Matrično množenje - rek. alg.

Klas alg ~ n^3 $\quad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Rek. - dijelj. na pola dalje za $n = 2^k$

$$T(n) = 8T(n/2) + c_1 n^2$$

tj. opet n^3.

Volker Strassen (1969)

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Algoritam:

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22}) B_{11}$$
$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$
$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$
$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

bitno je nekomutat!

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 + M_3 - M_2 + M_6$$

$$T(n) = \begin{cases} 7T(n/2) + c_2 \cdot n^2 & \text{za } n > 2 \\ \text{const (b)} & \text{za } n = 2 \end{cases}$$

7 množ, 18 zbraj. matrica (veda n/2).

Algoritam (Shmuel Winograd

$$M_1 = (A_{21} + A_{22} - A_{11}) \cdot (B_{22} - B_{12} + B_{11})$$
$$M_2 = A_{11} \cdot B_{11}$$
$$M_3 = A_{12} \cdot B_{21}$$
$$M_4 = (A_{11} - A_{21}) \cdot (B_{22} - B_{12})$$
$$M_5 = (A_{21} + A_{22}) \cdot (B_{12} - B_{11})$$
$$M_6 = (A_{12} - A_{21} + A_{11} - A_{22}) \cdot B_{22}$$
$$M_7 = A_{22} \cdot (B_{11} + B_{22} - B_{12} - B_{21})$$

$$C_{11} = M_2 + M_3$$
$$C_{12} = M_1 + M_2 + M_5 + M_6$$
$$C_{21} = M_1 + M_2 + M_4 - M_7$$
$$C_{22} = M_1 + M_2 + M_4 + M_5$$

7 množ, 24 zbroj. mat u/2.

Ali, ako se pamte među-rez. poput

$$M_1 + M_2 \quad i \quad M_1 + M_2 + M_4$$

$A_{11} - A_{22}$, $B_{11} - B_{12}$,
$A_{21} + A_{22}$ $B_{22} - B_{12}$

i sl. → broj zbr. se može spustiti na 15.

— Ima točno 36 načina (razl.) da se nađu C_{ij} i svi imaju 7 množ. (K. Glover)

Hopcroft - Kerr (1971) → treba bar 7 množ.

— 1978; Pan: 70×70 mat s 143640 množ → $n^{2.795}$

1986, Coppersmith, Winograd → $n^{2.376}$

$$(x-3)(x-2) = 0 .$$

Opće rješenje je:

$$t_k = c_1 \cdot 3^k + c_2 \cdot 2^k$$

ili

$$T(n) = c_1 \cdot 3^{\lg n} + c_2 \cdot n .$$

Za logaritme mijenja:

$$a^{\lg b} = b^{\lg a} = 2^{\lg b \cdot \lg a}$$

pa je:

$$\underline{T(n) = c_1 \cdot n^{\lg 3} + c_2 n \text{ , } n \text{ pot. od } 2} .$$

Dakle je:

$$\underline{T(n) = O(n^{\lg 3})} \text{ , } n \text{ pot. od } 2$$

Koeficijente iz rekurzije:

$$c_1 3^k + c_2 2^k = 3(c_1 3^{k-1} + c_2 2^{k-1}) + c \cdot 2^k \text{ , } k > 0$$

$$c_2 2^k = \frac{3}{2} c_2 \cdot 2^k + c \cdot 2^k$$

$$\frac{1}{2} c_2 = -c \text{ , } \underline{c_2 = -2c}$$

Dakle:

$$\underline{T(n) = c_1 \cdot n^{\lg 3} - 2c \cdot n} \text{ , } n \text{ pot. od } 2$$

Uz zahtjev $T(n) \geq 0$ za n pot. od 2, dobivamo uvjet

$$\underline{c_1 > 0}$$

pa je:

$$\underline{T(n) \sim c_1 n^{\lg 3}} \text{ , } n \text{ pot od } 2$$

Vrijedi: $\lg 3 = 1.58496\ldots$ ■

Skica rekurzija, javlja se kod jedne ubrzane varijante algoritma za množenje n znamenkastih cijelih brojeva. Standardnoj algoritam zahtijeva reda veličine n^2 množenja pojedinačnih znamenki. Ubrzana varijanta - tzv. Karacubin algoritam, zahtijeva reda veličine $n^{\lg 3}$ množenja pojedinačnih znamenki.

