Složenost množenja prirodnih brojeva

Babić, Domagoj

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:317710

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Domagoj Babić

SLOŽENOST MNOŽENJA PRIRODNIH BROJEVA

Diplomski rad

Voditelj rada:
prof. dr. sc. Filip Najman

Zagreb, 2023.

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. _____

Mojim roditeljima, Vesni i Miroslavu, koji su me podržavali i uvijek bili uz mene.

Sadržaj

Uvod

Algoritmi su predmet formalnog akademskog istraživanja tek nekoliko desetljeća, a poznati su od početka civilizacije. Opisi koji su sadržavali upute za izvođenje pojedine operacije su među prvim povijesnim zapisima, daleko prije Fibonaccija i al-Khwarizmija pa čak i prije pozicijskog brojevnog sustava koji su oni popularizirali. Ljudi su se s matematikom susretali od samih početaka, u pretpovijesno doba, tj. operacijama zbrajanja, oduzimanja, množenja i dijeljenja relativno malih brojeva, a aritmetika i algoritmi za računanje su se pojavili kada su ljudi počeli računati sa većim brojevima.

U prvom poglavlju rada je predstavljen model za analizu algoritama. U drugom poglavlju su analizirani egipatski i naivni (školski) algoritmi koji bili jedini poznati algoritmi za množenje prirodnih brojeva i u raznim inačicama su se koristili do sredine 1960-tih. A. A. Karatsuba i Y. Ofman su 1963. godine otkrili novi algoritam za množenje prirodnih brojeva koji je danas poznat pod nazivom Karatsubin algoritam za množenje. Taj algoritam je predstavljao inspiraciju i prototip je svih kasnijih algoritama za brzo množenje, ponajprije Toom-Cook algoritama koji su opisani i analizirani u trećem poglavlju. U četvrtom, završnom poglavlju je predstavljen algoritam za množenje koji se temelji na brzoj Fourierovoj transformaciji i analizirana je njegova složenost te Schönhage Strassen algoritam iz 1971. godine, algoritam M. Fürera iz 2007. godine te algoritam D. Harveya i J. van der Hoevena iz 2019. godine.

Diplomski rad izrađen je u sklopu aktivnosti Projekta KK.01.1.1.01.0004 - Znanstveni centar izvrsnosti za kvantne i kompleksne sustave te reprezentacije Liejevih algebri.

Poglavlje 1

Analiza algoritama

Neformalno, algoritam se može shvatiti kao recept, proces, metoda, tehnika, procedura ili rutina, a može se definirati i kao skup pravila iz kojih slijedi niz radnji koje pretvaraju neki ulaz u izlaz. Često za određeni problem postoji više algoritama i korisno je znati koji je najbolji ili optimalan. To nas dovodi do područja analize algoritama. Za zadani problem je korisno odrediti karakteristike algoritama kojima se taj problem rješava. Efikasnost algoritama se može promatrati kroz vrijeme potrebno za njegovo izvođenje, što se može izraziti u broju koraka koji se trebaju izvesti, ali efikasnost možemo promatrati i prema jednostavnosti, eleganciji i mogućnosti praktične primjene, tj. implementacije u računalima.

1.1 Složenost računske operacije

Na prvi pogled se čini da treba biti oprezan pri izboru računalnog modela, odnosno hardvera o kojem će ovisiti je li algoritam učinkovit, međutim, za proučavanje pitanja učinkovitosti je dovoljan apstraktni računalni model - Turingov stroj koji je sposoban simulirati sve fizički ostvarive računalne modele s vrlo malim gubitkom učinkovitosti. Tako je skup "učinkovito izračunljivih" problema barem jednako velik za Turingov stroj kao i za bilo koji drugi drugi računalni model [4].

Računalna učinkovitost se mjeri kao broj osnovnih operacija koje se izvode za rješenje nekog problema, a ovise o duljini ulaza. Međutim, ova funkcija ponekad ovisi o detaljima za definiciju osnovne operacije. Na primjer, algoritam zbrajanja će trebati otprilike tri puta više operacija ako se promatra zbrajanje jednoznamenkastih binarnih brojeva kao osnovna operacija, za razliku od iste operacije na dekadskim brojevima. Složenost nekog algoritma govori koliko je utrošeno resursa za realizaciju algoritma, a može se ticati prostora koji podatci zauzimaju, njihove veličine te vremena potrebnog za realizaciju algoritma, odnosno broja aritmetičkih operacija

potrebnih za izvršavanje algoritma.

Usporedba rasta funkcija

Neka su f i g dvije pozitivne funkcije definirane na skupu prirodnih brojeva.

Definicija 1.1.1 (veliko O)**.** *Kažemo da je funkcija f ograničena funkcijom g ako postoji konstanta C takva da za sve dovoljno velike n vrijedi*

$$f(n) \leq C \cdot g(n).$$

Pišemo $f(n) = O(g(n))$.

Primjer 1.1.2. *Neka je $P_k(n)$ polinom stupnja k, tada vrijedi*

$$P_k(n) = O(n^k).$$

Algoritam složenosti $O(n^3)$ je "teži" od algoritma složenosti $O(n^2)$ jer će zahtijevati više koraka, odnosno više operacija za izvođenje neovisno o arhitekturi računala, procesorskoj brzini ili programskom jeziku u kojem je implementiran.

Definicija 1.1.3. *Kažemo da je problem rješiv u polinomijalnom vremenu ako postoji algoritam složenosti $O(n^p)$, $p \in \mathbb{N}$ koji ga rješava.*

Primjer 1.1.4. *Ako su a i b jednoznamenkasti brojevi, $a \geq b$, tada je složenost računskih operacija zbrajanja $a + b$, oduzimanja $a - b$ i množenja definiranog kao $a \cdot b = \underbrace{a + a + \cdots + a}_{b \text{ puta}}$ $O(1)$ i između tih operacija nema razlike po složensti.*

Složenost pojedinog algoritma se općenito odnosi na gornju granicu broja potrebnih operacija potrebnih za izvođenje, odnosno zapisana je pomoću *velikog O*, no može se izraziti i pomoću donje granice, *velike* Ω, ili ako su gornja i donja granica jednake, pomoću *velike* Θ.

Definicija 1.1.5 (veliko Ω)**.** *Ako postoji konstanta $B > 0$ takva da za sve dovoljno velike n vrijedi*

$$f(n) \geq B \cdot g(n).$$

tada ćemo pisati $f(n) = \Omega(g(n))$.

Definicija 1.1.6 (veliko Θ)**.** *Ako istovremeno vrijedi $f(n) = O(g(n))$ i $f(n) = \Omega(g(n))$, tj. ako postoje konstante B i C takdve da za sve dovoljno velike n vrijedi*

$$B \cdot g(n) \leq f(n) \leq C \cdot g(n),$$

tada ćemo pisati $f(n) = \Theta(g(n))$.

Primjer 1.1.7. *Vrijedi*

$$P_k(n) = \Theta(n^k).$$

Primjer 1.1.8. *Ako su f i g dvije rastuće funkcije za koje postoji limes*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C,$$

za neku konstantu $C \neq 0$, onda je $f(n) = \Theta(g(n))$.

Definicija 1.1.9. *Bit, kratica od binary digit, je znamenka koja može poprimiti vrijednosti* 0 *ili* 1*, odnosno binarna znamenka.*

Teorem 1.1.10. *Za prikaz prirodnog broja $n < 2^m$ dovoljno je m bitova.*

Binarne - elementarne operacije

Pretpostavimo da su brojevi prikazani u binarnom sustavu. Simboli 0 i 1 u tom sustavu se zovu bitovi, a operacije zapisivanja znaka, zbrajanja, oduzimanja ili množenja dva bita ili zapis zagrada se smatraju operacijama jednog bita ili elementarnim operacijama.

Tako je za prikaz dva n-znamenkasta broja a i b u binarnom brojevnom sustavu potrebno $2n$ elementanih operacija iz čega slijedi da složenost njihovog zbroja nije manja od $3n$. Također, algoritam zbrajanja i oduzimanja u općenitom smislu ne zahtijeva više od $4n$ elementarnih operacija, odnosno složenosti je $O(n)$. Slijedi da je broj potrebnih elementarnih operacija za zbrajanje i oduzimanje u asimptotskom smislu jednak.

Sljedeći problem je odrediti koliko je elemtarnih operacija potrebno za izračunati umnožak ab. Može se uočiti da je taj problem istovjetan računanju kvadrata broja jer uz pretpostavku $a > b$ vrijedi

$$ab = \frac{(a+b)^2 - (a-b)^2}{4},$$

što znači da je složenost računanja umnoška ab istovjetna računanju složenosti računanja kvadrata broja. Funkcija $M(n)$ će označavati složenost računanja umnoška dva broja a i b, gdje je $a > b$, odnosno a^2, gdje je a n-znamenkasti broj.

Galaktički algoritam

Pojam *galaktički algoritam* uveli su R. J. Lipton i K. W. Regan kako bi opisali algoritme koji imaju poželjnu asimptotsku složenost, ali se nikada ne bi koristili u praksi [16]. Izraz je postao popularan 2020. godine, nakon razvoja prvog algoritma

za množenje cijelih brojeva koji se izvodi u vremenu $O(n \cdot \log n)$ što je predstavljalo veliko otkriće, ali algoritam postiže navedenu složenost za ulazne brojeve sa najmanje $2^{1729^{12}}$ bita (najmanje $10^{10^{38}}$ znamenki) [13] što je više od broja atoma poznatog svemira, kojih ima oko 10^{80} [5].

Poglavlje 2

Povijesni algoritmi

2.1 Složenost egipatskog algoritma za množenje

Papirus iz Egipta je napisan oko 1800. prije nove ere. Pisar Ahmos navodi da egipatski algoritam za množenje potječe iz izvornika napisanog u Srednjem Kraljevstvu (2000. - 1800. prije nove ere). U njemu je opisano oko 84 problema posvećenih tehnikama računanja i korištenju dekadskog brojevnog sustava, a algoritam množenja se izdvaja svojim neuobičajenim svojstvima. Izvodi se udvostručavanjem i zbrajanjem rezultirajućih vrijednosti u međuračunu [10]. Napomena: Udvostručavanje se u binarnom sustavu svodi na pomak znamenaka ulijevo i složenosti je $O(1)$ [10].

