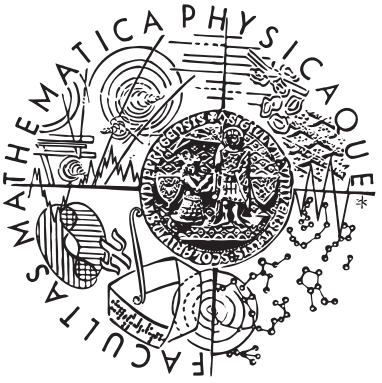
UNIVERZITA KARLOVA
MATEMATICKO-FYZIKÁLNÍ FAKULTA




BAKALÁRSKA PRÁCA

Andrea Frisová

# Schönhageove násobenie

KATEDRA ALGEBRY

Vedúci bakalárskej práce:
RNDr. David Stanovský, Ph.D.

Študijný program:
Matematika
Obecná matematika

Chcela by som poďakovať RNDr. Davidovi Stanovskému, Ph.D. za konzultácie a rady, ktorými mi pomohol túto prácu dokončiť.

Prehlasujem, že som svoju bakalársku prácu napísala samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 10.7.2006 Andrea Frisová

Názov práce: Schönhageove násobenie
Autor: Andrea Frisová
Katedra (ústav): Katedra algebry
Vedúci bakalárskej práce: RNDr. David Stanovský, Ph.D.
e-mail vedúceho: stanovsk@karlin.mff.cuni.cz

V predloženej práci sa zaoberám algoritmami na násobenie dlhých čísel. Popisujem tu tri rôzne algoritmy a to primitívny algoritmus, algoritmus Karacuba a Schönhageov-Strassenov algoritmus. Ich časové zložitosti sú $O(n^2)$, $O(n^{\log_2 3})$ a $O(n \log n \log\log n)$. Zameriavam sa na posledný z nich, ktorý má najlepšiu asymptotickú časovú zložitosť a snažím sa porovnať jeho skutočný čas výpočtu s časmi ostatných algoritmov v závislosti na dĺžke násobených čísel. Hlavným cieľom tejto práce je zhodnotiť efektívnosť tohto algoritmu v praxi, teda zistiť od akých veľkých čísel je rýchlejší ako algoritmus Karacuba.

Title: Schönhage multiplication
Author: Andrea Frisová
Department: Department of Algebra
Supervisor: RNDr. David Stanovský, Ph.D.
Supervisor's e-mail address: stanovsk@karlin.mff.cuni.cz

I study the multiplication algorithms of multiprecision numbers in this work. I show three different algorithms - the primitive one, Karacuba and Schönhage-Strassen algorithm. Their complexities are $O(n^2)$, $O(n^{\log_2 3})$ and $O(n \log n \log\log n)$, respectively. I compare Schönhage-Strassen algorithm with the others and I find the threshold when this algorithm becomes more effective than Karacuba.

# Obsah

KAPITOLA 1

# Primívny algoritmus a algoritmus Karacuba

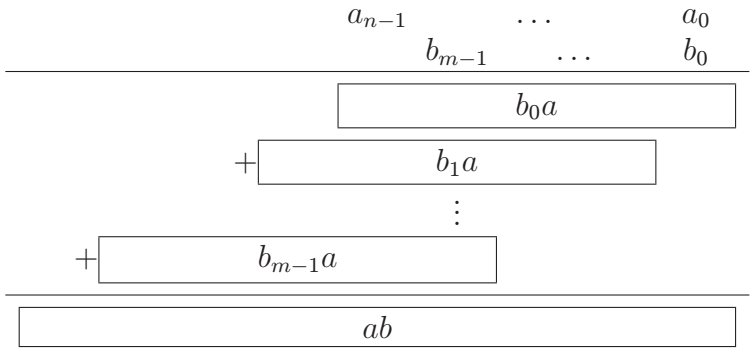
## 1. Primitívny algoritmus

Nech $a$ a $b$ sú čísla v sústave o základe $B$, potom ich môžeme zapísať následovne:

$$a = \sum_{i=0}^{n-1} a_i B^i, \quad b = \sum_{i=0}^{m-1} b_i B^i,$$

kde $a_i, b_i \in \{0, \ldots, B-1\}$.

Primitívny algoritmus je klasické „papierové“ násobenie. Schématicky ho môžeme znázorniť takto:




Pri násobení dvoch čísel násobíme $m$-krát číslo dĺžky $n$, urobíme teda $mn$ krokov. Na nasledujúce sčítanie potrebujeme ďalších $O(mn)$ operácií, teda celková časová zložitosť násobenia je $O(mn)$. Ak budeme predpokladať, že násobíme čísla rovnakej dĺžky $n$ (to kratšie môžeme vždy na začiatku doplniť nulami), potom je časová zložitosť $O(n^2)$.

## 2. Karacubove násobenie

Tento algoritmus na násobenie je založený na jednoduchej myšlienke a pritom je veľmi efektívny. Základný princíp spočíva v nasledujúcom: Nech $x, y$ sú dve čísla dĺžky $n$ v sústave o základe $B$, kde $n$ je párne číslo. Každé číslo rozdělíme na polovicu, teda

$$x = aB^{n/2} + b,$$
$$y = cB^{n/2} + d.$$

Potom $xy = (aB^{n/2} + b)(cB^{n/2} + d) = acB^n + (ad + bc)B^{n/2} + bd$. Potrebovali by sme zistiť 4 súčiny čísel dĺžky $n/2$ a niekoľko súčtov čísel délky $n$. Avšak pomocou jednoduchej úpravy dostávame:

$$xy = acB^n + (ad + bc)B^{n/2} + bd = acB^n + ((a + b)(c + d) - ac - bd)B^{n/2} + bd.$$

Teda násobíme už len trikrát čísla dĺžky $n/2$. Máme síce viac sčítaní, ale to nevadí, pretože sčítanie má menšiu zložitosť ako násobenie. A práve kvôli tomuto triku bude celková časová zložitosť algoritmu menšia než $O(n^2)$.

Algoritmus je založený na metóde „rozdeľuj a panuj“ a môžeme ho zapísať následovne:

ALGORITMUS ($KARACUBA(x, y)$).
VSTUP: $x, y$ celé čísla
VÝSTUP: $xy$
1. $n := max($dĺžka $x$, dĺžka $y)$
   Ak $n$ menšej dĺžky než nejaká konštanta, odpovedz $xy$,
   kde sa násobí pomocou primtívneho algoritmu; koniec
2. Ak $n$ je nepárne, potom $n := n + 1$
   Definuj $a$, $b$, $c$, $d$, aby platilo $x = aB^{n/2} + b$ a $y = cB^{n/2} + d$.
3. $u_1 := KARACUBA(a + b, c + d)$
   $u_2 := KARACUBA(a, c)$
   $u_3 := KARACUBA(b, d)$
4. Odpovedz $u_2B^n + (u_1 - u_2 - u_3)B^{n/2} + u_3$

Teraz potrebujeme zistiť časovú zložitosť algoritmu. Označme ju $T(n)$ pre vstup dĺžky $n$. Dostávame, že $T(1) = 1$ a $T(n) = 3T(n/2) + kn$ pro nejakú konštantu $k$ (tri násobenia čísel dĺžky $n/2$ a potom niekoľko sčítaní čísel dĺžky $n$). Budem predpokladať, že $n = 2^i$ pre nejaké $i$ celé. Rekurentným rozpísaním dostávame:

$$\begin{aligned}
T(2^i) &= 3T(2^{i-1}) + k2^i = 3(3T(2^{i-2}) + k2^{i-1}) + k2^i = 3^2T(2^{i-2}) + k2^i(\frac{3}{2} + 1) \\
&= 3^2(3T(2^{i-3}) + k2^{i-2}) + k2^i(\frac{3}{2} + 1) = 3^3T(2^{i-3}) + k2^i((\frac{3}{2})^2 + \frac{3}{2} + 1) \\
&= \cdots = 3^iT(2^{i-i}) + k2^i((\frac{3}{2})^{i-1} + \cdots + 1) = 3^iT(1) + k2^i\frac{(\frac{3}{2})^i - 1}{\frac{3}{2} - 1} \\
&= 3^i + 2k3^i - 2k2^i = n^{\log_2 3} + 2kn^{\log_2 3} - 2kn = O(n^{\log_2 3})
\end{aligned}$$

Takže máme algoritmus na násobenie, ktorý má časovú zložitosť $O(n^{\log_2 3})$.