Množenje velikih brojeva - rekurzivni algoritam.

Pretpostavimo da su U, V veliki brojevi, dani pozicionim zapisom u pogodno odabranoj bazi b, tako da aritmetičke operacije na znamenkama možemo efikasno izvesti (direktno - arhitekturom računala)

Radi jednostavnosti, pretpostavimo da oba broja imaju točno n znamenki u bazi b (u protivnom, dopunimo sprijeda nulama do n). Neka je $n = 2k$ paran.

$$U = \sum_{i=0}^{n-1} u_i \cdot b^i = U_1 \cdot b^k + U_0 \quad , \quad U_1 \text{ i } U_0 \text{ po } n/2 = k \text{ znamenki}$$

$$U_1 = \sum_{i=0}^{k-1} u_{i+k} b^i \, , \quad U_0 = \sum_{i=0}^{k-1} u_i b^i$$

i analogno za V:

$$V = \sum_{i=0}^{n-1} v_i \cdot b^i = V_1 \cdot b^k + V_0 \, , \quad V_1 = \sum_{i=0}^{k-1} v_{i+k} b^i \, , \quad V_0 = \sum_{i=0}^{k-1} v_i \cdot b^i$$

Tj. U i V gledamo kao dvoznamenkaste brojeve u bazi $b^{n/2} = b^k$.

- Produkt $W = U \cdot V$ u toj bazi ima oblik:

$$W = (U_1 \cdot V_1) \cdot b^{2k} + (U_1 \cdot V_0 + U_0 \cdot V_1) \cdot b^k + U_0 \cdot V_0$$

tj. trebamo 4 produkta $n/2$ znamenkastih brojeva.

- Srednji faktor možemo zapisati u obliku:

$$U_1 V_0 + U_0 V_1 = (U_1 + U_0)(V_1 + V_0) - \underbrace{U_1 V_1}_{\text{već imamo}} - \underbrace{U_0 V_0}_{\text{već imamo}}$$

tj. za njega je potrebno još samo 1 množenje, ali $k+1$ znam. brojeva. No, to ne mijenja bitno kompleksnost.

- Dakle, W možemo naći s 3 produkta s $n/2$ znam. brojeva

$$\left.\begin{aligned} p &\leftarrow U_1 V_1 \\ q &\leftarrow U_0 V_0 \\ r &\leftarrow (U_1 + U_0)(V_1 + V_0) \end{aligned}\right\} \text{ova 3 produkta se računaju nerekurzivno, istim ovim alg.}$$

$$W \leftarrow p \cdot b^{2k} + (r - p - q) b^k + q \, .$$

(Karacuba, 1962).

Za W, treba još obaviti množenja s b^{2k}, b^k što se svodi na pomake, i zbrajanja/oduzimanja te normalizaciju. Sve ove operacije imaju trajanje $c \cdot n$ (linearno ovisno o duljini broja W, a ta je $\approx 2n$)

Razmotrimo pitanje optimalnosti algoritma NMUL tj., općenito, kako dokazati algoritma za množenje prirodnih brojeva.

Pošto nas zanima broj aritmetičkih operacija potrebnih za množenje 2 prirodna broja u pozicionom zapisu u bazi b, zgodno je uvesti oznake za taj broj, kao funkciju duljine (broja znamenki) brojeva koje množimo.

Definicija 2.7.

Neka je MUL bilo koji opći algoritam za množenje prirodnih brojeva $u, v \in \mathbb{N}$ u pozicionom zapisu u nekoj (može i svakoj) bazi b. Aritmetičku kompleksnost algoritma, u ovisnosti o duljini brojeva u, v, označavamo sa

$$Mul(n,m) = Compl_A(MUL),$$

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

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

Pri tome smatramo da je algoritam opći, ako radi korektno za sve brojeve $u, v \in \mathbb{N}$, tj. za proizvoljne duljine brojeva.

Ovo je bitan zahtjev, jer ako fiksiramo duljinu brojeva (ili raspon brojeva), onda problem trivijalan. Tada možemo unaprijed "konstruirati" "tablicu množenja" za danu duljinu (raspon), pa se množenje svodi na očitanje rezultata iz tablice.

