aligned}$$

(2) Pomnožimo $u_1v_1, u_2v_2, \ldots, u_6v_6$. Ti brojevi imaju najviše $6q_k + 7 = 18q_{k-1} + 1 < p_{k-1}$ bitova, pa ih računamo već poznatom procedurom za množenje p_{k-1}-bitnih brojeva.

(3) Slično kao u (1), računamo

$$\begin{aligned} w_1 &= u_1v_1 \pmod{m_1}, \\ w_2 &= u_2v_2 \pmod{m_2}, \\ w_3 &= u_3v_3 \pmod{m_3}, \\ w_4 &= u_4v_4 \pmod{m_4}, \\ w_5 &= u_5v_5 \pmod{m_5}, \\ w_6 &= u_6v_6 \pmod{m_6}. \end{aligned}$$

(4) Računamo broj w, $0 \leq w < m$, za koji vrijedi

$$\begin{aligned} w_1 &= w \pmod{m_1}, \\ w_2 &= w \pmod{m_2}, \\ w_3 &= w \pmod{m_3}, \\ w_4 &= w \pmod{m_4}, \\ w_5 &= w \pmod{m_5}, \\ w_6 &= w \pmod{m_6}. \end{aligned}$$

Označimo s t_k vrijeme potrebno za računanje umnoška dva p_k-bitna broja. Koraci (1) i (3) se računaju kako je opisano u (3.15) i (3.16). Za to nam je potrebno $\Theta(p_k)$ vremena. Korak (2) zahtijeva $6t_{k-1}$ vremena. Korak (4) je postupak rješavanja Kineskog problema ostataka, za koji je Schönhage pokazao da se gore navedenom Newtonovom metodom može riješiti u $\Theta(p_k \log(p_k))$ vremena. Dakle imamo

$$t_k = 6t_{k-1} + \Theta(p_k \log(p_k)).$$

Uvrštavajući $p_k = 3^{k+2} + 17$ slijedi da je vrijeme potrebno za množenje n-bitnih brojeva ovom metodom

$$T(n) \in \Theta(n^{\log_3 6}) \approx \Theta(n^{1.63}),$$

što prikazuje poboljšanje u odnosu na klasični algoritam za množenje brojeva.

Bibliografija

[1] Alfred V. Aho i John E. Hopcroft, *The design and analysis of computer algorithms*, Pearson Education India, 1974.

[2] Rudolf Albrecht, *Computer algebra: symbolic and algebraic computation*, sv. 4, Springer Science & Business Media, 2012.

[3] Andrej Dujella, *Uvod u teoriju brojeva (skripta)*, 2009, https://web.math.pmf.unizg.hr/~duje/utb/utblink.pdf.

[4] Jon Kleinberg i Eva Tardos, *Algorithm design*, Pearson Education India, 2006.

[5] Donald E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*, Addison-Wesley Professional, 2014.

[6] Isaac J. Schoenberg, *The Chinese Remainder Problem and Polynomial Interpolation*, The College Mathematics Journal **18** (1987), br. 4, 320–322, ISSN 07468342, 19311346, http://www.jstor.org/stable/2686805.

[7] Saša Singer, *Aritmetički i algebarski algoritmi*, 2007, https://web.math.pmf.unizg.hr/~singer/aa_alg/00_aa.pdf.

[8] Saša Singer, Mladen Rogina i Sanja Singer, *Numericka matematika*, 2008, https://web.math.pmf.unizg.hr/~singer/num_mat/num_mat2.pdf.

Sažetak

Vremenska složenost jedan je od najbitnijih faktora kod ocjenjivanja efikasnosti algoritma. Klasični algoritmi za množenje i dijeljenje brojeva imaju kvadratnu vremensku složenost i dovoljno su dobri kod izvođenja aritmetičkih operacija na relativno malim brojevima.