KAPITOLA 2

# Pomocné tvrdenia

## 1. Čínska veta o zvyškoch

VETA 1 (Čínska veta o zvyškoch). *Nech $m_1, m_2, \ldots, m_n$ sú po dvoch nesúdeliteľné celé čísla, označme $N = m_1 m_2 \cdots m_n$. Potom pre $u_1, \ldots, u_n \in \mathbb{Z}$ existuje práve jedno riešenie $x \in \{0, 1, \ldots, N-1\}$, ktoré spĺňuje:*

$$\begin{aligned} x &\equiv u_1 \pmod{m_1}, \\ x &\equiv u_2 \pmod{m_2}, \\ &\vdots \\ x &\equiv u_n \pmod{m_n}. \end{aligned}$$

DÔKAZ. Najprv dokážem jednoznačnosť. Ak platí $x \equiv y \pmod{m_j}$ pre všetky $j = 1, 2, \ldots, n$, potom $x - y$ je násobok $m_j$ a $m_j$ sú nesúdeliteľné, teda $x - y$ je násobkom $N$. Súčasne platí, že $x - y \in \{-(N-1), \ldots, N-1\}$ a preto $x = y$.

Existencia bude vidieť z algoritmu, ktorý nasleduje.

□

Algoritmus, ktorý nájde $x$ z predchádzajúcej vety sa nazýva Garnerov algoritmus. Niektorá literatúra ho často uvádza aj pod názvom Newtonov algoritmus. Môžeme ho zapísať následovne:

ALGORITMUS (*GARNER*).

`VSTUP:` $m_1, \ldots, m_n \in \mathbb{Z}$, $u_1, \ldots, u_n \in \mathbb{Z}$

`VÝSTUP:` $x \in Z$ také, že $x \equiv u_i \pmod{m_i}$ pre $1 \leq i \leq n$, $0 \leq x < N$

`1.` $a_1 := u_1$

`i+1.` $a_{i+1} := (u_{i+1} - a_1 - a_2 m_1 - a_3 m_1 m_2 - \cdots - a_i m_1 m_2 \ldots m_{i-1}) c_i \bmod m_{i+1}$, kde $c_i$ je také, že $m_1 \ldots m_i c_i \equiv 1 \pmod{m_{i+1}}$

`n+1.` $x := a_1 + a_2 m_1 + \cdots + a_n m_1 \ldots m_{n-1} \bmod m_1 \ldots m_n$

Na otázku či tento algoritmus počíta to čo potrebujeme odpovedá nasledujúce tvrdenie.

TVRDENIE 2. *Garnerov algoritmus nájde $x$ také, že $x \equiv u_i \pmod{m_i}$ pre $i = 1, 2, \ldots, n$.*

DÔKAZ. Potrebujeme dokázať, že $x \equiv u_i \pmod{m_i}$ pre všetky $1 \leq i \leq n$. Pre $i = 1$ je to zrejmé z algoritmu. Nech $i > 1$. Označme

$$y_i = u_{i+1} - a_1 - a_2 m_1 - a_3 m_1 m_2 - \ldots - a_i m_1 m_2 \ldots m_{i-1}.$$

Potom platí

$$x = a_1 + a_2 m_1 + \ldots + a_n m_1 \cdots m_{n-1} = u_i - y_{i-1} + a_i m_1 \ldots m_{i-1} + \cdots + a_n m_1 \ldots m_{n-1}.$$

Teda stačí dokázať, že $a_i m_1 \ldots m_{i-1} - y_{i-1} \equiv 0 \pmod{m_i}$.

Z algoritmu vieme, že $a_i = y_{i-1}c_{i-1} \bmod m_i$, teda $a_i \equiv y_{i-1}c_{i-1} \pmod{m_i}$. Potom $a_i m_1 \dots m_{i-1} \equiv y_{i-1}c_{i-1}m_1 \dots m_{i-1} \pmod{m_i}$. Pretože $c_{i-1}m_1 \dots m_{i-1} \equiv 1 \pmod{m_i}$, tak platí $a_i m_1 \dots m_{i-1} \equiv y_{i-1} \pmod{m_i}$.

□

## 2. Fourierova transformácia

Definícia 3. Nech $K$ je okruh. Potom prvok $\omega \in K$ sa nazýva *$n$-tá primitívna odmocnina z jednej* práve vtedy, keď $\omega^n = 1$ a $\omega^j \neq 1$ pre $0 < j < n$.

Veta 4. *V $\mathbb{Z}_p$ existuje primitívna $n$-tá odmocnina z jednej práve vtedy, keď $n|(p-1)$.*

Dôkaz. Podľa Lagrangeovej vety rád prvku grupy delí rád grupy. Keďže multiplikatívna grupa $\mathbb{Z}_p^*$ má rád $p-1$, $n$ musí deliť $p-1$.

Naopak predpokladajme, že $n|(p-1)$. Multiplikatívna grupa $\mathbb{Z}_p^*$ je cyklická, teda má generátor, ktorý označím $\alpha$. Prvok $\beta := \alpha^{\frac{p-1}{n}}$ má rád $n$ v $\mathbb{Z}_p^*$, teda $\beta$ je primitívna $n$-tá odmocnina z jednej.

□

Veta 5. *Nech $n$ a $\omega$ sú mocniny dvojky s kladnými exponentmi a nech $m = \omega^{n/2} + 1$. Potom v okruhu $\mathbb{Z}_m$ existuje inverzný prvok k $n$ vzhľadom k násobeniu a $\omega$ je primitívna $n$-tá odmocnina z jednej v $\mathbb{Z}_m$.*

Dôkaz. Čísla $m$ a $n$ sú nesúdeliteľné, teda $n$ je invertibilný v $\mathbb{Z}_m$. Keďže $\omega^n = \omega^{n/2}\omega^{n/2} \equiv (-1)(-1) \equiv 1 \pmod{m}$, $\omega$ je $n$-tá odmocnina z jednej v $\mathbb{Z}_m$. Ešte potrebujem dokázať, že $\omega^j \neq 1$ pre $0 < j < n$. Pre $0 < j < n/2$ máme $1 < \omega^j < m-1$, teda $\omega^j \not\equiv \pm 1 \pmod{m}$. Pre $j = n/2$ máme $\omega^j \equiv -1 \pmod{m}$. Pre $n/2 < j < n$ máme $\omega^j = \omega^{n/2}\omega^{j-n/2} \equiv -\omega^{j-n/2} \not\equiv \pm 1 \pmod{m}$.

□

Definícia 6. Nech $\omega$ je $n$-tá primitívna odmocnina z jednej v okruhu $K$. Nech $A = (A_{ij})_{0 \leq i,j < n}$ je matica, kde $A_{ij} = \omega^{ij}$. Nech $a = (a_0, a_1, \dots, a_{n-1})^T$ je vektor dĺžky $n$ nad $K$. Vektor $F_\omega(a) = A \cdot a$, ktorého $i$-tá zložka je $\sum_{k=0}^{n-1} a_k \omega^{ik}$, $0 \leq i < n$, sa nazýva *diskrétna Fourierova transformácia* vektoru $a$ (skrátene *DFT*) a vektor $F_\omega^{-1}(a) = A^{-1} \cdot a$ sa nazýva *inverzná diskrétna Fourierova transformácia* vektoru $a$ (skrátene *IDFT*).

Po rozpísaní dostaneme:

$$\begin{aligned} F_\omega(a) = A \cdot a &= \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} a_0 + a_1 + a_2 + \dots + a_{n-1} \\ a_0 + a_1\omega + a_2\omega^2 + \dots + a_{n-1}\omega^{n-1} \\ \vdots \\ a_0 + a_1\omega^{n-1} + a_2\omega^{2(n-1)} + \dots + a_{n-1}\omega^{(n-1)^2} \end{pmatrix} \end{aligned} \tag{1}$$

Tvrdenie 7. *Ak je $\omega$ primitívna $n$-tá odmocnina z jednej v okruhu $K$, prvok $\underbrace{1+1+\dots+1}_{n}$, kde $n \in \mathbb{N}$, je invertibilný v $K$ a matica $A$ je tvaru $(\omega^{ij})_{i,j=0}^{n-1}$, potom $A^{-1} = \frac{1}{n}(\omega^{-ij})_{i,j=0}^{n-1}$.*

*Teda* $F_\omega^{-1}(a) = A^{-1} \cdot a = \frac{1}{n} F_{\omega^{-1}}(a)$.

Dôkaz. Stačí dokázať, že $AA^{-1} = E$, kde $E$ je jednotková matica. Platí, že

$$AA^{-1} = (\omega^{ij})_{i,j=0}^{n-1} \cdot \frac{1}{n}(\omega^{-ij})_{i,j=0}^{n-1} = \frac{1}{n}\left(\sum_{k=0}^{n-1} \omega^{ik}\omega^{-kj}\right)_{i,j=0}^{n-1}.$$

Ak $i = j$, potom $\frac{1}{n}\sum_{k=0}^{n-1} \omega^{ik}\omega^{-kj} = \frac{1}{n}\sum_{k=0}^{n-1} \omega^{ik}\omega^{-ki} = \frac{1}{n}\sum_{k=0}^{n-1} 1 = \frac{1}{n}n = 1$. Ak $i \neq j$, potom $\sum_{k=0}^{n-1} \omega^{ik}\omega^{-kj} = \sum_{k=0}^{n-1} \omega^{k(i-j)} = \sum_{k=0}^{n-1}(\omega^{i-j})^k = \frac{(\omega^{i-j})^n - 1}{\omega^{i-j} - 1}$. Vieme, že $\omega^{i-j} \neq 1$, pretože $i - j \in \{\pm 1, \pm 2, \ldots, \pm(n-1)\}$ a $\omega$ je primitívna $n$-tá odmocnina z jednej a preto menovateľ zlomku má zmysel. Ďalej $(\omega^{i-j})^n = (\omega^n)^{i-j} = 1^{i-j} = 1$. Platí tedy $\frac{(\omega^{i-j})^n - 1}{\omega^{i-j} - 1} = \frac{1-1}{\omega^{i-j}-1} = 0$.