[Fiksiranje baze b je dopušteno, jer čak i da konstruiramo tablicu množenja u bazi b, još uvijek je duljina brojeva proizvoljna, pa trajanje množenja ovisi o duljini - bar kao broj čitanja u tablici]

Uočimo da je na sekvencijalnim arhitekturama, $Mul(u,m)$ oko $Mul(n)$ dobra mjera potrebnog vremena.

Funkcija Mul, naravno, ovisi o algoritmu. No, sa stanovišta kompleksnosti, ona zapravo karakterizira algoritam za množenje.

[Za upotrebu nisu bitni detalji algoritma, već njegovo trajanje, tj. kompleksnost]

Osim toga, ta funkcija je vrlo korisna za zapis kompleksnosti svih algoritama koji koriste množenja. Zapis kompleksnosti tada ne ovisi o izboru algoritma za množenje i jasno je vidljiv odnos bitne kompleksnosti algoritma i kompleksnosti aritmetike kojom se on realizira. To je potrebno zbog toga što ćemo dati niz algoritama za množenje s bitno različitim kompleksnostima.

Uz ove oznake, relacije (2.115) i (2.116) za algoritam NMUL možemo zapisati u obliku

$$\begin{aligned} Mul(n,m) &= 5nm+2 \\ Mul(n) &= 5n^2+2, \end{aligned} \tag{2.164}$$

tj. broj operacija potrebnih za množenje n-znamenkastih prirodnih brojeva je proporcionalan s n^2.

Dalje je za bilo koji opći algoritam ispunjeno

$$Mul(n) = \Omega(n), \tag{2.165}$$

jer ulaz sadrži $2n$, općenito nerazumnih znamenki, na osnovu kojih treba postaviti $2n$ znamenki rezultata.

Ova dva rezultata pokazuju da je $Mul(n)$ negdje "između" n i n^2 (bližem asimptotski), tj. da ima prostora za bitno ubrzanje klasičnog algoritma.

Ubrzanju klasičnog algoritma pristupamo strategijom "podijeli pa vladaj" kojom problem reda veličine n svodimo

na neki niz istovrsnih problema, ali manje veličine. Taj postupak ponavljamo rekurzivno, sve dok veličina problema ne padne toliko da je problem lako rješiv.

[Ta metoda se nova mjesi obati ulrzanje!]

Ilustrirajmo ovaj postup.

Pretpostavimo, radi jednostavnosti, da je $e(u) = e(v) = n$ i da je n paran broj

$$n = 2k.$$

Zadala brojeve

$$u = \sum_{i=0}^{n-1} u_i b^i, \quad v = \sum_{i=0}^{n-1} v_i b^i, \tag{2.166}$$

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

$$u = U_1 b^k + U_0, \quad v = V_1 b^k + V_0, \tag{2.167}$$

gdje

$$U_1 = \sum_{i=0}^{k-1} u_{i+k} b^i = (u_{2k-1} \cdots u_k)_b$$

$$U_0 = \sum_{i=0}^{k} u_i b^i = (u_{k-1} \cdots u_0)_b,$$

tj. U_1 je "značajnija" ili "gornja" polovica, a U_0 je "donja" polovica broja U, i

analogno

$$V_1 = \sum_{i=0}^{k-1} v_{i+k} b^i = (v_{2k-1} \cdots v_k)_b$$

$$V_0 = \sum_{i=0}^{k} v_i b^i = (v_{k-1} \cdots v_0)_b.$$

Očito su U_i, V_i, $i = 0, 1$ znamenke brojeva u i v u bazi b^k, i zapis (2.167) je normaliziran u bazi b^k, ali je zapis (2.166) normaliziran u bazi b.

Produkt $w = u \cdot v$, prikazan u bazi b^k, ima oblik

$$w = (U_1 V_1) b^{2k} + (U_0 V_1 + U_1 V_0) b^k + U_0 V_0 . \quad (2.168)$$

Ovaj zapis se može lako normalizirati u bazi b^k, Navedene operacije i normalizaciju treba, naravno, provesti radeći na nizovima znamenki u bazi b.

Relacija (2.168) znači da je problem nalaženja produkta n-znamenkastih brojeva u i v, sveden na problem nalaženja produkata. $n/2 = k$-znamenkastih brojeva $(U_1 V_1, U_0 V_1, U_1 V_0, U_0 V_0)$ uz zbrajanja i umnoženja potencijom baze (pomake)

[Zbrajanje i pomaci $\Rightarrow$ linearno mjerne, pa promjena baze/duljine ne utječe na broj oper.]