Primjer 2.1.1. *Umnožak* $12 \cdot 12$ *je u Papirusu naveden kao Problem br. 32, zapisan hijeroglifima, a u moderniziranom zapisu izgleda ovako:*

$$\begin{array}{rl} 1 & 12 \\ 2 & 24 \\ *4 & 48 \\ *8 & 96 \end{array}$$

Broj 12 *treba pomnožiti brojem* 4 *i brojem* 8*, odnosno ponoviti postupak udvostručavanja tri puta. Zatim promatramo brojeve s lijeve strane koji u zbroju daju* 12*, tj.* $4+8$*, odnosno množitelj zadanog broja* 12*, a s lijeve strane zbrojimo odgovarajuće vrijednosti, tj.* $48 + 96 = 144$*.*

Primjer 2.1.2. *Opis postupka množenja* $11 \cdot 13$ *:*

$*1$	13
$*2$	26
4	52
$*8$	104

Množenje broja 11 *brojem* 13 *se odvija na sljedeći način: u prvom koraku je broj* 13 *pomnožen brojem* 2 *(udvostučen), zatim je dobiveni rezultat ponovno udvostručen, tj.* 13 *pomnožen brojem* 4*, zatim je dobiveni rezultat opet udvostručen, tj.* 13 *pomnožen brojem* 8*. Za izračun umnoška* $13 \cdot 11$ *je potrebno zbrojiti sve brojeve s lijeve strane koji u zbroju daju* 11 *te zbrojiti odgovarajuće brojeve s desne strane, što ukupno daje* 143*.*

Primjer 2.1.3. *Egipćani su koristili sljedeću shemu za računanje umnoška* $15 \cdot n$ *:*

$$n \cdot 15 = n \cdot (2^0 + 2^1 + 2^2 + 2^3) = n \cdot 1 + n \cdot 2 + n \cdot 2^2 + n \cdot 2^3.$$

Prikazali su množitelj (faktor) u binarnom sustavu i onda proveli množenje svakom binarnom znamenkom posebno.

Ako su a i b n-znamenkasti brojevi, tada će operacije zbrajanja i oduzimanja zahtijevati $O(n)$ operacija, a računanje umnoška $a \cdot b$, ako je definirano kao uzastopno zbrajanje, zahtijevati $O(n \cdot 2^n)$ operacija.

Za procjenu složenosti algoritma, neka je broj b prikazan u binarnom sustavu, tj.

$$b = 2^{n_1} + 2^{n_2} + \cdots + 2^{n-1},$$

$$0 \leq n_1 \leq n_2 \leq \cdots \leq n - 1.$$

Tada se $a \cdot b$ može prikazati kao

$$a \cdot b = a \cdot 2^{n_1} + a \cdot 2^{n_2} + \cdots + a \cdot 2^{n-1}. \tag{2.1}$$

Počevši s a, uzastopnim udvostručavanjem a dobiva se

$$2a, 2^2a, \ldots 2^{n_1}a, \ldots, 2^{n_2}a, \ldots 2^{n-1}a.$$

Na ovaj način, zbrajajući one članove koji se pojavljuju na desnoj strani (2.1), računamo $a \cdot b$. Svaki od pribrojnika je najviše $2n$-znamenkasti broj i ukupan broj pribrojnika je manji od n. Za izračunati svaki od pribrojnika, potrebno je $O(n)$ operacija jer se svaki od pribrojnika dobiva zbrajanjem najviše $2n$-znamenkastih brojeva. S obzirom da u konačnici treba uzastopno zbrojiti najviše n članova slijedi da je za izračun desne strane (2.1) dovoljno $O(n^2)$ operacija. Dakle, egipatski algoritam za množenje je složenosti $O(n^2)$ [14].

2.2 Složenost školskog algoritma za množenje

Definicija 2.2.1. *Kažemo da je broj* $x = \overline{X[m-1]X[m-2]...X[0]}$ *zapisan u dekadskoj bazi ako je*

$$x = \sum_{i=0}^{m-1} X[i] \cdot 10^i.$$

Algoritam školskog ili naivnog množenja pretpostavlja da su brojevi zapisani kao nizovi znamenki. Pretpostavit ćemo da su brojevi zapisani u dekadskom brojevnom sustavu, ali algoritam se može generalizirati na bilo koju bazu. Za pojednostavljenje zapisa, pretpostavit ćemo da su ulazni parametri nizovi $X[0 \ldots m-1]$ i $Y[0 \ldots n-1]$ koji predstavljaju brojeve

$$x = \sum_{i=0}^{m-1} X[i] \cdot 10^i \qquad \text{i} \qquad y = \sum_{j=0}^{n-1} Y[j] \cdot 10^j$$

i slično, da je izlaz (rezultat) predstavljen nizom znamenaka $Z[0 \ldots m+n-1]$,

$$z = x \cdot y = \sum_{k=0}^{m+n-1} Z[k] \cdot 10^k.$$

Algoritam školskog množenja se može zapisati kao formula

$$x \cdot y = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (X[i] \cdot Y[j] \cdot 10^{i+j}).$$

Različite inačice ovog algoritma izračunavaju parcijalne umnoške $(X[i] \cdot Y[j] \cdot 10^{i+j})$ u različitim poretcima i koriste različite strategije za računanje sume.

U ovom algoritmu, zbrajanje i množenje jednoznamenkastih brojeva su elementarne operacije, tj. složenosti O(1). U praksi se množenje jednoznamenkastih brojeva izvodi koristeći tablicu množenja koja može biti uklesana na glinenim pločicama, napisana na papiru, pohranjena u memoriji računala ili zapamćena.

Fibonaccijev algoritam se često izvodi zapisujući sve parcijalne umnoške u dvodimenzionalnu tablicu, koju možemo zamisliti kao rešetku, i zatim zbrajamo po dijagonalama pazeći pritom na odgovarajući prijenos. U osnovnoj školi se algoritam množenja poučava na sličan način tako što svaku znamenku jednog broja množimo svakom znamenkom drugog, zapisujući rezultate svih međukoraka, koje na kraju zbrojimo. Ovu metodu je na sličan način opisao Eutocije, a svi spomenuti algoritmi (i nekolicina ostalih) su opisani i ilustrirani u knjizi *L'Arte dell'Abbaco*, prvoj matematičkoj knjizi tiskanoj na Zapadu, iz 1458. godine, poznatoj i pod nazivom *Treviso Arithmetic* [10].

Procijenimo $M(n)$ algoritma školskog množenja. Neka broj a ima najmanje $\frac{n}{2}$ jedinica u svojem binarnom prikazu. Tada tablica brojeva koji odgovaraju a^2, i zapisu školskog množenja, koje uključuje zbrajanje, sadrži najmanje $\frac{n^2}{2}$ bitova i ne više od $2n^2$ bitova. Nije potrebno više od $8n^2$ elementarnih operacija za zbrajanje n, najviše $2n$-znamenkastih brojeva. Slijedi da se $M(n)$ može procijeniti na sljedeći način

$$4n \leq M(n) \leq 8n^2,$$

odnosno, ako nas zanima gornja međa, $M(n) = O(n^2)$ što se podudara sa složenosti egipatskog algoritma za množenje [14].

Složenost svih algoritama za množenje m-znamenkastog broja n-znamenkastim koji su zasnovani na principu rešetke je $O(mn)$, tj. $O(n^2)$.

Uočimo da u slučaju kada bi se izrazi na desnoj strani (2.1) dobivali dodavanjem odgovarajućeg broja nula na desnoj strani broja a, a ne uzastopnim zbrajanjem, tada bi umjesto egipatskog dobili klasični školski algoritam za množenje prirodnih brojeva. Pretpostavlja se da se do školskog algoritma za množenje došlo nakon što je 0 uvedena kao broj [14].

Na sastancima Moskovskog Matematičkog Društva 1956. godine, Kolmogorov je iznio svoju slutnju da je najmanja procjena za $M(n)$ reda veličine $O(n^2)$. Prirodno je tu slutnju zvati *Kolmogorovljeva n^2 slutnja* koju je tako postavio jer su ljudi kroz povijest koristili algoritme za množenje čija je složenost $O(n^2)$, a da je postojala *ekonomičnija* metoda, vjerojatno bi već bila otkrivena. U sličnom nizu pretpostavki da ne postoji algoritam manje složenosti od $O(n^2)$ za množenje prirodnih brojeva komentirao je i Babenko: *Za razvoj znanosti je poznata važnost odabira dobrog brojevnog sustava što se očituje i u starom Babilonu gdje se koristio heksagezimalni brojevni sustav koji je se pokazao kao odličan i za prikaz razlomaka. Što se brojevnih sustava i metoda računanja tiče, izgleda da su najbolji algoritmi već davno otkriveni* [15].

Poglavlje 3

Rekurzivni algoritmi za množenje

Redukcija problema je najčešća tehnika koja se koristi u dizajniranju algoritama. Reducirati problem X na problem Y znači opisati algoritam za X koji koristi algoritam za Y kao tzv. crnu kutiju, odnosno subrutinu. Bitno je naglasiti da korektnost takvog algoritma za X ne ovisi o načnu na koji algoritam radi za Y. Pretpostavlja se da algoritam korektno izvodi Y. Takav algoritam zovemo crnom kutijom jer nas ne zanima na kojim principima on radi.
Rekurzija je poseban tip redukcije koja se može opisati kao:

- ako se dani problem može riješiti direktno, riješi,
- u suprotnom, reduciraj problem u manje dijelove istog problema.

Algoritam rekurzije koji se izvodi u sljedeća 3 koraka zovemo *podijeli pa vladaj*:

1. **Podijeli** zadani problem u nekoliko manjih nezavnisnih problema istog tipa,
2. **Izvrši** manji problem,
3. **Spoji** dobivena rješenja manjih problema u konačno rješenje.

Ako je veličina manjeg problema manja od neke postavljene veličine, tada napuštamo rekurziju i rješavamo ga direktno u konstantnom vremenu, tj. $O(1)$. Jedini uvjet kako bi rekurzivna metoda bila ispravna jest da ne smije postojati beskonačan niz redukcija na jednostavnije probleme istog tipa, odnosno svaka rekurzivna redukcija mora voditi k elementarnom slučaju koji se može riješiti nekom drugom metodom. U suprotnom, rekurzivni algoritam neće dati rezultat u konačnom vremenu.