□

Ak budeme mať algoritmus na výpočet *DFT*, predchádzajúce tvrdenie hovorí, že budeme vedieť pomocou neho zistiť i *IDFT*. Teraz sa budeme snažiť takýto algoritmus popísať. Ztotožníme vektor $a = (a_0, a_1, \ldots, a_{n-1})$ s polynómom $a(x)$ tak, že $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1}$. Z (1) vidíme, že

$$F_\omega(a) = \begin{pmatrix} a(1) \\ a(\omega) \\ \vdots \\ a(\omega^{n-1}) \end{pmatrix}.$$

Predpokladajme, že $n$ je párne číslo a označíme si $m = n/2$. Pretože platí, že

$$\begin{aligned} a(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1}x^{n-1} \\ &= \underbrace{(a_0 + a_2 x^2 + a_4 x^4 + \ldots + a_{n-2}x^{n-2})}_{q(x^2)} + \underbrace{(a_1 x + a_3 x^3 + \ldots + a_{n-1}x^{n-1})}_{xr(x^2)} \\ &= q(x^2) + xr(x^2), \end{aligned}$$

kde $q(x) = \sum_{i=0}^{m-1} a_{2i}x^i$ a $r(x) = \sum_{i=0}^{m-1} a_{2i+1}x^i$. Teda

$$F_\omega(a) = \begin{pmatrix} a(1) \\ a(\omega) \\ \vdots \\ a(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} q(1) + r(1) \\ q(\omega^2) + \omega r(\omega^2) \\ q(\omega^4) + \omega^2 r(\omega^4) \\ \vdots \\ q(\omega^{2(n-1)}) + \omega^{n-1} r(\omega^{2(n-1)}) \end{pmatrix}.$$

Predpokladajme, že $n = 2^k$ pre nejaké $k \in \mathbb{N}$ a $\omega$ je $n$-tá primitívna odmocnina z jedné v $K$. Potom algoritmus, ktorý rekurzívne využíva predchádzajúce zistenie na výpočet *DFT* sa dá zapísať nasledujúcim spôsobom:

ALGORITMUS (*Rýchla Fourierova transformácia-FFT*($\omega$)).
VSTUP: $(a_0, \ldots, a_{n-1})$
VÝSTUP: $F_\omega((a_0, \ldots, a_{n-1}))$
1. Ak $n = 1$, odpovedz $a_0$, koniec
2. $m := n/2$;
   $(b_0, \ldots, b_{m-1}) := FFT(\omega^2)(a_0, a_2, \ldots, a_{n-2})$
   $(c_0, \ldots, c_{m-1}) := FFT(\omega^2)(a_1, a_3, \ldots, a_{n-1})$
3. pre $i$ od 0 do $m-1$ dosaď
   $d_i := b_i + \omega^i c_i$
   $d_{m+i} := b_i - \omega^i c_i$
4. odpovedz $(d_0, \ldots, d_{n-1})$

TVRDENIE 8. *Ak je $\omega$ primitívna $n$-tá odmocnina z jednej v $K$, potom algoritmus funguje.*

DÔKAZ. Najprv musíme dokázať, že $\omega^2$ je primitívna $m$-tá odmocnina z jednej. Platí

$$(\omega^2)^m = (\omega^2)^{\frac{n}{2}} = \omega^{2\frac{n}{2}} = \omega^n = 1.$$

Pre všetky $i = 1, 2, \ldots, m-1$ platí $(\omega^2)^i = \omega^{2i} \neq 1$, pretože $2i \in \{1, \ldots, n-1\}$ a $\omega$ je primitívna $n$-tá odmocnina z jednej.

Ďalej dokážeme, že algoritmus počíta to, čo chceme. Dôkaz budeme robiť indukciou podľa $n$.

Pre $n = 1$ algoritmus funguje. Z indukčného predpokladu platia rovnosti

$$(b_0, \ldots, b_{m-1}) = (q(1), q(\omega^2), q(\omega^4), \ldots, q(\omega^{2n-2}))$$
$$(c_0, \ldots, c_{m-1}) = (r(1), r(\omega^2), r(\omega^4), \ldots, r(\omega^{2n-2})).$$

Pre $i = 0, 1, \ldots, m-1$ chceme dokázať, že $d_i = a(\omega^i)$ a $d_{m+i} = a(\omega^{m+i})$. Z algoritmu a indukčného predpokladu dostávame, že

$$d_i = b_i + \omega^i c_i = q(\omega^{2i}) + \omega^i r(\omega^{2i}) = a(\omega^i)$$

Pretože $(\omega^m)^2 = 1$ a $\omega^m \neq 1$ ($\omega$ je primitívna $n$-tá odmocnina z jednej), musí byť rovná $-1$, pretože sú len dve druhé odmocniny z 1 (korene polynómu $x^2 - 1$). Teda platí

$$d_{m+i} = b_i - \omega^i c_i = q(\omega^{2i}) - \omega^i r(\omega^{2i}) = q(\omega^{2i}) + \omega^{m+i} r(\omega^{2i}).$$

Ďalej platí, že

$$\omega^{2(m+i)} = \omega^{2i}\omega^{2m} = \omega^{2i}\omega^n = \omega^{2i}$$

a teda

$$d_{m+i} = q(\omega^{2i}) + \omega^{m+i} r(\omega^{2i}) = q(\omega^{2(m+i)}) + \omega^{m+i} r(\omega^{2(m+i)}) = a(\omega^{m+i}).$$

□

VETA 9. *Nech $\omega$ a $n$ sú mocniny 2 a $m = \omega^{n/2} + 1$. Nech $a = (a_0, \ldots, a_{n-1})$ je vektor nad okruhom $\mathbb{Z}_m$. Potom DFT a IDFT vektoru $a$ v okruhu $\mathbb{Z}_m$ majú časovú zložistosť $n^2 \log n \log \omega$.*

DÔKAZ. Nech $T(n, \omega)$ je časová zložitosť $FFT(\omega)$ v okruhu $\mathbb{Z}_m$ na vstupe dĺžky $n$. Všetky operácie budú prebiehať v okruhu $\mathbb{Z}_m$. Zložitosť $T(n, \omega)$ je rovná dvakrát $T(n/2, \omega^2)$ (z kroku 2) plus $2m$-krát časová zložitosť násobenia $\omega^j$ a následného sčítania a odčítania na výpočet $d_i$ a $d_{m+i}$ z kroku 3.

Pretože $\omega$ je mocnina dvojky, je násobenie $\omega^j$ posunutím doľava a výsledok je určite menší ako $\omega^n$. Teda môžeme napísať:

$$\omega^j c_j = z_0 + z_1\omega^{n/2} \equiv z_0 - z_1 \pmod{m},$$

kde $0 \leq z_i < \omega^{n/2}$. Dostávame, že časová zložitosť na výpočet $d_i$ je $O(\log m) = O(n \log \omega)$. Potom

$$T(n, \omega) = 2T(n/2, \omega^2) + O(n^2 \log \omega).$$

Rozpísaním dostaneme:

$$\begin{aligned} T(n,\omega) &\leq 2T(n/2,\omega^2) + c(n^2 \log \omega) \\ &\leq 2(2T(n/4,\omega^4) + c(n^2/4 \log \omega^2)) + c(n^2 \log \omega) \\ &= 4T(n/4,\omega^4) + 2c(n^2 \log \omega) \leq \cdots \leq nT(1,\omega^n) + c \log n(n^2 \log \omega), \end{aligned}$$

kde $c$ je nejaká konštanta. Pretože $T(1, \omega^n) = T(1, 1)$ má konštantnú časovú zložitosť, platí $T(n, \omega) = O(n^2 \log n \log \omega)$.

Dokázali sme si, že počítať IDFT znamená počítať DFT s parametrom $\omega^{-1}$ a výsledok vydeliť $n$. Násobenie $\omega^{-j}$ je to isté ako násobenie $\omega^{n-j}$. To znamená, že v kroku 3 je násobenie zase len posunutím doľava. Pretože $n/2 < n - j < n$, výsledok je menší než $\omega^{(3/2)n}$ a môžeme napísať, že

$$\omega^{-j} c_j = z_0 + z_1\omega^{n/2} + z_2\omega^n \equiv z_0 - z_1 + z_2 \pmod{m}.$$

Naviac násobíme $c_j$ aspoň $\omega^{n/2}$ a preto $z_0 = 0$. Teda časová zložitosť na výpočet $d_i$ je $O(\log m) = O(n \log \omega)$ ako pre DFT.