Može se pokazati da relacija (2.168) ne poboljšava klasični algoritam, jer je

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

što daje

$$Mul(n) = \Theta(n^2)$$

(v. lema 2.2., u nastavku).

Vrijedi li da je (2.169

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

$$\text{NE} \Big[\quad w = (b^{2k} - b^k) U_1 V_1 + b^k (U_1 + U_0)(V_1 + V_0) + (1 - b^k) U_0 V_0$$

onda je dovoljno naći 3 (a ne više 4) produkta. $U_0 V_0$, $U_1 V_1$ su produkti k-znamenkastih, a $(U_1 + U_0)(V_1 + V_0)$ je produkt najviše $k+1$-znamenkastih brojeva

Relacije (2.168) i (2.169) daju složenost algoritma, kojeg prema u složenosti obliku.

[1.6.1]

Algoritam 2.28. (NMULKO – Karacuba (1962))

```
begin
  t1 ← (U1+U0)(V1+V0);
  t2 ← U0V0 ;
  t3 ← U1V1 ;
  w ← t3 · b^2k + (t1-t2-t3) b^k + t2 ;
end;
```

Ako pretpostavimo da množenje 3 množenja obavljamo rekurzivno, istim ovim algoritmom, za asimptotsku kompleksnost dobivamo

$$Mul(n) \leq 2Mul\left(\frac{n}{2}\right) + Mul\left(\frac{n}{2}+1\right) + c_1 n . \qquad (2.170)$$

Član $c_1 n$ predstavlja gornju ograđu broja operacija za zbrajanja i množenja potencijom baze, jer asimptotska složenost tih dviju operacija ovisi linearno o duljini brojeva.

Član $Mul\left(\frac{n}{2}+1\right)$ opisuje složenost množenja $(U_1+U_0)(V_1+V_0)$, jer ta dva broja mogu imati $n/2+1$ znamenki. Ako implementiramo na pamoj duljini brojeva, moramo dodati vodeće znamenke nula i zamijeniti $Mul(n/2+1)$ sa $Mul(n/2+2)$. Ova promjena ne utječe bitno na karakter relacije (2.170), jer unjedi složenosti.

Propozicija 2.23

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

$$Mul(n+t) \leq Mul(n) + Cn . \qquad (2.171)$$

C ovisi o t, ali ne i o n.

Dokaz:

Neka su $u, v \in \mathbb{N}$ $n+t$-znamenkasti brojevi. Zapišimo te brojeve u obliku

$$u = U_1 b^t + U_0 \quad , \quad v = V_1 b^t + V_0$$

gdje su U_1, V_1 n-znamenkasti, a U_0, V_0 t-znamenkasti brojevi.

Produkt $w = u \cdot v$ možemo zapisati u formi sličnoj relaciji (2.168)

$$w = U_1 V_1 \cdot b^{2t} + (U_0 V_1 + U_1 V_0) b^t + U_0 V_0 . \quad (2.172)$$

Produkt $U_1 V_1$ zahtjeva $Mul(n)$ aritmetičkih operacija.

Produkte $U_0 V_1$ i $V_1 U_0$ možemo naći tako da t puta ponovimo algoritam NMULD za množenje n-znamenkastog broja jednom znamenkom i rezultate zbrojimo, ali dvaput algoritmom NMUL. U oba slučaja potrebno je $c_1 nt$ aritmetičkih operacija, gdje c_1 neka konstanta.

Produkt $U_0 V_0$ zahtjeva $c_2 t^2$, ti konstantan broj aritmetičkih operacija, neovisno o n. Preostale operacije (zbrajanja i pomake) u (2.172) možemo obaviti za najviše $c_3(n+t)$ aritmetičkih operacija.

Ukupno je:

$$Mul(n+t) \leq Mul(n) + c_1 nt + c_2 t^2 + c_3(n+t).$$

No, tada je za $\forall u \in \mathbb{N}$

$$c_1 nt + c_2 t^2 + c_3(n+t) \leq (c_1 t + c_2 t^2 + c_3(t+1))n$$

pa postoji $C \in \mathbb{N}$ takav da vrijedi (2.170).

Ako fiksiramo donju granicu n_0 za n, onda se C može još bolje odabrati, a (2.170) vrijedi tada za sve $n \geq n_0$.