Rekurzije koje su povezane uz algoritme tipa **podijeli pa vladaj** obično imaju oblik

$$f(n) = bf(n/k) + g(n),$$

gdje je k prirodan broj koji govori na koliko se dijelova u algoritmu dijeli početni skup, a broj b je konstanta ovisna o algoritmu kao i funkcija smetnje $g(n)$.
Ova rekurzivna relacija ima smisla samo ako je n/k cijeli broj, a u suprotnom će se možda tek zanemarivo razlikovati od prave vrijednosti funkcije.

Teorem 3.0.1 (Složenost *podijeli pa vladaj* algoritama)**.** *Rekurzivna relacija ima sljedeća rješenja, u ovisnosti o vrijednosti konstante b i funkcije smetnje* $g(n)$

	b	$g(n)$	$f(n)$
1.	1	c	$c\log_k n + a_0$
2.	1	cn	$\frac{ck}{k-1}n + a_0 - \frac{ck}{k-1}$
3.	k	c	$An - \frac{c}{k-1}$
4.	k	cn	$n(c\log_k n) + A$
5.	$\neq k$	c	$An^{\log_k b} - \frac{c}{b-1}$
6.	$\neq k$	cn	$An^{\log_k b} - \frac{kc}{k-b}n$

A je neodređena konstanta koja se računa iz početnih uvjeta.

Dokaz. **1.** Promatrajmo članove niza s indeksima oblika $n = k^m$. Niz (a_m) definiran s

$$a_m = f(n) = f(k^m)$$

zadovoljava rekurzivnu jednadžbu

$$a_m = a_{m-1} + c.$$

Njezino je rješenje

$$a_m = a_0 + mc.$$

Budući da je $m = \log_k n$, dobivamo

$$f(n) = c\log_k n + a_0$$

Primjetimo da za n koji nije oblika k^m ovaj broj ne mora biti cijeli, i on predstavlja samo aproksimaciju prave vrijednosti za $f(n)$.
2. Na isti način dobivamo rekurziju

$$a_m = a_{m-1} + cn = a_{m-1} + k^m$$

pa je

$$\begin{aligned} a_m &= a_0 + c(k^m + k^{m-1} + \cdots + k) \\ &= a_0 + ck\frac{k^m - 1}{k - 1} \\ &= a_0 + ck\frac{n - 1}{k - 1} \end{aligned}$$

i odavdje slijedi tvrdnja.
3. Neka je sada $b \neq 1$. Onda rekurzija ima oblik

$$a_m = ba_{m-1} + g(n)$$

i njezino je opće rješenje oblika

$$a_m = A \cdot b^m + g_p.$$

Tu je g_p partikularno rješenje rekurzije. Sada imamo

$$b^m = b^{\log_k n} = n^{\log_k b}.$$

Ukoliko je $g(n) = c$, onda partikularno rješenje ima oblik $g_p = B$, pri čemu je B konstanta koja se dobiva vrštavanjem ove funkcije u rekurziju. Tako dobivamo **3.** i **5**.
Neka je sada $g(n) = cn$. Ako je $b \neq k$, partikularno rješenje ima oblik Bn pa uvrštavanjem dobivamo 6.
Konačno, za $g(n) = cn$ i $b = k$ rekurzija glasi

$$a_m = k \cdot a_{m-1} + ck^m.$$

Sad je opće rješenje oblika $A \cdot k^m = An$, a partikularno ima oblik Bmk^m. Uvrštavanjem dobivamo $B = c$ i odatle **4**. □

Pomoću Teorema 3.0.1 jednostavno određujemo složenost rekurzija tipa podijeli pa vladaj. Za određivanje složenosti rekurzija postoji i metoda koju možmo zamisliti i kao stablo koje od korijena ima po jedan list za svaki rekurzivan potproblem gdje je vrijednost svakog čvora količina vremena koje se utroši za rješavanje danog potproblema isključujući rekurzivni poziv [8].

Algoritam ruskog seljaka za množenje

Algoritam množenja ruskog seljaka se poučavao u Rusiji i krajem 20. stoljeća, a bio je implementiran u računalima koja nisu imala množenje ostvareno direktno pomoću sklopova [10]. Ovaj algoritam reducira množenje brojeva u četiri jednostavnije operacije: (1) određivanje parnosti, (2) zbrajanje, (3) udvostručavanje i (4) polovljenje zaokruživanjem na dolje.

Korektnost algoritma slijedi iz rekurzije, koja vrijedi za sve prirodne brojeve:

$$x \cdot y = \begin{cases} 0 & \text{ako je } x = 0, \\ \lfloor x/2 \rfloor \cdot (y+y) & \text{ako je } x \text{ paran}, \\ \lfloor x/2 \rfloor \cdot (y+y) + y & \text{ako je } x \text{ neparan}. \end{cases} \tag{3.1}$$

Navedeni algoritam je u svojoj biti rekurzivan i po njemu će se izvesti $O(\log x)$ operacija (1),(2) i (4), ali ga možemo unaprijediti do $O(\log \min\{x, y\})$ ako bi zamijenili brojeve x i y kada $x > y$. Uz pretpostavku da su brojevi zapisani u pozicijskom brojevnom sustavu, svaka operacija zahtijeva najviše $O(\log(xy)) = O(\log \max\{x, y\})$ operacija sa jednoznamenkastim brojevima pa je ukupno vrijeme za izvođenje algoritma $O(\log \min\{x, y\} \cdot \log \max\{x, y\}) = O(\log x \cdot \log y)$. Drugim riječima, ovaj algoritam zahtijeva $O(mn)$ operacija za množenje m-znamenkastog broja n-znamenkastim brojem, što je jednako složenosti algoritma školskog množenja. Možemo reći da su operacije (1)-(4) jednostavnije, ali su algoritmi ekvivalentni ako brojeve zapišemo i množimo u binarnom sustavu. Napomena: Za izvođnje ovog algoritma na papiru, postoji konstanta C za koju će ovaj algoritam zahtijevati C puta više mjesta [10].

Neka $T(x, y)$ označava broj operacija kojima se provjerava parnost, zbrajanja i polovljenja potrebnih za izračun $x \cdot y$. Funkcija T zadovoljava rekurzivnu nejednakost $T(x, y) \leq T(\lfloor x/2 \rfloor, 2y)+2$ za bazu $T(0, y) = 0$. Prema 3.0.1 slijedi $T(x, y) = O(\log x)$. Umnožak $x \cdot y$ se izvodi računanjem jednostavnijeg umnoška $x' \cdot y'$ te dodavajući y. Jednostavniji je zato što vrijedi $x' < x$, a ponavljanjem taj broj teži 0.

Primjer 3.0.2. *Pokušajmo dobiti efikasniji algoritam primjenom strategije podijeli pa vladaj. Podijelimo znamenke broja u dva jednaka dijela i provjerimo sljedeći identitet:*

$$(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd,$$

a svaki od manjih problema množenja ac, bc, ad i bd se računaju rekurzivno, ali u posljednjem koraku nije rekurzivno jer množenje potencijom broja 10 *izvodimo pomicanjem brojeva u lijevo i dodavanjem odgovarajućeg broja nula, sve u vremenu $O(n)$.*

Potrebno vrijeme za izvođenje tog algoritma je

$$T(n) = 4T(\lceil n/2 \rceil) + O(n).$$

Prema Teoremu 3.0.1 vrijedi $T(n) = O(n^{\log_2 4}) = O(n^2)$. *U stvari, izvođenjem ovog algoritma ćemo množiti svaku znamenku broja* x *sa svakom znamenkom broja* y, *kao u algoritmu školskog množenja. I u ovom primjeru nije došlo do smanjenja složenosti algoritma za množenje prirodnih brojeva.*

Karatsubin algoritam

U jesen 1960. godine, bio je održan seminar na Strojarskom fakultetu moskovskog sveučilišta na temu matematičkih problema u kibernetici pod vodstvom Kolmogorova na kojem je Kolmogorov iznio svoju slutnju i predstavio probleme koji se tiču procjene složenosti rješavanja sustava linearnih jednadžbi te sličnih računskih problema. Unutar tjedan dana od početka održavanja seminara, Karatsuba je pronašao algoritam za koji se nadao da će poboljšati $M(n)$, odnosno smanjiti složenost algoritma za množenje.

Prije sljedećeg seminara Karatsuba je svoje rješenje priopćio Kolmogorovu koji je bio vrlo uzrujan jer je ono bilo u kontradikciji s njegovom uvjerljivom slutnjom. Na idućem sastanku Kolmogorov je prikazao Karatsubino rješenje zbog čega je seminar bio otkazan. Dvije godine kasnije, u suradnji sa Ofmanom, Kolmogorov je napisao i objavio članak "Množenje višeznamenkastih brojeva na automatu" u kojem je umjesto sebe potpisao A. Karatsubu, a Karatsuba je za članak saznao tek nakon što je dobio kopiju. Kolmogorov je iste godine predstavio i rad Ofmana kojim je zainteresirao matematičare u tom području primijenjene matematike, koji su nazvali *brzo računanje* [14].