Ešte musíme výsledok vynásobiť $1/n$. Ak $n = 2^k$, potom

$$2^k 2^{n \log \omega - k} \equiv 2^{n \log \omega} \equiv \omega^n \equiv 1 \pmod{m}.$$

Takže násobenie $1/n$ môžeme počítať ako posunutie doľava o $n \log \omega - k$ pozícií a pretože $\omega^{n/2} \leq 2^{n \log \omega - k} < \omega^n$, môžeme postupovať ako vyššie. Teda násobenie $1/n$ modulo $m$ môžeme vypočítať v čase $O(n \log \omega)$. Podobným rozpísaním ako u DFT dostaneme, že IDFT má časovú zložitosť $O(n^2 \log n \log \omega)$.

□

Veta 10. *Nech $\omega$ je $n$-tá primitívna odmocnina z jednej v okruhu $K$, $\psi^2 = \omega$. Nech $a = (a_0, \ldots, a_{n-1})$ a $b = (b_0, \ldots, b_{n-1})$ sú vektory dĺžky $n$ nad okruhom $K$ a nech $c = (c_0, \ldots, c_{n-1})$ je vektor, ktorý spĺňuje*

$$c_p = \sum_{j=0}^{p} a_j b_{p-j} - \sum_{j=p+1}^{n-1} a_j b_{n+p-j} \quad \textit{pre} \quad p = 0, \ldots, n-1.$$

*Definujme vektory $\tilde{a} := (a_0, \psi a_1, \ldots, \psi^{n-1} a_{n-1})$, $\tilde{b} := (b_0, \psi b_1, \ldots, \psi^{n-1} b_{n-1})$ a $\tilde{c} := (c_0, \psi c_1, \ldots, \psi^{n-1} c_{n-1})$, potom*

$$\tilde{c} = F_\omega^{-1}(F_\omega(\tilde{a}) \cdot F_\omega(\tilde{b})).$$

Dôkaz. Nech $F_\omega(\tilde{c}) = (c'_0, \dots, c'_{n-1})$, teda

$$c'_l = \sum_{p=0}^{n-1} \psi^p c_p \omega^{lp} = \sum_{p=0}^{n-1} \sum_{j=0}^{p} \psi^p a_j b_{p-j} \omega^{lp} - \sum_{p=0}^{n-1} \sum_{j=p+1}^{n-1} \psi^p a_j b_{n+p-j} \omega^{lp}$$
$$= \sum_{p=0}^{n-1} \sum_{j=0}^{p} \psi^p a_j b_{p-j} \omega^{lp} + \sum_{p=0}^{n-1} \sum_{j=p+1}^{n-1} \psi^{n+p} a_j b_{n+p-j} \omega^{lp}. \qquad (2)$$

Na druhej strane, nech

$$F_\omega(\tilde{a}) = (a'_0, \dots, a'_{n-1}), \qquad a'_l = \sum_{p=0}^{n-1} \psi^p a_p \omega^{lp},$$

$$F_\omega(\tilde{b}) = (b'_0, \dots, b'_{n-1}), \qquad b'_l = \sum_{p=0}^{n-1} \psi^p b_p \omega^{lp}.$$

$$a'_l b'_l = \sum_{q=0}^{n-1} \sum_{r=0}^{n-1} \psi^{q+r} a_r b_q \omega^{l(q+r)}$$
$$= \sum_{s=0}^{n-1} \sum_{t=0}^{s} \psi^s a_t b_{s-t} \omega^{ls} + \sum_{s=0}^{n-1} \sum_{t=s+1}^{n-1} \psi^{n+s} a_t b_{n+s-t} \omega^{ls} = (2).$$

Druhú rovnosť dostaneme, ak si uvedomíme, že $\sum_{q=0}^{n-1} \sum_{r=0}^{n-1} \psi^{q+r} a_r b_q \omega^{l(q+r)}$ sa dá napísať ako $\sum_A \psi^{q+r} a_r b_q \omega^{l(q+r)}$, kde $A = \{[q,r], 0 \le q, r \le n-1\}$. Pretože $A_H \cup A_D$, je disjunktný rozklad množiny $A$, kde

$$A_D = \{[q,r], r+q \le n-1, 0 \le q, r \le n-1\}$$

a

$$A_H = \{[q,r], r+q > n-1, 0 \le q, r \le n-1\},$$

platí, že

$$\sum_A \psi^{q+r} a_r b_q \omega^{l(q+r)} = \sum_{A_D} \psi^{q+r} a_r b_q \omega^{l(q+r)} + \sum_{A_H} \psi^{q+r} a_r b_q \omega^{l(q+r)},$$

takže dostávame požadovanú rovnosť.

□

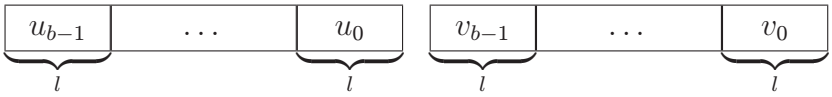
KAPITOLA 3

# Schönhageov-Strassenov algoritmus

Nech $u$ a $v$ sú binárne čísla dĺžky $n$, kde $n = 2^k$ pre nejaké $k \in \mathbb{N}$. Schönhageov-Strassenov algoritmus je algoritmus na násobenie modulo $2^n + 1$. Ak chceme nemodulárny výsledok, pridáme pred čísla $n$ núl. Dostaneme tak dve čísla dĺžky $2n$, ktoré vynásobíme modulo $2^{2n} + 1$. Výsledok tohto násobenia je nami požadovaný súčin, pretože súčin dvoch binárnych čísel dĺžky $n$ je menší ako $2^{2n}$.

Nech $u, v$ sú dve binárne čísla také, že $0 \leq u, v \leq 2^n$, ktorých súčin chceme vypočítať modulo $2^n + 1$. Ak je jedno z $u$, $v$ rovno $2^n$, riešime to ako špeciálny prípad tj. nech $v = 2^n$, potom $uv = u2^n \equiv -u \equiv 2^n + 1 - u \pmod{2^n + 1}$.

Inak rozdelíme $u$ a $v$ na $b$ blokov po $l$ bitoch tak, že $b = 2^{k/2}$, ak $k$ je párne a $b = 2^{(k-1)/2}$, ak $k$ je nepárne. Naviac platí, že $l = n/b$ a ďalej $l/b = 1$ alebo $l/b = 2$, teda $b \mid l$.




Teraz čísla $u$, $v$ môžeme zapísať následovne:

$$u = u_{b-1}2^{(b-1)l} + \cdots + u_1 2^l + u_0,$$
$$v = v_{b-1}2^{(b-1)l} + \cdots + v_1 2^l + v_0.$$

Potom platí, že

$$uv = y_{2b-1}2^{(2b-1)l} + \cdots + y_1 2^l + y_0, \tag{3}$$

kde

$$y_i = \sum_{j=0}^{b-1} u_j v_{i-j}, \qquad 0 \leq i < 2b.$$

(Pre $j > b-1$ alebo $j < 0$ položíme $u_j = v_j = 0$.) Aby sme mohli vypočítať súčin $u$ a $v$, musíme zistiť hodnoty $y_i$ pre $i = 0, \ldots, 2b-1$.

Z rovnosti $2^n + 1 = 2^{bl} + 1$ dostávame

$$y_i 2^{il} + y_{b+i}2^{(b+i)l} \equiv (y_i - y_{b+i})2^{il} \pmod{2^n + 1}.$$

Potom platí

$$\begin{aligned} uv &= y_{2b-1}2^{(2b-1)l} + \cdots + y_1 2^l + y_0 \\ &\equiv (y_{b-1} - y_{2b-1})2^{(b-1)l} + \cdots + (y_1 - y_{b+1})2^l + (y_0 - y_b) \\ &= w_{b-1}2^{(b-1)l} + \cdots + w_1 2^l + w_0 \pmod{2^n + 1}, \end{aligned}$$

kde

$$w_i = (y_i - y_{b+i}) \text{ pre } 0 \leq i < b.$$

Takže namiesto $2b-1$ hodnôt $y_i$ budeme počítať $b$ hodnôt $w_i$ pre $i = 0, \ldots, b-1$. Výsledok dostaneme, ak najprv spočítame súčet $w_i 2^{il}$, kde $w_i > 0$, potom súčet

$w_i 2^{il}$ pre $w_i < 0$ a nakoniec od prvej sumy odčítame druhú. Dôvodom tohto postupu namiesto obyčajného sčítania je zachovanie zložitosti algoritmu.