To je korisno u slučaju da za $n < n_0$ konstruiramo neki drugi algoritam množenja. ■

Primjenom ovog rezultata sa $t=1$ na relaciju (2.170), za kompleksnost algoritma Karacube izlazi

$$Mul(n) \leq 3Mul\left(\frac{n}{2}\right) + c_0 n, \qquad (2.173)$$

gdje je c_0 neka konstanta, koja ne ovisi o n. Relacija (2.173) vrijedi za $n > 1$, ako algoritam upotrebljavamo rekurzivno, sve dok ne dobijemo pojedinačne znamenke. Tada koristimo aritmetiku računala u algoritmu 2.28., pa je

$$Mul(1) \leq c_0'. \qquad (2.174)$$

Odaberemo li, radi jednostavnosti,

$$c = \max\{c_0, c_0'\},$$

dobivamo rekurzivne relacije:

$$Mul(1) \leq c$$
$$Mul(n) \leq 3Mul\left(\frac{n}{2}\right) + cn. \qquad (2.175)$$

Sljedeća lema omogućava rješenje ovih relacija.

Lema 2.2.

Neka je $T: \mathbb{N} \to \mathbb{R}$ monotono rastuća funkcija i neka su $a, d, c \in \mathbb{R}^+$ konstante takve da je $a > d$

Ako funkcija T zadovoljava rekurzivne jednadžbe

$$T(1) = c$$
$$T(n) = aT(n/d) + cn \quad , \; n = d^k, k > 0 \qquad (2.176)$$

onda je:

$$T(n) = \frac{c}{a-d}\left[an^{\log_d a} - dn\right] \qquad (2.177)$$

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

$$T(n) \leq C \cdot n^{\log_d a} \quad , \forall n \in \mathbb{N}. \qquad (2.178)$$

Dokaz:
Iz rekurzivnih relacija (2.176), indukcijom se lako dokazuje

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

Kako je

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

za $n = d^k$ izlazi (2.177). Relacija (2.178) je direktna posljedica monotonosti funkcije i pretpostavke $a > d$. ■

Funkcija Mul je očito monotona, jer rastuća brojeve dužine između $n/2$ i n množenjem dopunjuje vodećim nulama, do dužine n. Zbog toga, iz relacija (2.175) i leme 2.2.) za algoritam Karacube izlazi

$$Mul(n) < C \cdot n^{\log_2 3}, \quad n \in \mathbb{N} \qquad (2.179)$$

što je znatno ubrzanje obzirom na klasični algoritam, jer je

$$\log_2 3 \approx 1.585.$$

Napomena 2.33.
Precizna analiza složenosti rekurzivnih algoritama je izvanredno složena o detaljima realizacije, posebno o realizaciji rekurzije. Zbog toga smo proveli samo analizu reda veličine Mul(n). funkcije

Algoritam Karacube se može generalizirati tako da brojeve u i v rastavljamo na $r+1$ dijelova, a ne samo na 2 dijela.

Pretpostavimo, radi jednostavnosti, da je $n = (r+1)k$, gdje je $r \in \mathbb{N}$ neki fiksan broj.

Brojeve u i v možemo zapisati u bazi b^k u obliku

$$u = \sum_{i=0}^{r} U_i b^{ki}, \quad v = \sum_{i=0}^{r} V_i b^{ki}, \tag{2.180}$$

gdje su U_i, V_i, $i=0,\dots,r$ k-znamenkasti brojevi.

Definirajmo polinome $U(x)$, $V(x)$ sa

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

i njihov produkt

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

Posto je $u = U(b^k)$, $v = V(b^k)$, onda je

$$w = u \cdot v = W(b^k). \tag{2.181}$$

Ako nademo koeficijente W_i, $i=0,\dots,2r$ polinoma $W(x)$, onda je lako izračunati

$$w = \sum_{i=0}^{2r} W_i b^{ki} \tag{2.182}$$

koristeći zbrajanja i množenja potencijom baze. Ta faza zahtijeva broj operacija proporcionalan s $n = (r+1)k$.

Pretpostavimo da je polinom W potpuno određen bilo koeficijentima, bilo vrijednostima u $2r+1$ točaka.