Kod množenja

$$(a + b)(c + d) = ac + bc + ad + bd$$

Karatsuba je uočio da se srednji koeficijent $bc + ad$ može izračunati iz preostala dva koeficijenta ac i bd koristeći samo *jedno* množenje iz identiteta:

$$ac + bd - (a - b)(c - d) = bc + ad.$$

Koristeći ovaj trik, četiri rekurzivna poziva možemo zamijeniti sa samo tri, iz čega slijedi da je vrijeme potrebno za izvođenje Karatsubinog algoritma za brzo množenje

$$T(n) \leq 3T(\lceil n/2 \rceil) + O(n).$$

Karatsubin algoritam za množenje ima složenost

$$M(n) = O\left(n^{\log_2 3}\right), \qquad \log_2 3 = 1.5849\ldots.$$

Razdjeljivanje znamenaka u 2 dijela

Opišimo Karatsubin algoritam matematički. Pretpostavimo da su brojevi A i B otprilike jednake duljine i razdijelimo ih u dva dijela

$$A = xa_1 + a_0 \quad \text{i} \quad B = xb_1 + b_0,$$

gdje je x potencija baze, tj. broj sa upola manje znamenaka nego A. Uobičajenim množenjem će biti potrebne 4 operacije množenja:

$$AB = a_0 \cdot b_0 + x(a_0 \cdot b_1 + b_0 \cdot a_1) + x^2 a_1 \cdot b_1.$$

Promotrimo samo množenja $a_i \cdot b_j$ za $i, j = 0, 1$ jer množenje sa bazom x možemo shvatiti kao operaciju pomaka u lijevo, odnosno zapisivanje odgovarajućeg broja 0 s desne strane broja. Ako upotrijebimo relaciju

$$AB = (1 + x)a_0 \cdot b_0 + x(a_1 - a_0) \cdot (b_0 - b_1) + (x + x^2)a_1 \cdot b_1,$$

trebat ćemo samo tri operacije množenja, od toga dva množenja broja sa upola manje znamenaka. Rekurzivnim zadavanjem ovakvog množenja, početni problem možemo rastaviti do problema množenja brojeva koje je moguće izmnožiti u jednom koraku. Na takav način ćemo dobiti algoritam koji se izvodi u $O(n^{\log_2 3}) \approx O(n^{1.585})$. Alternativno, podjelu možemo opisati i na način

$$AB = (1 - x)a_0 \cdot b_0 + x(a_1 + a_0) \cdot (b_0 + b_1) + (x^2 - x)a_1 \cdot b_1.$$

Dakle, za kvadriranje broja A koristi se

$$A^2 = (1 + x)a_0^2 - x(a_1 - a_0)^2 + (x + x^2)a_1^2,$$
$$A^2 = (1 - x)a_0^2 + x(a_1 + a_0)^2 + (x^2 - x)a_1^2.$$

Primjer 3.0.3. *Izračunajmo* $8231^2 = 67749361$ *koristeći prvu jednakost:*

$$\begin{aligned} 8231^2 &= (100 \cdot 82 + 31)^2 \\ &= (1 + 100) \cdot 31^2 - 100 \cdot (82 - 31)^2 + (100 + 100^2) \cdot 82^2 \\ &= (1 + 100) \cdot 961 - 100 \cdot 2601 + (100 + 100^2) \cdot 6724 \\ &= 961 + 96100 - 260100 + 672400 + 6740000 \\ &= 67749361. \end{aligned}$$

3.1 Karatsubin algoritam i njegova složenost

Ovako je Karatsuba u svom radu Karatsuba [14] predstavio analizu složenosti svojeg algoritma za množenje. Kao što je već spomenuto, množenje dva broja a i b, gdje je $a > b$ se može reducirati na kvadriranje n-znamenkastog broja a. Bez smanjenja općenitosti, pretpostavimo $n = 2^m$ i neka vrijedi $a = 2^{n_1}a_1 + a_2,\ 2n_1 = n$ gdje su a_1 i a_2 n_1-znamenkasti brojevi. Imamo

$$a^2 = (2^{n_1}a_1 + a_2)^2 = 2^n a_1^2 + 2^{n_1} 2a_1a_2 + a_2^2.$$

Štoviše,

$$2a_1a_2 = (a_1 + a_2)^2 - a_1^2 - a_2^2,$$

iz čega slijedi

$$a^2 = 2^n a_1^2 - 2^{n_1}a_1^2 + 2^{n_1}(a_1 + a_2)^2 + a_2^2 - 2^{n_1}a_2^2.$$

S obzirom da su a_1 i a_2 n_1-znamenkasti brojevi, njihov je zbroj najviše $(n_1 + 1)$-znamenkast i može se prikazati i u obliku

$$a_1 + a_2 = \epsilon + 2a_3, \tag{3.2}$$

gdje je $\epsilon = 0$ ili 1, a a_3 je n_1-znamenkasti broj. Iz čega slijedi

$$(a_1 + a_2)^2 = \epsilon^2 + 4\epsilon a_3 + 4a_3^2. \tag{3.3}$$

Iz (3.2) i (3.3) slijedi

$$a^2 = 2^n a_1^2 - 2^{n_1}a_1^2 + 2^{n_1+2}a_3^2 + 2^{n_1+2}\epsilon a_3 + 2^{n_1}\epsilon^2 + a_2^2 - 2^{n_1}a_2^2. \tag{3.4}$$

Neka je $M(n)$ broj elementarnih operacija dovoljnih za izračun a^2. Napomena: množenje potencijom broja 2 u binarnom sustavu se svodi na dodavanje odgovarajućeg broja nula s desne strane početnog broja. Tako je za izračun a^2, prema (3.4), potrebno kvadrirati tri n_1-znamekasta broja a_1, a_2 i a_3 što zahtijeva $3M(n_1)$ operacija. Tada svaki dobiveni broj trebamo još pomnožiti s 2^n, 2^{n_1} i 2^{n_1+2} što zahtijeva najviše $6n$ operacija. Na kraju je potrebno zbrojiti sedam, najviše $2n$-znamenkastih brojeva što ne zahtijeva više od

$$4 \cdot 2n \cdot 4 + 4(2n + 2) \cdot 2 + 4(2n + 2) = 56n + 24$$

operacija. Iz čega slijedi

$$M(n) \leq 3M(n_1) + 6n + 56n + 24 \leq 3M(n_1) + 70n. \tag{3.5}$$

Nadalje, definiramo li $2n_{j+1} = n_j$, $j = 1, \ldots, m-1$, dobivamo

$$M(n_j) \leq 3M(n_{j+1}) + 70n_j. \tag{3.6}$$

S obzirom da je $n_{j+1} = 2^{m-j-1}$ imamo $n_m = 1$ i, trivijalno $M(1) = 1$. Koristeći (3.5) i (3.6), matematičkom indukcijom možemo dokazati da nejednakost

$$M(n) \leq 3^j M(n_j) + 3^{j-1} \cdot 70n_{j-1} + \cdots + 3 \cdot 70n_1 + 70n \tag{3.7}$$

vrijedi za $j \geq 1$. Očito vrijedi za $j = 1$. Pretpostavimo da izraz vrijedi za neki $j \in \mathbb{N}$, dokažimo da vrijedi za $j + 1 \leq m$. Koristeći (3.6) i uvrštavanjem u (3.7) slijedi:

$$M(n) \leq 3^{j+1} M(n_{j+1}) + 3^j \cdot 70n_j + 3^{j-1} \cdot 70n_{j-1} + \cdots + 3 \cdot 70n_1 + 70n, \tag{3.8}$$

što smo i htjeli dokazati.
Postavimo sada $j = m$, $M(n_m) = M(1) = 1$ i $n_j = 2^{m-j}$ u (3.7) kako bismo dobili

$$\begin{aligned} M(n) &\leq 3^m + 3^{m-1} \cdot 70n_{m-1} + 3^{m-2} \cdot 70n_{m-2} + \cdots + 3 \cdot 70n_1 + 70n \\ &= 3^m + 3^{m-1} \cdot 70 \cdot 2 + 3^{m-2} \cdot 70 \cdot 2^2 + \cdots + 3 \cdot 70 \cdot 2^{m-1} + 70 \cdot 2^m \\ &= 3^m \left(1 + 70 \cdot \frac{2}{3} + 70 \cdot \left(\frac{2}{3}\right)^2 + \cdots + 70 \cdot \left(\frac{2}{3}\right)^{m-1} + 70 \cdot \left(\frac{2}{3}\right)^m \right) \\ &< 70 \cdot 3^m \cdot \left(1 - \frac{2}{3}\right)^{-1} = 210 \cdot 3^m. \end{aligned}$$

S obzirom da je $n = 2^m$, odnosno $m = \log_2 n$, slijedi:

$$M(n) < 210 \cdot n \log_2 3, \quad \log_2 3 = 1.5849...$$

Posebno, slijedi

$$M(n) = O(n^{\log_2 3}).$$

Uočimo da je procjena $M(n)$ značajno premašena kako bi račun bio što jednostavniji. Račun izveden na opisani način se može izvesti efikasnije, tj. s manje operacija i konstanta će biti puno manja od 210.
Postoji i inačica Karastubinog algoritma po kojem se dva n-znamenkasta broja a i b pomnože direktno, a algoritam je opisan na sljedeći način:

$$a = 2^{n_1} a_1 + a_2, \quad b = 2^{n_1} b_1 + b_2, \quad 2n_1 = n,$$

iz čega slijedi

$$\begin{aligned} ab &= (2^{n_1} a_1 + a_2)(2^{n_1} b_1 + b_2) = 2^n a_1 b_1 + 2^{n_1}(a_2 b_1 + a_1 b_2) + a_2 b_2 \\ &= 2^n a_1 b_1 - 2^{n_1} a_1 b_1 + a_2 b_2 - 2^{n_1} a_2 b_2 + 2^{n_1}(a_1 + a_2)(b_1 + b_2). \end{aligned} \tag{3.9}$$

U ovako postavljenoj relaciji imamo tri umnoška a_1b_1, a_2b_2 i $(a_1+a_2)(b_1+b_2)$. Svaki od faktora je najviše (n_1+1)-znamenkasti broj. Neka $M(n)$ opet predstavlja broj potrebnih (dovoljnih) računskih operacija za izračunati umnožak dva n-znamenkasta broja tada iz (3.9) slijedi

$$M(n) < 3M(n_1) + c \cdot n,$$

gdje je $c > 0$ konstanta i $2n_1 = n$. Broj potrebnih operacija ovako zadanog algoritma se može procijeniti na

$$M(n) < c_1 \cdot n^{\log_2 3},$$

gdje je sada $c_1 > 0$ konstanta.

3.2 Višestruko razdjeljivanje znamenaka

Razdjeljivanje znamenaka u 3 dijela

Karatsubinu ideju možemo razraditi, dijeleći znamenke broja u više dijelova i spajajući rezultat kako bi postigli još brže algoritme. Andrei Toom je otkrio beskonačnu familiju algoritama koji dijele bilo koji prirodni broj u k dijelova, svaki sa n/k znamenki te zatim računajući umnožak koristeći samo $2k-1$ rekurzivnih množenja. Toomove algoritme je daljnje pojednostavio Stephen Cook u svojoj doktorskoj disertaciji [7]. Algoritmi koji razdjeljuju faktore A i B u više od 2 dijela općenito se nazivaju *Toom-Cook* algoritmi. Za neki fiksni k, Toom-Cookov algoritam se izvodi u $O(n^{1+1/(log\ k)}) = O(n^{1+\epsilon})$ vremenu, gdje ϵ ovisi o k. Svi ti algoritmi asimptotski su brži od klasičnog za koji je potrebno izvesti $O(n^2)$ operacija.