Pretože násobok dvoch binárnych čísel dĺžky $l$ je menší než $2^{2l}$ a $y_i$ je súčet $i+1$ súčinov dvoch binárnych čísel dĺžky $l$ a $y_{b+i}$ je súčet $b-(i+1)$ súčinov dvoch binárnych čísel dĺžky $l$, máme odhad

$$(i+1-b)2^{2l} < w_i < (i+1)2^{2l},$$

pre $i = 0, \ldots, b-1$. Teda $w_i$ môže nabývať najviac $b2^{2l}$ rôznych hodnôt a môžeme teda počítať $w_i$ modulo $b2^{2l}$ a to následovne:

Budeme najprv počítať modulo $b$ a modulo $2^{2l}+1$. Označíme

$$w'_i = w_i \bmod b, \quad w''_i = w_i \bmod (2^{2l}+1).$$

Keďže $b$ je mocnina dvojky a $2^{2l}+1$ je nepárne číslo, sú nesúdeliteľné. Môžeme teda aplikovať Čínsku vetu o zvyškoch a z Garnerovho algoritmu dostávame, že

$$z_i := w''_i + (2^{2l}+1)((w'_i - w''_i) \bmod b) \qquad \text{pro } 0 \le i < b,$$

a platí

$$z_i \equiv w_i \pmod{b(2^{2l}+1)} \quad \text{pro} \quad 0 \le i < b. \tag{4}$$

Navyše $0 \le z_i < (2^{2l}+1)b$. Pretože $b2^{2l} < b(2^{2l}+1)$ a musí platiť (3) a súčasne (4), potom

$$\begin{aligned} w_i &= z_i && \text{pre} \quad z_i < (i+1)2^{2l}, \\ w_i &= z_i - b(2^{2l}+1) && \text{pre} \quad z_i \ge (i+1)2^{2l}. \end{aligned} \tag{5}$$

Teda $w_i$ vieme vypočítať, ak poznáme $w'_i$ a $w''_i$ pre $i = 0, 1, \ldots, b-1$.

Teraz budeme počítať $w'_i$. Označíme

$$\begin{aligned} u'_i &= u_i \bmod b, \\ v'_i &= v_i \bmod b \end{aligned}$$

a zkonštruujeme čísla $\hat{u}$, $\hat{v}$ tak, že medzi každé $u'_{i+1}$ a $u'_i$ a každé $v'_{i+1}$ a $v'_i$ pre $i = 0, 1, \ldots, b-2$ vsunieme $2\log b$ núl.

$$\begin{aligned} \hat{u} &= u'_{b-1}00\ldots0u'_{b-2}0\ldots\ldots0u'_0, \\ \hat{v} &= v'_{b-1}\underbrace{00\ldots0}_{2\log b}v'_{b-2}0\ldots\ldots0v'_0 \end{aligned} \tag{6}$$

Tieto čísla majú dĺžku $3b\log b$. Počítaním súčinu $\hat{u}\hat{v}$ pomocou Karacubovho algoritmu dostaneme

$$\hat{u}\hat{v} = \sum_{i=0}^{2b-1} y'_i 2^{(3\log b)i}, \text{ kde} \qquad y'_i = \sum_{j=0}^{b-1} u'_j v'_{i-j}.$$

Pretože $y'_i < 2^{3\log b}$ ľahko ho dostaneme z $\hat{u}\hat{v}$ (ako $i$-tý blok o $3\log b$ bitoch). Potom

$$w'_i = w_i \bmod b = y'_i - y'_{b+i} \bmod b.$$

Ešte nám zostáva vypočítať $w''_i$. Keďže $w''_i = w_i \bmod (2^{2l}+1)$, budeme pracovať v okruhu $\mathbb{Z}_m$, kde $m = 2^{2l}+1$. Nech $\omega = 2^{4l/b}$. Z vety 5 má $\underbrace{1+\cdots+1}_{b}$

inverzný prvok v tomto okruhu a $\omega$ je $b$-tá primitívna odmocnina z jednej. Nech $\psi^2 = \omega$ a teda $\psi = 2^{2l/b}$. Nech $c = (c_0, \ldots, c_{b-1})$ je vektor, ktorý spĺňuje

$$c_i = \sum_{j=0}^{i} u_j v_{i-j} - \sum_{j=i+1}^{b-1} u_j v_{b+i-j} = y_i - y_{b+i} = w_i. \tag{7}$$

Definujme $\tilde{u}, \tilde{v}$ a $\tilde{c}$ následovne:

$$\begin{aligned} \tilde{u} &= (u_0, \psi u_1, \ldots, \psi^{b-1} u_{b-1}), \\ \tilde{v} &= (v_0, \psi v_1, \ldots, \psi^{b-1} v_{b-1}), \\ \tilde{c} &= (c_0, \psi c_1, \ldots, \psi^{b-1} c_{b-1}), \end{aligned} \tag{8}$$

kde zložky vektorov sú modulo $m$. Podľa vety 10 platí, že $\tilde{c} = F_\omega^{-1}(F_\omega(\tilde{u}) \cdot F_\omega(\tilde{v}))$. Teda podľa (7)

$$w_i'' = w_i \bmod m = c_i \bmod m = \psi^{-i} \tilde{c}_i \bmod m.$$

Algoritmus môžeme zapísať následovne:

Algoritmus (*NÁSOBSS*).

`VSTUP:` $u$,$v$ - 2 binárne čísla dĺžky $n$, kde $n = 2^k$ pre nejaké $k \in \mathbb{N}$
`VÝSTUP:` $w$ také, že $w = uv \bmod 2^n + 1$

`1.` pre $n$ menšej dĺžky ako nejaká konštanta použi nejaký jednoduchý algoritmus na násobenie; koniec
`2.` definuj $b$, $l$, $m$ a reprezentuj $u$ a $v$ v bázi o základe $2^l$:
- ak $k$ je párne $b := 2^{k/2}$ inak $b := 2^{(k-1)/2}$
- $l := n/b$
- $m := 2^{2l} + 1$
- $u = \sum_{i=0}^{b-1} u_i 2^{li}$, $v = \sum_{i=0}^{b-1} v_i 2^{li}$, kde $0 \le u_i, v_i \le 2^l - 1$

`3.` výpočet $w_i''$ pre $i = 0, 1, \ldots, b-1$:
- definuj $\tilde{u}$, $\tilde{v}$ podľa (8)

`a.` $p := FFT(\omega)(\tilde{u})$ v $\mathbb{Z}_m$
$q := FFT(\omega)(\tilde{v})$ v $\mathbb{Z}_m$
`b.` $r := p \cdot q$, kde $\cdot$ je násobenie po zložkách pomocou *NÁSOBSS* (v $\mathbb{Z}_m$)
`c.` $\tilde{c} := \frac{1}{b} FFT(\omega^{-1})(r)$ v $\mathbb{Z}_m$
`d.` $w_i'' = \psi^{-i} \tilde{c}_i \bmod m$ pre $i = 0, 1, \ldots, b-1$

`4.` výpočet $w_i'$ pre $i = 0, 1, \ldots, b-1$:
`a.` konštrukcia $\hat{u}$, $\hat{v}$ ako v (6)
`b.` $\hat{u}\hat{v}$ pomocou Karacubovho algoritmu
`c.` $w_i' = w_i \bmod b = y_i' - y_{b+i}' \bmod b$,
kde $y_i$ je $i$-tý blok $\hat{u}\hat{v}$ o $3 \log b$ bitoch, pre $i = 0, 1, \ldots, b-1$

`5.` $w_i$ podľa (5) pre $i = 0, 1, \ldots, b-1$
`6.` $w^+ := \sum_{w_i > 0} w_i 2^{il}$
$w^- := \sum_{w_i < 0} -w_i 2^{il}$
$w := (w^+ - w^-) \bmod 2^n + 1$

Veta 11. *Algoritmus NÁSOBSS má časovú zložitosť* $O(n \log n \log\log n)$.

Dôkaz. Označíme si $T(n)$ časovú zložitosť algoritmu *NÁSOBSS* pre vstupné binárne čísla dĺžky $n$.

Kroky `1` a `2` majú zložitosť maximálne lineárnu.

Krok `3.a` a krok `3.c`: Z vety 9 vieme, že časová zložitosť $FFT$ v $\mathbb{Z}_m$ je $O(b^2 \log b \log \omega) = O(b^2 \log b \log 2^{4l/b}) = O(b^2 \frac{4l}{b} \log b) = O(bl \log b)$.

Krok `3.b` má časovú zložitosť $bT(\log m) = bT(2l)$ ($b$ je počet zložiek vektoru, teda násobím $b$-krát číslo dĺžky najviac $\log m$).

Krok `3.d` má časovú zložitosť menšiu alebo rovnakú ako krok `3.c`, pretože násobenie inverzným prvkom k mocnine dvojky a počítanie modulo $m$ sa vykonáva aj v kroku `3.c`.

Takže celková asymptotická časová zložitosť kroku `3` je $bT(2l) + O(bl \log b)$.

Krok `4` trvá $O((3b \log b)^{\log 3}) < O(b^2) < O(b^2 \log b \log \omega)$. Teda jeho časová zložitosť je menšia než časová zložitosť kroku `3`.