Odaberimo točke $0, 1, \dots, 2r$. Tada je

$$W(i) = U(i)V(i), \quad i = 0,\dots,2r.$$

Iz ovih podataka, koristeći Newton-ov oblik interpolacionog polinoma (razmak točaka je 1)

$$W(x) = \sum_{i=0}^{2r} \frac{1}{i!} \Delta^i W(0) \cdot \prod_{j=0}^{i-1} (x-j), \tag{2.183}$$

lako nalazimo koeficijente W_i, koristeći Hornerom

slanu i polinomnu aritmetiku. Još je lakše $W = W(b^k)$ izračunati direktno iz (2.183).

[1.6.2]

Algoritam 2.29. (NMULRØ - "podijeli, pa vladaj")

```
begin
    { U(i), V(i) }
for i ← Ø to 2r do
    begin
    U(i) ← U_r;
    for j ← r-1 downto Ø do
        U(i) ← U(i)·i + U_j;
    V(i) ← V_r;
    for j ← r-1 downto Ø do
        V(i) ← V(i)·i + V_j;
    end;
    { W(i) = U(i)V(i) }
for i ← Ø to 2r do
    W(i) ← U(i)V(i)
    {tablica konačnih razlika Δ^i W(Ø) }
for i ← 1 to 2r do
    for j ← 2r downto i do
        W(j) ← (W(j) - W(j-1)) ;
    { w = W(b^k) }
w ← W(2r);
for i ← 2r-1 downto Ø do
    w ← (w b^k - w·i)/(i+1) + W(i)
end;
```

Naočito procjenu broja aritmetičkih operacija Mul(n), za $n = (r+1)k$, uz pretpostavku da za množenja $W(i) = U(i)V(i)$ koristimo rekurzivno opet isti algoritam.

Prvo treba odrediti duljinu brojeva $U(i)$, $V(i)$.

Posto je
$$U(i) = \sum_{j=0}^{r} U_j \, i^j \quad , \; i = 0, \dots, 2r,$$
gdje su U_j k-znamenkasti brojevi, to je $U_j < b^k$, $j = 0, \dots, r$. Zbog toga je
$$U(i) < b^k \cdot \sum_{j=0}^{r} i^j \quad .$$

Znamenke U_j (u bazi b^k) su nenegativne, pa brojevi $U(i)$ rastu po i.
Zbog toga je za $i = 0, \dots, 2r$
$$U(i) \leq U(2r) < b^k \, \frac{(2r)^{r+1} - 1}{2r - 1} \quad .$$

Broj r je konstanta, neovisna o n, pa postoji konstanta $t \in \mathbb{N}$, takva da je
$$U(i) < b^{k+t} \quad , \; i = 0, \dots, 2r \quad . \tag{2.184}$$

Isto vrijedi i za $V(i)$, $i = 0, \dots, 2r$.

Zbog toga, nalaženje $U(i), V(i)$, $i = 0, \dots, 2r$ homerovom shemom koristi operacije sa najviše $k+t$-znamenkastim brojevima.

Brojevi $i = 0, \dots, 2r$ rastu i imaju konstantnu duljinu ($< t$) i možemo čak pretpostaviti da je njihova duljina jednaka 1 (radi to nije bitno). Zbog toga svaka operacija za nalaženje $U(i), V(i)$ zahtijeva najviše
$$c_1 (k+t) \qquad \begin{matrix} \cdot, + \\ \text{za } U \text{ i za } V \end{matrix}$$
aritmetičkih operacija.
Takvih operacija je ukupno $(2r+1) \cdot \frac{2r}{4}$, pa je
$$\mathrm{Compl}_A \left(U(i), V(i), \, i = 0, \dots, 2r \right) \leq c_1 \cdot \frac{2r(2r+1)(k+t)}{4} \tag{2.185}$$

Za $2r+1$ produkata $W(i) = U(i)V(i)$ potrebno je umnožaka

$$(2r+1)\,Mul(k+t)$$

operacija. Koristeći propoziciju 2.23., dobivamo

$$Compl_A(W(i), i=\overline{0,2r}) \leq (2r+1)[Mul(k) + c_2 k]$$

Slijedi da je $2(k+t)$ $\leftarrow$ 16.01.98 – nije li to $2(k+t)-1$ (2.186).