U algoritmu razdjeljivanja znamenaka u 3 dijela, tzv. *Toom-Cook-3* algoritmu, neka su znamenke brojeva podijeljene u tri dijela iste duljine, tj. neka su a_i i b_i za $i = 0, 1, 2$ istog broja znamenaka i neka su brojevi koje množimo, po dijelovima svojih znamenki a_i i b_i, zapisani kao polinomi:

$$A(x) = a_2x^2 + a_1x + a_0,$$
$$B(x) = b_2x^2 + b_1x + b_0,$$

gdje je x "prikladna" i-ta potencija broja 10.

Primjer 3.2.1. *Kako bi pomnožili dva broja* $831275469 \cdot 879512436$*, postavit ćemo* $a_2 = 831$*,* $a_1 = 275$ *i* $a_0 = 469$ *i* $b_2 = 897$*,* $b_1 = 512$ *i* $b_0 = 436$ *te* $x = 10^3$

Umnožak $A(x)\cdot B(x) = C(x) = c_4x^4+c_3x^3+c_2x^2+c_1x+c_0$ se dobiva uvrštavanjem vrijednosti x. .

Postavimo:

$$\begin{aligned}
S_0 &:= a_0 \cdot b_0,\\
S_1 &:= (a_2 + a_1 + a_0) \cdot (b_2 + b_1 + b_0),\\
S_2 &:= (4 \cdot a_2 + 2a_1 + a_0) \cdot (4 \cdot b_2 + 2b_1 + b_0),\\
S_3 &:= (a_2 - a_1 + a_0) \cdot (b_2 - b_1 + b_0),\\
S_4 &:= a_2 \cdot b_2,
\end{aligned}$$

Ovaj postupak zahtijeva 5 množenja brojeva duljine $N/3$. U prvom koraku se izračuna $c_0 = S_0$ i $c_4 = S_4$, a c_1, c_2 i c_3 računamo prema:

$$\begin{aligned}
T_1 &:= 2S_3 + S_2 && (= 18c_4 + 6c_3 + 6c_2 + 3c_0),\\
T_1 &:= T_1/3 && (= 6c_4 + 2c_3 + 2c_2 + c_0),\\
T_1 &:= T_1 + S_0 && (= 6c_4 + 2c_3 + 2c_2 + 2c_0),\\
T_1 &:= T_1/2 && (= 3c_4 + c_3 + c_2 + c_0),\\
T_1 &:= T_1 - 2S_4 && (= c_4 + c_3 + c_2 + c_0),\\
T_2 &:= (S_1 + S_3)/2 && (= c_4 + c_2 + c_0),\\
S_1 &:= S_1 - T_1 && (= c_1),\\
S_2 &:= T_2 - S_0 - S_4 && (= c_2),\\
S_3 &:= T_1 - T_2 && (= c_3).
\end{aligned}$$

Slijedi

$$C = A \cdot B = S_4x^4 + S_3x^3 + S_2x^2 + S_1x + S_0.$$

Složenost algoritma na ovom principu razdjeljivanja znamenki je $O(N^{\log_3 5}) \approx O(N^{1.465})$. Uočimo dijeljenje brojem 3. Dijeljenje konstantom (koja nije potencija broja 2) se ne može izbjeći kod n-strukog razdjeljivanja znamenaka za $n \geq 3$. To dijeljenje je bez ostatka kao i sva dijeljenja konstantnom prikazana u radu u ostalim algoritmima koji koriste tehniku razdjeljivanja brojeva. Postoje metode kvadriranja koje ne uključuju takva dijeljenja [3].

Razdjeljivanje znamenaka u 3 dijela [Bodrato i Zanoni]

Alternativni algoritam za razdjeljivanje znamenaka u 3 skupine:

Postavimo $S_1, \ldots, S_4$ kao u prethodnom algoritmu, tada c_1, c_2 i c_3 računamo prema:

$$\begin{aligned}
&S_2 := (S_2 - S_3) && (= 5c_4 + 5c_3 + c_2 + c_1),\\
&S_3 := (S_1 - S_2)/2 && (= c_3 + c_1),\\
&S_1 := S_1 - S_0 && (= c_4 + c_3 + c_2 + c_1),\\
&S_2 := (S_2 - S_1)/2 && (= 2c_4 + c_3),\\
&S_1 := S_1 - S_3 - S_4 && (= c_2),\\
&S_2 := S_2 - 2S_4) && (= c_3),\\
&S_3 := S_3 - S_2 && (= c1).
\end{aligned}$$

Slijedi:

$$C = A \cdot B = S_4x^4 + S_2x^3 + S_1x^2 + S_3x + S_0.$$

Ovaj algoritam zahtijeva samo jedno množenje brojem 2, dok prethodni algoritam zahtijeva dva.

Primjer 3.2.2. *Uz pretpostavku skrivene konstante* $= 1$*, računanje umnoška dva broja sa* 10^6 *znamenaka bi, po školskom algoritmu množenja zahtjevalo otprilike* 10^{12} *operacija. Današnji procesori po svakoj jezgri izvršavaju oko* $5 \cdot 10^9$ *operacija u sekundi, što znači da bi jednoj jezgri procesora trebalo oko* 200 *sekundi za izračun umnoška, a koristeći Toom-Cook-3, Uz pretpostavku skrivene konstante* $= 2$*, računanje umnoška dva milijun-znamenkasta broja će zahtijevati* $\approx 2 \cdot (10^6)^{1.465} \approx 1.23 \cdot 10^9$ *računskih operacija, što će današnje računalo izvesti u otprilike* 1 *sekundi.*

Kvadriranje razdjeljivanjem znamenaka u 3 dijela

Računamo kvadrat $C = A^2$ broja A.

$$\begin{aligned}
A &= a_2x^2 + a_1x + a_0,\\
C = A^2 &= S_4x^4 + S_3x^3 + S_2x^2 + S_1x + S_0.
\end{aligned}$$

Postavimo

$$\begin{aligned}
S_0 &:= a_0^2,\\
S_1 &:= (a_2 + a_1 + a_0)^2,\\
S_2 &:= (a_2 - a_1 + a_0)^2,\\
S_3 &:= 2a_1 \cdot a_2,\\
S_4 &:= a_2^2.
\end{aligned}$$

Ovaj postupak zahtijeva 4 kvadriranja i jedno množenje brojeva duljine $N/3$. S obzirom da su S_0, S_3 i S_4 izračunati, S_1 i S_2 računamo prema:

$$\begin{aligned} T_1 &:= (S_1 + S_2)/2, \\ S_1 &:= S_1 - T_1 - S_3, \\ S_2 &:= T_1 - S_4 - S_0. \end{aligned}$$

Množenje razdjeljivanjem znamenaka u 4 dijela

Elegantan način za razdjeljivanje znamenki broja u 4 dijela su također ponudili Bodrato i Zanoni [6]. Taj algoritam se izvodi u vremenu $O(n^{\log_4 7}) \approx O(n^{1.403})$. Općenitije, s-way splitting shema će se izvoditi u $O(n^{f(s)})$ vremenu gdje je $f(s) = \log_s(2s+1)$. Napomena: $\lim\limits_{x\to+\infty} \log_x(2x+1) = 1$.

Slično kao u prethodnom poglavlju, neka je

$$\begin{aligned} A &= a_3x^3 + a_2x^2 + a_1x + a_0, \\ B &= a_3x^3 + a_2x^2 + a_1x + a_0. \end{aligned}$$

Postavimo

$$\begin{aligned} S_1 &= a_3 \cdot b_3, \\ S_2 &= (8\cdot a_3 + 4\cdot a_2 + 2\cdot a_1 + a_0)\cdot(8\cdot b_3 + 4\cdot b_2 + 2\cdot b_1 + b_0), \\ S_3 &= (a_3 + a_2 + a_1 + a_0)\cdot(b_3 + b_2 + b_1 + b_0), \\ S_4 &= (-a_3 + a_2 - a_1 + a_0)\cdot(-b_3 + b_2 - b_1 + b_0), \\ S_5 &= (8\cdot a_0 + 4\cdot a_1 + 2\cdot a_2 + a_3)\cdot(8\cdot b_0 + 4\cdot b_1 + 2\cdot b_2 + b_3), \\ S_6 &= (-8\cdot a_0 + 4\cdot a_1 - 2\cdot a_2 + a_3)\cdot(-8\cdot b_0 + 4\cdot b_1 - 2\cdot b_2 + b_3). \end{aligned}$$

Koristeći prethodno postavljene jednakosti računamo redom:

$$\begin{aligned} S_2 &:= S_2 + S_5, \\ S_4 &:= S_4 - S_3, \\ S_6 &:= S_6 - S_5, \\ S_4 &:= S_4/2, \end{aligned}$$

$$
\begin{aligned}
S_5 &:= S_5 - S_1,\\
S_5 &:= S_5 - (64 \cdot S_7),\\
S_3 &:= S_3 + S_4,\\
S_5 &:= 2 \cdot S_5,\\
S_5 &:= S_5 + S_6,\\
S_2 &:= S_2 - (65 \cdot S_3),\\
S_3 &:= S_3 - S_1,\\
S_3 &:= S_3 - S_7,\\
S_4 &:= -S4,\\
S_6 &:= -S6,\\
S_2 &:= S_2 + (45 \cdot S_3),\\
S_5 &:= S_5 - (8 \cdot S_3),\\
S_5 &:= S_5/24 \quad \text{(dijeljenje s 24)},\\
S_6 &:= S_6 - S_2,\\
S_2 &:= S_2 - (16 \cdot S_4),\\
S_2 &:= S_2/18 \quad \text{(dijeljenje s 18)},\\
S_3 &:= S_3 - S_5,\\
S_4 &:= S_4 - S_2,\\
S_6 &:= S_6 + (30 \cdot S_2),\\
S_6 &:= S_6/60 \quad \text{(dijeljenje sa 60)},\\
S_2 &:= S_2 - S_6.
\end{aligned}
$$

Slijedi:

$$C = S_1x^6 + S_2x^5 + S_3x^4 + S_4x^3 + S_5x^2 + S_6x + S_7.$$

Složenost ostalih algoritama

Karatsubin algoritam je predstavljao inspiraciju i prototip je svih kasnijih algoritama za brzo množenje što se najprije odnosi na Strassenov algoritam za množenje matrica iz 1969. godine [19], koji u svojoj biti primjenjuje Karatsubin algoritam množenja prirodnih brojeva na množenje elemenata matrica. Množenje matrica školskim (naivnim) algoritmom je složenosti $O(n^3)$ operacija, a Strassen je uspio postići $O(n^{\log_2 7}) \approx O(n^{2.807})$. Postoje poboljšanja koja nisu tako značajna. Najbolji trenutni rezultat je $O(n^{2.37188})$ koji je galaktički algoritam [9].