Jednoducho sa dá dokázať, že krok `5` a krok `6` majú lineárnu zložitosť. Dostali sme, že

$$T(n) = bT(2l) + O(bl \log b) = bT(2l) + O(n \log b).$$

Takže platí

$$T(n) \leq bT(2l) + cn \log b,$$

kde $c$ je nejaká konštanta. Označím $T'(n) := \frac{T(n)}{n}$. Potom

$$T'(n) \leq \frac{b}{n} T(2l) + c \log b = \frac{2}{2l} T(2l) + c \log b = 2T'(2l) + c \log b.$$

Pretože $l = n/b$, je $l = 2^{\frac{k}{2}} = \sqrt{n}$ alebo $l = 2^{\frac{k+1}{2}} = 2\sqrt{n}$. Teda $l \leq 2\sqrt{n}$ a $b < n$, potom

$$T'(n) \leq 2T'(4\sqrt{n}) + c \log n.$$

Teraz budem indukciou podľa $n$, dokazovať, že $T'(n) \leq c' \log n \log \log n$, pre nejakú konštantu $c'$ a $n$ dostatočne veľké. Vhodnou voľbou $c'$ to platí pre všetky $n$, $n_0 \leq n \leq \frac{n_0^2}{4}$, kde $n_0$ zvolíme neskôr. Nech $n > \frac{n_0^2}{4}$ a predpokladajme, že platí $T'(m) \leq c' \log m \log \log m$ pre $n_0 \leq m < n$. Pretože $n_0 \leq 4\sqrt{n} < n$ a $2 + \frac{1}{2} \log n \leq \frac{2}{3} \log n$ pre $n$ dosť veľké, platí:

$$\begin{aligned} T'(n) &\leq 2T'(4\sqrt{n}) + c \log n \leq 2c' \log(4\sqrt{n}) \log \log(4\sqrt{n}) + c \log n \\ &= (4c' + c' \log n) \log(2 + \frac{1}{2} \log n) + c \log n \\ &\leq (4c' + c' \log n) \log(\frac{2}{3} \log n) + c \log n \\ &= 4c' \log \frac{2}{3} + 4c' \log \log n + c' \log n \log \frac{2}{3} + c' \log n \log \log n + c \log n \end{aligned} \tag{9}$$

Vieme, že $\log \frac{2}{3} < -\frac{1}{2}$ a zvolíme $c'$ tak, aby $4c \leq c'$. Potom:

$$\begin{aligned} c \log n + 4c' \log \log n + c' \log n \log \frac{2}{3} &< \frac{c'}{4} \log n + 4c' \log \log n - \frac{c'}{2} \log n \\ &= -\frac{c'}{4} \log n + 4c' \log \log n = \frac{c'}{4}(16 \log \log n - \log n) < 0, \end{aligned} \tag{10}$$

pre veľké $n$. Aby všetky vyššie uvedené nerovnosti platili, stačí zvoliť napríklad $n_0 = 2^{12}$.

Keďže $4c' \log \frac{2}{3} < 0$ a platí (9) a (10) dostávame, že $T'(n) \leq c' \log n \log \log n$.

□

KAPITOLA 4

# Realizácia algoritmov

Uvedené algoritmy som naprogramovala v programovacom jazyku Object Pascal (Delphi).

## 1. Reprezentácia veľkých čísel

Na reprezentáciu veľkých celých čísel som použila datovú štruktúru *record* skladajúcu sa zo znamienka, dĺžky čísla a ukazateľ na miesto v pamäti, kde je dané číslo uložené:

```
type
 TDigits=array [0..0] of longword;
 PDigits=^TDigits;
 TBigInt=record
   sgn,len:integer;
   digit:PDigits;
 end;
```

Vzhľadom k tomu, že pracujem s 32-bitovým procesorom, čísla sú uložené v sústave o základe $2^{32}$.

## 2. Primitívny algoritmus

Procedúra, ktorá realizuje primitívny algoritmus je *multnaive_* . Na vylepšenie primitívneho algoritmu som využila knihu [**K**].

```
procedure multnaive_(a,b:TBigInt; var c:TBigInt);
\\násobí čísla a,b a výsledok ukladá do c
var i,j,z:integer;
    ta,tb,tc:longword;
begin
 if (a.sgn and b.sgn)=0 then begin c.sgn:=0; c.len:=0; exit; end;
 if a.sgn>0 then c.sgn:=b.sgn else c.sgn:=-b.sgn;
 \\pokiaľ je jedno z čísel nulové výsledok je tiež nula
 c.len:=a.len+b.len;
 fillchar(c.digit^,4*a.len,0);
 \\vynuluje miesto v pamäti
 for j:=0 to b.len-1 do begin
  tb:=b.digit[j];
  if tb=0 then c.digit[a.len+j]:=0 else
    \\ak je cifra nulová zbytočne nenásobí
  begin
   z:=0;
   for i:=0 to a.len-1 do
   begin
```

```
    ta:=a.digit[i]; tc:=c.digit[i+j];
    asm
      mov eax,ta        //eax:=ta
      mul eax,tb        //eax:=ta*tb mod 2^32, edx:=ta*tb div 2^32
      add eax,z         //eax:=eax+z
      adc edx,0         //prípadný prenos do vyššej cifry v edx
      add eax,tc        //eax:=eax+tc
      adc edx,0         //prípadný prenos do vyššej cifry v edx
      mov tc,eax        //ulož výsledok tc:=ta*tb+z+tc mod 2^32
      mov z,edx         //ulož zvyšok z:=ta*tb+z+tc div 2^32
    end;
    c.digit[i+j]:=tc;
   end;
   c.digit[j+a.len]:=z;
  end;
 end;
 if z=0 then dec(c.len);
end;
```