Brojevi $W(i)$ imaju najviše $2(k+t)+1$ znamenki, direktno iz (2.184). Nalaženje tablice konačnih razlika zahtijeva $r(2r+1)$ oduzimanja, pa je

$$\frac{2r\cdot(2r+1)}{2} = r\cdot(2r+1)$$

$$Compl_A(\Delta^i W(0), i=\overline{0,2r}) \leq c_3 r(2r+1)(2k+2t+1) \quad (2.187)$$

Može se pokazati da primjenom oduzimanja ne dobivamo negativne brojeve

Računanje broja w koristi samo pomake, zbrajanja, te množenja i dijeljenja znamenkama (tj. brojevima konstantne duljine). Zbog toga je

duljina polja — 5 oper. — zbog je linearna u duljini brojeva

$$Compl_A(w) \leq c_4 \cdot 2r(2k+2t+1). \quad (2.188)$$

Zbrajanjem relacija (2.185) – (2.188) izlazi

jer r konst

$$Mul((r+1)k) \leq (2r+1)\,Mul(k) + c(r+1)k,$$

za neku konstantu c. Zbog toga je

$$Mul(n) \leq (2r+1)\,Mul\left(\frac{n}{r+1}\right) + c\cdot n.$$

Ako konstantu c odaberemo tako da je

$$Mul(1) \leq c,$$

lema 2.2. daje

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

Kako je

$$\log_{r+1}(2r+1) < 1 + \log_{r+1} 2,$$

dobivamo ocjenu aritmetičke kompleksnosti algoritma 2.29.

$$Mul(n) \leq C\, n^{1+\log_{r+1} 2}. \qquad (2.189)$$

Ovdje je konstanta C ovisna samo o r i bazi b.

Zbog

$$\log_{r+1} 2 \to 0 \text{ za } r \to \infty,$$

relacija (2.189) dokazuje sljedeći teorem:

Teorem 2.9.
Za bilo koji fiksni $\varepsilon > 0$, postoji algoritam množenja takav da je broj aritmetičkih operacija potrebnih za množenje dva n-znamenkasta broja u bazi b, manji od

$$Mul(n) \leq C(\varepsilon, b) \cdot n^{1+\varepsilon}, \quad \forall n \in \mathbb{N}, \qquad (2.190)$$

gdje je $C(\varepsilon, b)$ neka konstanta koja ovisi samo o ε i b, a ne ovisi o n. ■

Napomena 2.34. Iz analize kompleksnosti algoritma NMULRØ, vidi se da konstanta proporcionalnosti C vrlo brzo raste, kad povećavamo broj r (stupanj polinoma U, V). Zbog toga je taj algoritam efikasan samo za ogromne dulžine n. Za sve razumne dulžine n (pa čak i za računalu), algoritam je efikasan samo za vrlo male brojeve r ($r \lesssim 5$), a i tada n mora biti velik.

Za sve praktične potrebe dovoljni su klasični algoritam i algoritam Karacube.

[Malo poboljšanje je ako se uzmu točke
$-r, \ldots, 0, \ldots, r$
ali se tada mogu javiti i neg. mjeohusjh !]

strategija "podijeli, pa vladaj" koristeći polinomne reprezentacije i interpolaciju, može se, za dovoljno velike n, dobro aproksimirati za r, kako n raste.

Birajući $r \approx \sqrt{n}$, može se postići (algoritam Toom-Cook, v. [Knuth, 1981])

$$Mul(n) \leq C \cdot n^{1+3.5/\sqrt{\log_2 n}}, \quad n \in \mathbb{N},$$

a uz male modifikacije i

$$Mul(n) \leq C \cdot n^{1+\sqrt{2/\log_2 n}} \cdot \log n, \quad n \in \mathbb{N}.$$

<u>Napomena 2.35.</u> Najbrži poznati algoritam za množenje je algoritam Schönhage-Strassen (v. [Aho, Hopcroft, Ullman, 1976.]) za koji

$$Mul(n) \leq C\, n \log n \log\log n, \quad n \in \mathbb{N}.$$

Algoritam je baziran na sličnoj ideji, uz korištenje brze Fourierove transformacije i modularne aritmetike.

Najbolja donja ograda za <u>sve</u> algoritme množenja n-znamenkastih brojeva na klasičnim arhitekturama je

$$Mul(n) = \Omega(n \log n / \log\log n).$$

Zadnja dva rezultata daju pravo za široko rasprostranjeno vjerovanje da je

$$Mul(n) = \Theta(n \log n)$$

za optimalni algoritam množenja.