Kao što je već spomenuto, razdjeljivanjem broja u više dijelova, tj. zapisujući ga u obliku

$$a = a_0 + 2^m \cdot a_1 + 2^2 m \cdot a_2 + \cdots + 2^r m \cdot a_r,$$

gdje su $a_0, a_1, \ldots, a_r$ m-znamenkasti brojevi i $rm = n$. Slijedi:

$$a^2 = \left(\sum_{j=0}^{r} a_j \cdot 2^{mj} \right)^2 = \sum_{s=0}^{2r} c_s \cdot 2^{ms},$$

gdje je

$$c_s = \sum_{\substack{j+v=s \\ 0 \le j,v \le r}} a_j a_v.$$

Koeficijenti c_s proizlaze iz sljedećeg sustava jednadžbi:

$$(a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_r \cdot x^r)^2 = \sum_{s=0}^{2r} c_s x^s,$$

gdje možemo postaviti $x = 0, \pm 1, \ldots, 2r$. Izabirom optimalne vrijednosti za r, može se odrediti drugačiji $M(n)$. Na ovaj način su procjenu za $M(n)$ poboljšali Toom [21], Cook [7] i Schönhage 1966. godine [17]. Složenost tih poboljšanih algoritama je oblika $M(n) = O(ne^{c\sqrt[c]{\log n}})$, gdje je $c > 0$ konstanta [15]. Valja napomenuti da je Schönhage za svoj algoritam koristio modularnu aritmetiku što je rezultiralo otpisivanjem konstante. Inačica D. Knuthova algoritma je složenosti $O(n \cdot 2^{\sqrt{2 \log n}} \cdot \log n)$ [15].

Poglavlje 4

FFT Algoritmi

4.1 Brza Fourierova transformacija

Svijet algoritama je velik, ali ga možemo podijeliti u dvije skupine, oni koji su korisni i efikasni, koje zbog toga želimo razumjeti i oni koji su jednostavno lijepi, kao na primjer algoritam slaganja Hanojskih tornjeva [10] koji inspirira i potiče na razmišljanje. Algoritam Brze Fourierove transformacije (FFT) s pravom pripada u obje skupine, a jedan je od najznačajnijih algoritama i otkriven je u 20. stoljeću. Tehnologija koju koristimo danas se temelji na FFT, kao na primjer: bežična komunikacija, GPS, i sve što se odnosi na procesiranje signala koristi ovaj algoritam, a neke od najčešćih primjena FFT-a su tehnike komprimiranja digitalnih audio i video sadržaja. S druge strane, to je jedan od najljepših algoritama. Često se ne obaziremo na ljepotu ovog algoritma jer se uvodi kod složenih primjena i zahtijeva široko predznanje.

Definicija 4.1.1 (Polinom)**.** *Polinom u varijabli x nad algebarskim poljem F predstavljen je funkcijom $A(x)$ kao suma:*

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

Vrijednosti a_0, a_1, ..., a_{n-1} su koeficijenti polinoma, $a_j \in F$, $x \in \mathbf{C}$, $j = 1, \ldots, n-1$. Polinom $A(x)$ je stupnja k ako je vodeći koeficijent $a_k \neq 0$ i u tom slučaju pišemo $deg(A) = k$.

Neka su zadana dva polinoma $A(x)$ i $B(x)$

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k,$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_k x^k,$$

tada je umnožak

$$C(x) = A(x) \cdot B(x)$$
$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{2k} x^{2k}.$$

Umnožak dva polinoma k-tog stupnja će biti najviše stupnja $2k$. Školski algoritam za množenje dva polinoma, korištenjem svojstva distributivnosti, je složenosti $O(k^2)$ jer svaki koeficijent polinoma A treba pomnožiti svakim koeficijentom polinoma B. Način na koji možemo zapisati umnožak $C(x)$ je

$$C(x) = \sum_{i=0}^{2n-2} c_i x^i,$$

gdje je

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}.$$

Definicija 4.1.2. *Prikaz polinoma* $A(x) = \sum_{i=0}^{n-1} a_i x^i$ *pomoću koeficijenata je vektor* $a = (a_0, a_1 \ldots, a_{n-1})$.

Takav prikaz je prikladan za evaluaciju polinoma $A(x)$ u točki x_0. Za evaluaciju polinoma $A(x_0)$ koristi se Hornerovo pravilo, odnosno algoritam koji je složenosti $\Theta(n)$:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1})))).$$

Definicija 4.1.3. *Neka su f i g nizovi koeficijenata dva polinoma stupnja $N-1$, takvi da je $f[n]$ koeficijent uz x^n. Konvolucija f i g, u oznaci $f * g$ se definira kao*

$$(f * g)[n] = \sum_{m=0}^{N-1} f[m] g[n-m].$$

S obzirom da svaki prirodan broj može biti vrijednost nekog polinoma u nekoj točki (npr. $275 = 2x^2 + 7x + 5$, za $x = 10$), množenje prirodnih brojeva se može promatrati i kao konvolucija znamenki tih brojeva.

Definicija 4.1.4. *Prikaz polinoma pomoću točaka i odgovarajućih vrijednosti $A(x)$ stupnja $n-1$ je skup n uređenih parova*

$$\{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\},$$

takvih gdje su svi x_k različiti i

$$y_k = A(x_k), \tag{4.1}$$

za $k = 0, 1, \ldots, n-1$.

Prikaz polinoma pomoću koeficijenata i pomoću vrijednosti u dovoljnom broju točaka je u nekom smislu ekvivalentan jer svaki polinom za kojeg su poznate vrijednosti u dovoljnom broju točaka ima jedinstven prikaz pomoću koeficijenata.

Polinom ima više različitih prikaza pomoću točke i odgovarajuće vrijednosti s obzirom da bilo koji skup n različitih točaka $x_0, x_1, \ldots, x_{n-1}$ može poslužiti kao baza za takav prikaz.
Ako su poznati koeficijenti polinoma, računanje n različitih vrijednosti polinoma pomoću Hornerovog algoritma je složenosti $\Theta(n^2)$.
Inverzna operacija od evaluacije polinoma, tj. određivanje koeficijenata polinoma, ako je poznat prikaz polinoma pomoću točke i određene vrijednosti, se zove interpolacija.

Teorem 4.1.5. *Svaki skup* $\{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ *n točaka i odgovarajućih vrijednosti takvih da su svi* x_k *različiti predstavlja jedinstven polinom* $A(x)$ *stupnja* $n-1$ *gdje je* $y_k = A(x_k)$ *za* $k = 0, \ldots, n-1$.

Dokaz. Dokaz se oslanja na postojanje inverza određenih matrica. Jednadžba (4.1) je ekvivalentna jednadžbi

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}. \tag{4.2}$$

Matrica lijevo, zapisana kao $V(x_0, x_1, \ldots, x_{n-1})$ je poznata pod nazivom Vandermondeova matrica i ima determinantu

$$\prod_{0 \le j < k \le n-1} (x_k - x_j),$$

što znači da je invertibilna ako su svi x_k različiti. Koeficijenti a_j slijede računanjem inverza Vandermondeove matrice:

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y.$$

□

U dokazu iz formule (4.2) slijedi algoritam koji se temelji na računanju sustava jednadžbi (4.2), a ima složenost $O(n^3)$. Brži algoritam za interpolaciju n točaka se temelji na Lagrangeovoj interpolacijskoj formuli:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

i ima složenost $\Theta(n^2)$ [8].

Slijedi da su evaluacija polinoma u n-točaka i interpolacija dobro definirane inverzne operacije koje pretvaraju iz prikaza polinoma pomoću koeficijenata u prikaz polinoma pomoću točaka i odgovarajućih vrijednosti. Algoritmi za rješavanje tih problema imaju složenost $\Theta(n^2)$.

Prikaz polinoma pomoću točaka i odgovarajućih vrijednosti je pogodan za množenje polinoma. Ako je $C(x) = A(x)B(x)$ tada je $C(x_k) = A(x_k)B(x_k)$ za svaku točku x_k i kako bi dobili prikaz polinoma C pomoću točaka i odgovarajućih vrijednosti dovoljno je pomnožiti vrijednosti polinoma A i B u istim točama. S obzirom da je $deg(C) = deg(A) + deg(B)$ i ako pretpostavimo da je $deg(A) = deg(B) = n$, tada je $deg(C) = 2n$. Množenjem polinoma A i B pomoću njihovog prikaza odgovarajućih točaka i vrijednosti se dobije n točaka i odgovarajućih vrijednosti, što znači da je za prikaz polinoma C pomoću točaka i odgovarajućih vrijednosti, polinome A i B potrebno prikazati u ukupno $2n$ točaka, tj. ako je zadan polinom A kao

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\},$$

i polinom B kao

$$\{(x_0, y_0'), (x_1, y_1'), \ldots, (x_{2n-1}, y_{2n-1}')\},$$

tada je prikaz polinoma C pomoću točaka i odgovarajućih vrijednosti

$$\{(x_0, y_0y_0'), (x_1, y_1y_1'), \ldots, (x_{2n-1}, y_{2n-1}y_{2n-1}')\}.$$

Dakle, množenje dva polinoma, ako su oni zadani u proširenom prikazu pomoću $2n$ točaka i odgovarajućih vrijednosti, je složenosti $\Theta(n)$, što je puno efikasnije od složenosti $\Theta(n^2)$ kod množenja polinoma u prikazu pomoću koeficijenata.

Polinome A i B se može vrednovati u bilo kojim točkama, ali neke točke omogućuju manju složenost kod pretvorbe između navedena dva prikaza.