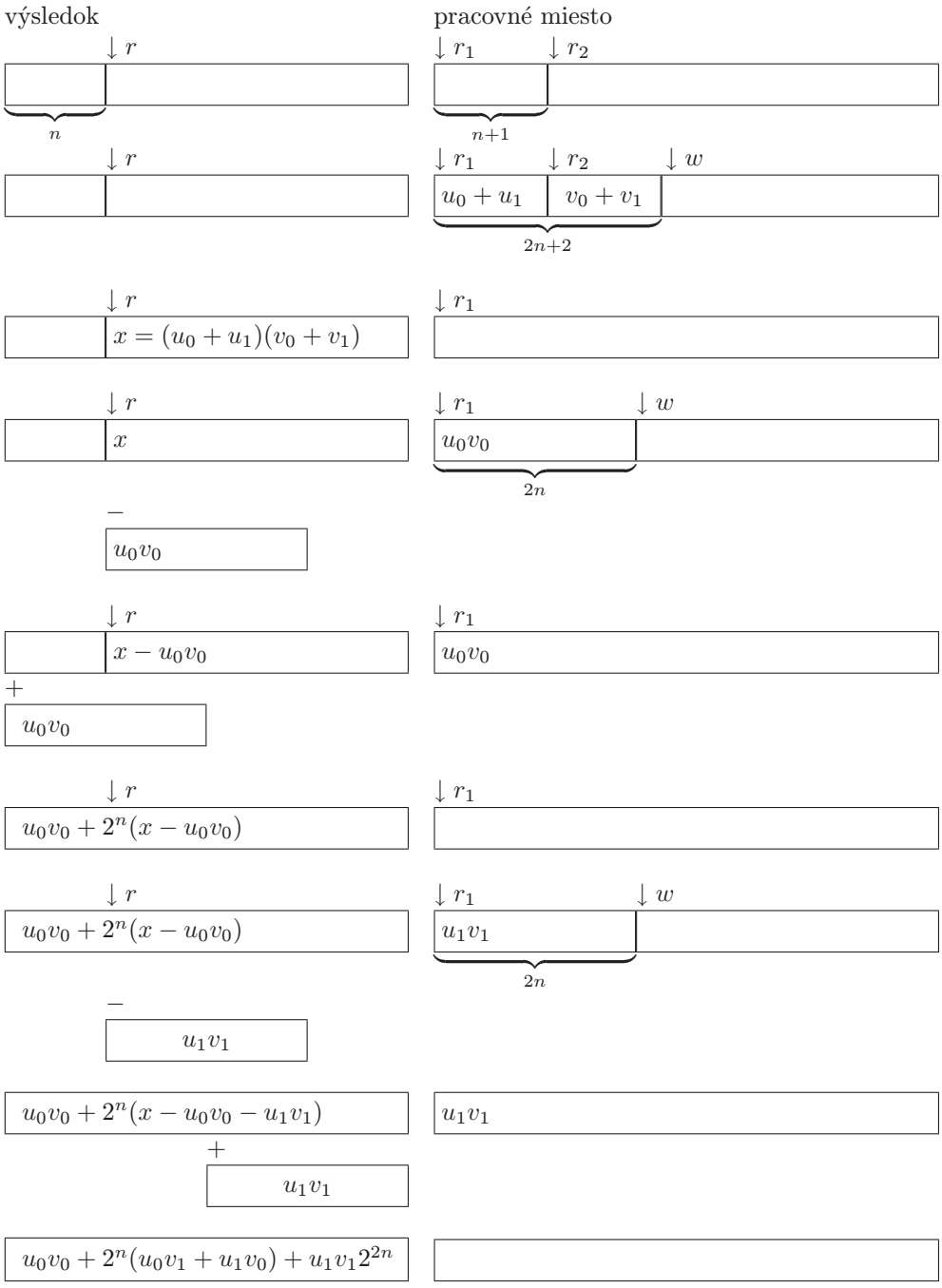
## 3. Algoritmus Karacuba

Na realizáciu Karacubovho algoritmu slúži procedúra *karatsuba_* . Na optimalizáciu algoritmu mi čiastočne pomohol článok [**M**]. Pomocou zlepšenia budem potrebovať len miesto na uloženie výsledku, ktoré má veľkosť $M + N$ a $2N + 180$ číslic pracovného miesta, kde $N$ je dĺžka väčšieho čísla a $M$ menšieho čísla. Veľkosť pracovného miesta uvedená v [**M**] je $2(N - 4 + 3\lfloor \log(N - 3) \rfloor)$. Toto číslo je pre jednoduchosť odhadnuté mnou použitou hodnotou, pretože $N \leq 2^{30}$ (ak uvážime maximálnu veľkosť pamäti pre 32-bitovú architektúru). Ďalej budeme používať $n = \lceil N/2 \rceil$.

```
procedure karatsuba_(a,b:TBigInt; work:PDigits; var c:TBigInt);
\\vynásobí čísla
var u0,v0,u1,v1,r1,r2,r:TBigInt;
    i,n:integer;
begin
  if b.len>a.len then begin u0:=a; a:=b; b:=u0; end;
  \\dlhšie číslo bude a
  if b.sgn=0 then begin c.sgn:=0; c.len:=0; exit; end;
  \\ak jedno je nula, potom výsledok je nulový
  if a.len<=KAR_THR then begin multnaive_(a,b,c); exit; end;
  \\KAR_THR je konštanta, od ktorej sa volá primitívny algoritmus
  n:=(a.len+1) shr 1;
  u0.sgn:=1; u0.digit:=a.digit;
  i:=n-1; while (i>=0) and (u0.digit[i]=0) do dec(i);
  u0.len:=i+1; if u0.len=0 then u0.sgn:=0;
  \\inicializácia u0, v ktorom bude uložené b z algoritmu
  u1.sgn:=1; u1.digit:=@a.digit[n]; u1.len:=a.len-n;
  \\inicializácia a z algoritmu
  if b.len>n then begin \\inicializácia c, d z algoritmu
```

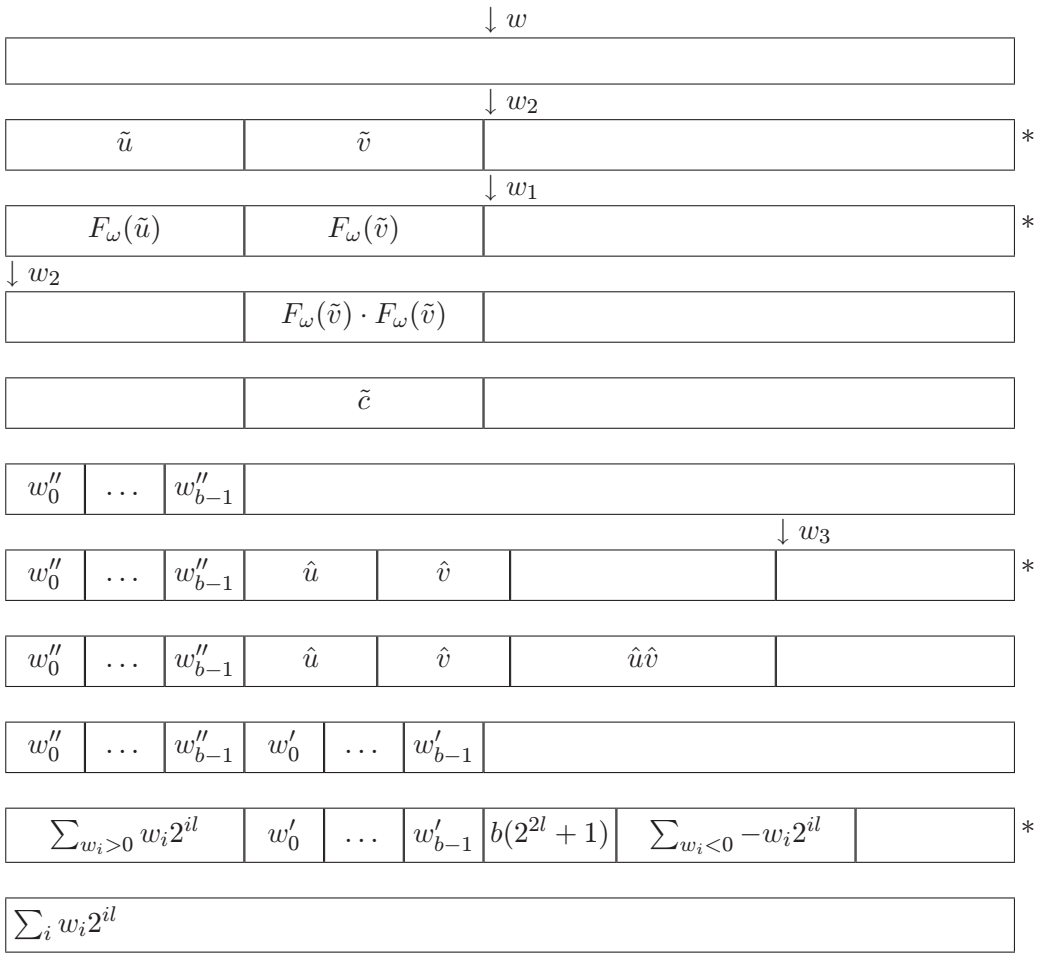
```
    v0.sgn:=1; v0.digit:=b.digit;
    i:=n-1; while (i>=0) and (v0.digit[i]=0) do dec(i);
    v0.len:=i+1; if v0.len=0 then v0.sgn:=0;
    v1.sgn:=1; v1.digit:=@b.digit[n]; v1.len:=b.len-n;
  end else begin
    v0.sgn:=1; v0.digit:=b.digit; v0.len:=b.len;
    v1.sgn:=0; v1.len:=0;
  end;
  r1.digit:=work; \\r1 ukazuje na začiatok pracovného miesta
  r2.digit:=@work[n+1];
  \\r2 ukazuje na n+2 miesto pracovného miesta
  i_sum(u0,u1,r1);
  \\súčet u1 a u0 ulož na začiatok pracovného miesta
  i_sum(v0,v1,r2);
  \\súčet v1 a v0 ulož do pracovného miesta
  \\so začiatkom na n+1-ej pozícii
  r.digit:=@c.digit[n]; \\r ukazuje na n+1-té miesto výsledku
  karatsuba_(r1,r2,@work[2*n+2],r);
  \\súčin (u0+u1) a (v0+v1) ulož do výsledku na miesto
  \\začínajúce na n-tú pozíciu
  karatsuba_(u0,v0,@work[2*n],r1);
  \\súčin u0 a v0 ulož na začiatok pracovného miesta
  i_sub(r1,r);
  \\od (u0+u1)(v0+v1) odpočítaj u0v0 a ulož do výsledku na miesto
  \\začínajúce n-tou pozíciu miesta
  \\výsledok r nemôže byť nula
  i_addspecr(n,r1,r);
  \\u0v0+2^{n/2}((u0+u1)(v0+v1)-u0v0) na začiatku výsledku
  karatsuba_(u1,v1,@work[2*n],r1);
  \\súčin u1v1 ulož na začiatok pracovného miesta
  i_sub(r1,r);
  \\od (u0+u1)(v0+v1)-u0v0 odpočítaj u1v1
  i_addspecl(n,r1,r);
  \\u1v1*2^n+((u0+u1)(v0+v1)-u0v0-u1v1)*2^{n/2}+u0v0
  \\výsledok r nemôže byt nula
  c.len:=r.len+n;
  c.sgn:=1;
end;
```

Ak môj postup nakreslím schématicky, kde $w$ bude označovať pracovné miesto pre rekurzívne volanú procedúru:




## 4. Schönhageov-Strassenov algoritmus

Procedúra, ktorá počíta násobok pomocou Schönhageovho-Strassenovho algoritmu je *multss_*. Na optimalizáciu algoritmu je dobré si uvedomiť, že $\psi$ a $\omega$ môžu byť rovné len dvom hodnotám a to $\psi = 2^2$ a $\omega = 2^4$, ak $k$ je párne, a $\psi = 2^4$ a $\omega = 2^8$, ak $k$ je nepárne. Ďalej som sa snažila zminimalizovať pracovné miesto a to ako som postupovala sa dá znázorniť takto:




Rozmery tabuľky neodpovedajú skutočným veľkostiam miest potrebných na uloženie. Ukazatele $w, w_1, w_2$ a $w_3$ označujú pracovné miesto pre *multss_*, na rekurzívne volanie *multss_*, FFT v $\mathbb{Z}_m$ a na násobenie $\hat{u}$ a $\hat{v}$ pomocou algoritmu Karacuba. Teda pracovné miesto potrebné na procedúru *multss_* bude maximum z minimálneho miesta potrebného na uloženie údajov v riadkoch označených *. Výpočtom dostaneme, že najviac miesta potrebujeme na uloženie druhého riadku. Toto miesto má veľkosť $\frac{5b}{2}(\frac{2l}{32} + 1)$ 32-bitových číslic.