Međutim, za množenje i dijeljenje velikih brojeva, potrebni su nam asimptotski brzi algoritmi za izvođenje tih operacija. Karatsubin algoritam, opisan u ovom radu, jedan je od takvih algoritama. Ima bolju, odnosno manju vremensku složenost od klasičnog algoritma za množenje prirodnih brojeva. Znatno poboljšanje može se primijetiti za velike ulazne brojeve koji imaju puno znamenaka. Zato kažemo da je algoritam asimptotski brz.

Složenost algoritma za računanje recipročne vrijednosti prirodnog broja linearno ovisi o složenosti korištenog algoritma za množenje. To znači da, ukoliko u postupku dijeljenja prirodnih brojeva koristimo asimptotski brz algoritam za množenje brojeva, dobivamo asimtotski brz algoritam za dijeljenje brojeva. Navedeni algoritmi koriste pozicijski zapis prirodnog broja u proizvoljnoj bazi $b \geq 2$. Složenost algoritama ne ovisi o izboru baze.

Alternativni način za prikaz brojeva je modularni zapis, koji dobrim izborom modula omogućuje izvođenje modularne aritmetike u linearnom vremenu. Međutim, ukoliko koristimo modularnu aritmetiku, moramo uračunati vrijeme potrebno algoritmima za konverziju iz modularnog u pozicijski zapis i obrnuto. Schönhage je pokazao da postoji algoritam za množenje prirodnih brojeva koji koristi modularnu aritmetiku. Algoritam uključuje potrebnu konverziju iz jednog zapisa u drugi te ima bolju složenost od klasičnog algoritma, no nešto lošiju od Karatsubinog algoritma.

Možemo zaključiti da pravi izbor algoritma ovisi o potrebama i uvjetima u kojima ćemo ga koristiti, pa je optimalan algoritam često onaj koji je dovoljno dobar.

Summary

Time complexity is one of the most important factors in the evaluation of an algorithm's efficiency. Classical algorithms for multiplication and division of numbers use quadratic time for computation. Therefore, they are good enough for execution of arithmetic operations on relatively small numbers.

However, asymptotically fast algorithms are needed to multiply or divide relatively big ones. The Karatsuba algorithm, described in this paper, is one of those algorithms. It has better, or smaller time complexity than the classical algorithm for multiplication of natural numbers. We can notice a significant improvement for relatively big numbers with numerous digits. That is why we say that the algorithm is asymptotically fast.

The algorithm for calculating the reciprocal value of a number is linearly dependent on the multiplication algorithm it uses. That means if we use an asymptotically fast multiplication algorithm when dividing two numbers, we have an asymptotically fast division algorithm for natural numbers. Given algorithms use the positional notation for representing natural numbers in an arbitrary base $b \geq 2$. An algorithm's complexity does not depend on our choice of the base.

An alternative way of representing numbers is the modular notation. When moduli are chosen correctly, the modular arithmetic is executed in linear time. However, when using the modular arithmetic, we must take into account the conversion from modular to positional notation and vice versa. Schönhage has shown that there is a multiplication algorithm for natural numbers, that uses modular arithmetic. The algorithm includes necessary conversion from one notation to another and has a better complexity than the classical algorithm, but slightly worse one than the Karatsuba algorithm.

We can conclude that the right choice of algorithm depends on our requirements and conditions in which it will be used, so the optimal algorithm is often the one that is just good enough.

Životopis

Rođena sam 9. studenog 1994. godine u Zagrebu. Svoje školovanje sam započela u Osnovnoj školi Gračani u Zagrebu, koju sam završila 2009. godine s odličnim uspjehom. Iste godine upisala sam se u jezičnu XVIII. gimnaziju u Zagrebu, a završila sam ju 2013. godine s odličnim uspjehom. Te sam se godine upisala na Prirodoslovno-matematički fakultet Sveučilišta u Zagrebu, smjer Matematika. Preddiplomski studij završila sam 2018. godine i dobila titulu prvostupnika matematike. Iste godine sam upisala diplomski studij Računarstvo i Matematika na PMF-u u Zagrebu.