Definicija 4.1.6. *Kompleksni n-ti korijen jedinice je komplekni broj ω takav da je $\omega^n = 1$. Postoji točno n korijena jedinice: $e^{2\pi ik/n}$, $k = 0, 1, \ldots, n-1$, što slijedi iz definicije kompleksnog broja $e^{iu} = \cos(u) + i\sin(u)$.*

Definicija 4.1.7. *Glavni n-ti korijen jedinice je*

$$\omega_n = e^{2\pi i/n}, \tag{4.3}$$

a svi ostali kompleksni korijeni jedinice su potencije ω_n.

Tih n kompleksnih n-tih korijena jedinice, $\omega_n^0, \omega_n^1, \ldots, \omega^{n-1}$ čini grupu s obzirom na množenje koja ima istu strukturu kao $(\mathbb{Z}_n, +)$ budući da iz $\omega_n^n = \omega_n^0 = 1$ slijedi $\omega_n^j\omega_n^k = \omega_n^{j+k} = \omega_n^{(j+k) \mod n}$. Također, vrijedi $\omega_n^{-1} = \omega_n^{n-1}$.

Lema 4.1.8. *Za sve $n, k, d \in \mathbf{N}$, takve da $n > 0$, $k \geq 0$ i $d > 0$ vrijedi*

$$\omega_{dn}^{dk} = \omega_n^k. \tag{4.4}$$

Dokaz. Lema 4.1.8 slijedi iz (4.3), jer $\omega_{dn}^{dk} = (e^{2\pi i/dn})^d k = (e^{2\pi i/n})^k = \omega_n^k$ □

Lema 4.1.9. *Ako je n paran, tada su kvadrati n kompleksnih n-tih korijena jedinice $(n/2)$-ti korijeni jedinice, a ima ih $n/2$.*

Dokaz. Iz Leme 4.4 slijedi $(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$. □

Lema 4.1.10. *Za sve prirodne brojeve n i k, $k \nmid n$*

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Dokaz. $\displaystyle\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0.$ □

Definicija 4.1.11. *Neka je zadan polinom A u prikazu pomoću koeficijenata, tj. vektor $a = (a_0, a_1, \ldots, a_{n-1})$ i neka je y_k za $k = 0, 1, \ldots, n-1$ zadan sa*

$$\begin{aligned} y_k &= A(\omega_n^k) \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{kj}. \end{aligned} \tag{4.5}$$

Vektor $y = (y_0, y_1, \ldots, y_{n-1})$ je diskretna Fourierova transformacija (DFT) vektora koeficijenata a. Pišemo $y = DFT_n(a)$.

Brza Fourierova transformacija (FFT) koristi svojstva kompleksnih korijena jedinice za izračun $DFT_n(a)$ i složenosti je $\Theta(n \log n)$. Pretpostavimo da je n potencija broja 2. FFT koristi strategiju *podijeli pa vladaj* koristeći zasebno parno i neparno indeksirane koeficijente $A(x)$ na način da se definiraju dva nova polinoma

$$\begin{aligned} A^{parni}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}, \\ A^{neparni}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}. \end{aligned}$$

Slijedi

$$A(x) = A^{parni}(x^2) + x A^{neparni}(x^2) \tag{4.6}$$

tako da se problem evaluacije $A(x)$ u točkama $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ svodi na

1. evaluaciju polinoma $A^{parni}(x)$ i $A^{neparni}(x)$ u točkama

$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2 \tag{4.7}$$

2. spajanje rezultata prema (4.6).

Koristeći Lemu 4.1.9, slijedi da u (4.7) nisu n različitih vrijednosti već $n/2$ kompleksnih $(n/2)$-tih korijena jedinice od kojih se svaki korijen pojavljuje točno dvaput. Na ovaj način FFT rekurzivno vrednuje polinome A^{parni} i $A^{neparni}$ stupnja $n/2$ u $n/2$-tih korijena jedinice. Manji problemi su istog oblika kao i glavni problem, ali upola manji, tj. dijeleći računanje DFT_n u računanje dvije $DFT_{n/2}$, a DFT jednog elementa je upravo taj element jer $y_0 = a_0\omega_1^0 = a_0 \cdot 1 = a_0$.

Za odrediti složenost opisanog FFT algoritma primijetimo da je složenost svakog rekurzivnog koraka $\Theta(n)$, gdje je n duljina ulaznog vektora. Slijedi da je složenost ovako zadane rekurzije

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \log n). \end{aligned}$$

Dakle, evaluacija polinoma stupnja $n-1$ u n točaka koje su n-ti korijeni jedinice je složenosti $\Theta(n \log n)$.

Algoritam množenja polinoma zahtijeva pretvaranje iz prikaza pomoću koeficijenata u prikaz pomoću točaka i odgovarajućih vrijednosti, zatim množenje odgovarajućih vrijednosti u točkama korijena jedinice i zatim pretvaranje iz prikaza pomoću točaka i odgovarajućih vrijednosti nazad u prikaz pomoću koeficijenata koristeći interpolaciju. Kako bi interpolirali polinom u točkama korijena jedinice, koristeći (4.2) zapišimo najprije DFT kao matričnu jednadžbu:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}. \tag{4.8}$$

U matrici je (k,j)-ti element V_n jednak ω_n^{kj} za $j,k = 0,1,\ldots,n-1$. Eksponenti elemenata V_n čine tablicu množenja za faktore od 0 do $n-1$.

Inverznu operaciju od DFT_n, koju zapisujemo $a = DFT_n^{-1}(y)$ dobivamo tako da y pomnožimo matricom V_n^{-1}, tj. inverzom matrice V_n.

Teorem 4.1.12. *Za $j,k = 0,1,\ldots,n-1$, (j,k)-ti element V_n^{-1} je ω_n^{-jk}/n.*

Dokaz. Kako bi dokazali da vrijedi $V_n^{-1}V_n = I_n$, promotrimo (k, k')-ti element $V_n^{-1}V_n$:

$$\begin{aligned}[V_n^{-1}V_n]_{kk'} &= \sum_{j=0}^{n-1} (\omega_n^{-jk}/n)(\omega_n^{jk'}) \\ &= \sum_{j=0}^{n-1} \omega_n^{j(k'-k)}/n. \end{aligned} \tag{4.9}$$

Suma (4.9) je jednaka 1 ako je $k' = k$, a u suprotnom, koristeći Lemu 4.1.10, je 0. Treba vrijediti da $(k' - k) \nmid n$, što vrijedi jer $-(n-1) \leq k' - k \leq n - 1$. □

Nakon što je definirana inverzna matrica V_n^{-1}, $DFT_n^{-1}(y)$ je zadan sa:

$$\begin{aligned} a_j &= \sum_{k=0}^{n-1} y_k \frac{\omega^{-jk}}{n} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj} \end{aligned} \tag{4.10}$$

Uspoređujući (4.5) i (4.10) se može vidjeti da je za izračun y i a pomoću FFT algoritma dovoljno zamijeniti ω_n sa ω_n^{-1} i podijeliti svaki element sa n, tj. složenost DFT_n^{-1} je također $\Theta(n \cdot \log n)$.

Dakle, algoritmi FFT i inverzni FFT pružaju način za transformaciju polinoma između prikaza pomoću koeficijenata i prikaza pomoću točaka i odgovarajućih vrijednosti i složenosti su $\Theta(n \cdot \log n)$ jer se umnožak dva polinoma f i g može zapisati kao

$$fg = DFT^{-1}(DFT(f) \cdot DFT(g)),$$

odnosno u tri koraka:

1. Upotrijebiti FFT za izračun $F := DFT(f)$ i $G := DFT(g)$ složenosti $O(n \cdot \log n)$,

2. $DFT(f * g)[k] = F[k] \cdot G[k]$ za množenje odgovarajućih vrijednosti po točkama iz prethodnog koraka, složenosti $O(n)$,

3. upotrijebiti FFT za izračun inverza DFT iz koraka 2., složenosti $O(n \cdot \log n)$.

što ukupno daje algoritam složenosti $O(n \cdot \log n)$.

4.2 Schönhage Strassen algoritam

Arnold Schönhage i Volker Strassen su 1971. godine osmislili algoritam za množenje prirodnih brojeva koji se izvodi u vremenu $O(n \cdot \log n \cdot \log(\log n))$ koji je za veliki n značajno brži od Karatsubinog algoritma. Zbog specifičnosti implementacije u računalima kod računanja umnožaka proizvoljno velikih brojeva pojavljuje se $\log \log n$ kada rekurzija postane dubine $\log(\log n)$ [18].

Nedostatak FFT algoritma iz prethodnog poglavlja je što koristi realne, odnosno kompleksne brojeve, a kod množenja prirodnih brojeva to se može izbjeći korištenjem NTT (Number theoretic transform), tj. modificiranim DFT algoritmom. Kao što je opisano, DFT algoritam koristi n-te korijene jedinice $\omega = e^{2\pi j/n}$, gdje je $\omega^k \neq 1$ za $1 \leq k < n$ i $\omega^n = 1$, a ključna ideja Schönhage Strassen algoritma je korištenje primitivnih korijena koji proizlaze iz modularne aritmetike jer se sve operacije nad brojevima provode *modulo p* gdje je p prost broj. Kod NTT se ω definira kao prirodan broj takav da $\omega^N \equiv 1 \pmod{M}$ i $\omega^k \equiv 1 \pmod{M}$ za sve $k \in [1, 2, \ldots, N\rangle$ iz čega proizlazi $\omega^k \not\equiv 1 \pmod{M} \iff k \equiv 0 \pmod{N}$ [20].

Na primjer, 2 je 5-ti primitivni korijen *modulo* 31 jer $2^1, 2^2, 2^3, 2^4 \not\equiv 1 \pmod{31}$, ali $2^5 \equiv 1 \pmod{31}$.

Analogno DFT algoritmu, umnožak polinoma f i g stupnja n prema ideji Schönhage Strassena se izvodi kao

$$fg = NTT^{-1}(NTT(f) \cdot NTT(g)).$$

Osnovni rekurzivni problem se svodi na množenje u $\mathbb{Z}/(2^n + 1)\mathbb{Z}$ gdje je n potencija broja 2. Postavimo $n' = 2^{\lceil (\log_2 2n/2) \rceil} = \Theta(n^{1/2})$ i $T = 2n/n' = \Theta(n^{1/2})$ tako da $(n')^2 \in \{2n, 4n\}$ i $T \mid n'$ tada dijeleći ulazne parametre u dijelove veličine $n'/2$ je problem množenja reduciran na množenje nad poljem $R[x]/(x^T + 1)$, gdje je $R = \mathbb{Z}/(2^{n'+1})\mathbb{Z}$. Potencije broja 2 u R se ponekad zovu "sintetički" ili "brzi" korijeni jedinice zato što omogućuju množenje elemenata iz R potencijom proizvoljne potencije broja 2 složenosti $O(n')$ što ima za posljedicu da je za $\omega = 2^{n'/T}$ moguće evaluirati polinom u točkama $\omega, \omega^3, \ldots, \omega^{2T-1}$ (korijeni $x^T + 1$) pomoću FFT složenosti $O((n' \log n')n') = O(n \log n)$. Tako je problem množenja reduciran na T rekurzivnih množenja odgovarajućih vrijednosti u točkama polinoma nad R. Ako stavimo $M_1(n)$ za oznaku složenosti dobivanja umnoška u $\mathbb{Z}/(2^n + 1)\mathbb{Z}$ dobivamo

$$M_1(n) < \frac{2n}{n'} M_1(n') + O(n \cdot \log n), \quad n' = O(n^{1/2},). \tag{4.11}$$

što predstavlja smanjenje složenosti na $O(n^{1/2})$ u svakom rekurzivnom pozivu, a broj stupnjeva rekurzije na $\log_2 \log n + O(1) = O(\log(\log n))$ [18].

Pojednostavljena verzija Schönhage Strassen algoritma se nalazi u *GNU Multiple Precision Arithmetic Library*, paket koji se koristi u svim programima za izvođenje

aritmetičkih operacija [1] [12]. Doduše, za brojeve manje od nekoliko tisuća znamenki se koriste i druge tehnike, uključujući i Karatsubin algoritam [2].

4.3 Fürerov algoritam

Više od tri desetljeća, Schönhage Strassen je bio najbrži poznati algoritam za množenje prirodnih brojeva, ali nije izgledao elegantno. "Množenje je svakako složenije od zbrajanja, ali član $\log(\log n)$ možda nije potreban. Estetski gledano, ne izgleda lijepo, a fundamentalna operacija kao što je množenje bi trebala imati *lijepu* složenost." rekao je Martin Fürer u svom govoru na Sveučilištu u Philadelphiji. Svakako, nije bilo realno očekivati da se složenost množenja prirodnih brojeva svede na $O(n)$ jer bi to izjednačilo algoritam množenja i zbrajanja, ali su Schönhage i Strasen postavili slutnju da je moguće pronaći algoritam za množenje složenosti $O(n \cdot \log n)$ [11].

Definicija 4.3.1 (Iterirani logaritam)**.**

$$\log^* x = \begin{cases} 0 & \textit{ako je } x \leq 0, \\ \log^*(\log x) + 1 & \textit{ako je } x > 1. \end{cases} \tag{4.12}$$

Iterirani logaritam je vrlo sporo rastuća funkcija koja govori koliko puta se treba primijeniti uzastopno logaritmiranje nekog broja, odnosno dobivenog rezultata, kako bi se postigla vrijednost manja od 1.

Primjer 4.3.2. $\log^* 2 = 1$, $\log^* 4 = 2$, $\log^* 16 = 3$, $\log^* 65536 = 4$, $\log^* 2^{65536} = 5$

Martin Fürer je uspio pronaći algoritam za množenje prirodnih brojeva složenosti $O(n \cdot \log n K^{\log^* n})$, za neku konstantu $K > 1$ [11].

4.4 Algoritam Harveya i van der Hoevena

Martin Fürer je 2007. godine uspio smanjiti izraz $\log(\log n)$ iz Schönhage Strassen algoritma što je u matematičkoj javnosti bilo prepoznato kao iskorak, ali je njegov algoritam bio nepraktičan. Usprkos složenoj implementaciji, njegov doprinos je raspalmsao Harveya i van der Hoevena koji su proteklih godina napisali preko 10 radova daljnjim unaprijeđivanjem Fürerovog algoritma. Harvey u svojoj prezentaciji algoritma kaže da su napisali još dvostruko više radova koji nisu objavljeni jer su konstantno unaprijeđivali algoritam. Konačno, u ožujku 2019. godine, Harvey i van der Hoeven su pronašli način kako da u potpunosti izostave bilo kakvu konstantu u procjeni složenosti algoritma za množenje. Njihov algoritam koristi višedimenzionalnu

inačicu brze Fourierove transformacije u kombinaciji sa posebnom metodom koju su osmislili primjenjujući razne matematičke trikove kako bi u konačnici postigli algoritam za množenje prirodnih brojeva složenosti $O(n \cdot \log n)$. Ključna komponenta njihovog rada je Gaussova tehnika koja omogućuje redukciju problema množenja na višedimenzionalne diskretne Fourierove transformacije nad poljem kompleksnih brojeva, a koje transformacije se mogu evaluirati korištenjem Nussbaumerovih brzih polinomijalnih transformacija [13].
Taj algoritam je brži od svih dosadašnjih, ali samo za ekstremno velike brojeve, $n > 2^{1729^{12}}$, pa za sada nema praktičnu upotrebu, ali je važan zbog svojih teoretskih implikacija. Množenje je u srži gotovo svake matematičke operacije te iz njihovog rada primjerice slijedi da je moguće izračunati prvih n znamenaka recipročnog broja ili drugog korijena nekog broja pomoću algoritma složenosti $O(n \cdot \log n)$.

Bibliografija

[1] *FFT Multiplication (GNU MP 6.2.1)*, `https://gmplib.org/manual/FFT-Multiplication`, (siječanj 2023.).

[2] *Karatsuba Multiplication (GNU MP 6.2.1)*, `https://gmplib.org/manual/Karatsuba-Multiplication`, (siječanj 2023.).

[3] J. Arndt, *Matters Computational, Ideas, Algorithms, Source Code*, Nürnberg, 2010.

[4] S. Arora i B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, New York, 2007.

[5] J. Bennett, *How Many Particles Are in the Observable Universe?*, `https://www.popularmechanics.com/space/a27259/how-many-particles-are-in-the-entire-universe`, (sječanj 2023.).

[6] M. Bodrato, *Towards Optimal Toom–Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0*, WAIFI'07 **4547** (2007), 116 – 133.

[7] S. A. Cook, *On the Minimum Computation Time of Functions*, Ph D Thesis (1966).

[8] T. K. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein, *Introduction to algorithms*, MIT Press, Cambridge, Massachusetts London, England, 2022.

[9] R. Duan, H. Wu i R. Zhou, *Faster Matrix Multiplication via Asymmetric Hashing*, 2022.

[10] J. Erickson, *Algorithms*, Independent, 2019.

[11] Martin Fürer, *Faster integer multiplication*, SIAM J. Comput. **39** (2007), 979–1005.

[12] P. Gaudry, A. Kruppa i P. Zimmermann, *A GMP-based implementation of Schonhage-Strassen's large integer multiplication algorithm*, ISSAC 2007 (2007.), 167 – 174.

[13] D. Harvey i J. van der Hoeven, *Integer multiplication in time $O(n\log n)$*, Annals of Mathematics **193** (2021.), br. 2, 563 – 617.

[14] A. A. Karatsuba, *The Complexity of Computations*, Proceedings of the Stelkov Institute of Mathematics **211** (1995), br. 2, 169 – 189.

[15] D. E. Knuth, *The Art of Computer Programming: Seminumerical algorithms*, Addison-Wesley series in computer science and information processing, Addison-Wesley, 1981.

[16] R. J. Lipton i K. W. Reagan, *People, Problems, and Proofs*, Springer Berlin, Heidelberg, 2013.

[17] A. Schönehage, *Schnelle Multiplikation großer Zahlen*, Computing **1** (1966).

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

[19] V. Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik **13** (1969), 354–356.

[20] P. Sun, *FFT-Based Integer Multiplication, Part 2*, `https://psun.me/post/fft2/`, (siječanj 2023.).

[21] A. A. Toom, *On the Complexity of a Circuit of Functional Elements Realizing Multiplication of Integers*, Dokl. Akad. Nauk SSSR **150** (1963).

Sažetak

U ovom radu su predstavljeni algoritmi za množenje s naglaskom na složenost tih algoritama. Kroz stoljeća se pretpostavljalo da su svi algoritmi za množenje prirodnih brojeva složenosti $O(n^2)$, kao na primjer: egipatski algoritam, naivni (školski) algoritam i algoritam ruskog seljaka. Karatsuba je svojim otkrićem sredinom 60-tih godina 20. stoljeća otkrio brži algoritam složenosti $O(n^{1.58})$ što je pokrenulo istraživanja na tom području te su za manje od 10 godina Schönhage i Strassen otkrili algoritam složenosti $O(n \cdot \log n \cdot \log(\log n))$. Harvey i van der Hoeven su 2019. godine otkrili algoritam složenosti $O(n \cdot \log n)$ koji je trenutno najbrži algoritam i predstavlja trenutni vrhunac napretka u tom području.

Summary

In this thesis we present algorithms for integer multiplication with the focus on their complexity. Through centuries it was believed that the best complexity was $O(n^2)$, i.e. Egyptian multiplication, naive (schoolbook) multiplication and the Russian peasant multiplication algorithm. Karatsuba in the mid 1960's discovered an algorithm whose complexity is $O(n^{1.58})$ which started lots of research in this field, and in less than 10 years Schönhage and Strassen invented an algorithm whose complexity is $O(n \cdot \log n \cdot \log(\log n))$. In the year 2019. Harvey and der Hoeven invented an algorithm whose complexity is $O(n \cdot \log n)$ which is currently the fastest algorithm for integer multiplication and represents the current pinnacle of progress in the field.

Životopis

Rođen sam 27. studenog 1987. u Zagrebu. Djetinjstvo sam proveo u Velikoj Mlaki gdje sam pohađao osnovnu školu nakon koje sam upisao XV. Gimnaziju u Zagrebu. Preddiplomski sveučilišni studij Matematika; smjer: nastavnički sam upisao u srpnju 2006. godine, a diplomski sveučilišni studij Matematika i informatika; smjer: nastavnički sam upisao u srpnju 2016. godine.