KAPITOLA 5

# Porovnanie algoritmov

## 1. Optimalizácia parametrov

Po naprogramovaní jednotlivých algoritmov som sa začala zisťovať od akej dĺžky čísla sú jednotlivé algoritmy efektívnejšie než ostatné. Najprv som však chcela zefektívniť jednotlivé algoritmy. Primitívny algoritmus nemá žiadny parameter, ktorým by sa dal zefektívniť. Ďalšie dva sa dajú vylepšiť vhodnou voľbou konštánt *KAR_THR* a *SS_THR*.

Konštanta *KAR_THR* sa používa v algoritme *KARACUBA* v kroku `1` a udáva dĺžku v 32-bitových číslicách, kde pre čísla menšej dĺžky ako hodnota tejto konštanty, sa nevolajú ďalšie kroky algoritmu.

Konštanta *SS_THR* sa používa v algoritme *NÁSOBSS* a $2^{SS\_THR}$ je konštanta v kroku `1`, ktorá určuje, že pre čísla dĺžky menšej, sa násobia algoritmom *KARACUBA* a algoritmus skončí. Teda *SS_THR* ovplyvňuje hĺbku rekurzie. Dĺžka sa udáva v bitoch.

Prvé meranie rýchlosti algoritmov som previedla s minimálnymi konštantami. Výsledky môjho merania sú znázornené na grafe 1 v prílohách. Graf odpovedajúci Schönhageovmu-Strassenovmu algoritmu je „schodovitý“, pretože program pridáva pred čísla nuly, aby mali dĺžku najbližšej väčšej mocniny dvojky a preto nastáva skok v mocninách dvojky.

**1.1. Algoritmus *KARACUBA*.** Na vylepšenie algoritmu *KARACUBA* som merala čas výpočtu tohto algoritmu s hodnotami *KAR_THR* v rozmezí od 5 do 40 a porovnávala som tieto časy s primitívnym algoritmom. Zaznamenané časy znázorňuje graf 2. Po preskúmaní grafu (je vidieť po postupnom odoberaní grafov pre jednotlivé hodnoty konštanty) som zistila, že pri hodnote konštanty 22 dosahuje algoritmus v porovnaní s primitívnym najlepšie časy. Použila som tento poznatok tak, že za konštantu *KAR_THR* som dosadila hodnotu 22. Táto konštanta zároveň určuje hranicu, kde je algoritmus *KARACUBA* je rýchlejší ako primitívny algoritmus.

**1.2. Algoritmus *NÁSOBSS*.** Pri analýze *NÁSOBSS* je potrebné si uvedomiť, že je to modulárny algoritmus, ktorý počíta $uv \bmod 2^n + 1$, pre vstup $u, v$ dĺžky $n = 2^k$. Ak chceme výsledok $uv$, musíme predĺžiť $u$ a $v$ pridaním núl na začiatok na dĺžku $2n$ a počítat $uv = uv \bmod 2^{2n} + 1$. Teda pri porovnávaní dĺžky trvania výpočtu $uv$ počíta algoritmus *NÁSOBSS* so vstupmi až štyrikrát dlhšími ako algoritmus *KARACUBA*. (Pretože vstupy sú umelo predĺžené dvakrát a potom ešte doplnené na dĺžku $2^k$.)

Najprv som hľadala optimálnu konštantu, ktorá rozhoduje, či sa v kroku `1` má volať rekurzívne Schönhageov-Strassenov algoritmus alebo rekurziu ukončiť a použiť algoritmus *KARACUBA*. Túto konštantu som označila *SS_THR*. Dôležité je uvedomiť si, že v tomto kroku potrebujeme modulárny výsledok, teda $uv \bmod 2^n + 1$ pro vstupy $u$, $v$ dĺžky $n = 2^k$, takže pri volaní algoritmu *NÁSOBSS*

sa neprevádza predlžovanie. Ak použijeme algoritmus *KARACUBA*, musíme vypočítat ešte $uv \bmod 2^n + 1$. Pre optimalizáciu kroku 1 teda neporovnávame rýchlosť algoritmu *NÁSOBSS* a *KARACUBA* pre celočíselné násobenie, ale rýchlosť modulárneho Schönhageovho-Strassenovho algoritmu oproti operácii „násobenie pomocou *KARACUBA* a modulo“. Naviac ma zaujímajú len vstupy dĺžky $2^k$, pretože pri rekurzívnom volaní budú mať vstupy vždy dĺžku $2^k$.

Aby som zistila najvhodnejšiu konštantu *SS_THR* urobila som sériu meraní pre rovnaké vstupy a rôzne konštanty *SS_THR*. Previedla som merania času algoritmu *NÁSOBSS* s konštantou *SS_THR* s hodnotami od 9 do 22. Graf 3 zobrazuje výsledok mojich meraní. Po analýze grafu som zistila, že najlepšie časy dosahuje algoritmus s konštantou 13, ktorá odpovedá dĺžke 256 32-bitových číslic. Pomocou tohto zistenia som algoritmus vylepšila dosadením 13 za *SS_THR*.

## 2. Porovnanie algoritmov

V poslednom meraní som porovnávala efektívnosti vylepšených verzií algoritmov. Rýchlosti jednotlivých algoritmov sú zaznamenané v grafe 4. Už z druhého merania vieme, že algoritmus *KARACUBA* je efektívnejší než primitívny algoritmus pre čísla, ktoré sú dlhšie ako 22 číslic.

Aby sme zistili hranicu, od ktorej je *NÁSOBSS* lepší než *KARACUBA* pre násobenie celých čísel, sa stačí pozrieť na grafy 4, 4a, 4b.

Na grafu 4a je vidieť, že kvôli schodovitému priebehu grafu Schönhageovho-Strassenovho algoritmu sú intervaly, kde je lepší *NÁSOBSS* a intervaly, kde je lepší *KARACUBA*. Mohli by sme sa v optimálnom algoritme pre násobenie rozhodovať, aký algoritmus použijeme podľa toho, v ktorom intervale ležia vstupné hodnoty. V tomto prípade by bol Schönhageov-Strassenov algoritmus použiteľný už pre čísla dĺžky medzi 900 a 1024 číslic. Ak by sme nechceli rozhodovací proces komplikovať a chceli by sme poznať približnú hranicu, kedy je *NÁSOBSS* efektívnejší než *KARACUBA*, mohli by sme vziať napríklad hranicu 1380. (Pri dĺžke okolo 2200 je síce algoritmus *KARACUBA* rýchlejší, ale asi len 1.15 krát.)

## 3. Záver

Násobenie je jedna zo základných výpočtových operácií, na ktorú sa prevádzajú aj ďalšie operácie, napríklad zistenie najväčšieho spoločného deliteľa. Vo väčšine algoritmov, kde sa používa, významne ovplyvňuje celkový čas výpočtu a preto je dôležité mať čo najoptimálnejší spôsob výpočtu.

Na násobenie celých čísel existuje mnoho algoritmov, rôzne komplikovaných a s rôznymi asymptotickými zložitosťami. Na princípe rozdeľuj a panuj sú založené napríklad Karacubov algoritmus (so zložitosťou $O(n^{\log_2 3})$) a ďalšie rozpracovanie jeho základnej myšlienky v rôznych verziách algoritmu Toom-Cook ($O(n^{1+\varepsilon})$, viď [**K**]). Ďalším typom algoritmov na násobenie sú algoritmy založené na použití rýchlej Fourierovej transformácie. Medzi takéto patria napríklad „One-prime FFT multiplication“ [**YL**], „Three-primes FFT multiplication“ (tieto algoritmy majú veľkosť vstupu obmedzenú, teda nemajú žiadnu asymptotickú zložitosť). Ďalej tu patrí zjednodušený Schönhageov-Strassenov algoritmus ($O(n(\log n)^{1+\varepsilon})$, [**Y**]), Strassenov FFT algoritmus využívajúci komplexné čísla ($O(n \log n (\log\log n)^{1+\varepsilon})$, [**K**]) a nakoniec i Schönhageov-Strassenov algoritmus ($O(n \log n \log\log n)$, [**W**]), ktorý má najlepšiu asymptotickú zložitosť zo všetkých.

Schönhageov-Strassenov algoritmus je dôležitý najmä z teoretického hľadiska, pretože umožňuje zkonštruovať subkvadratické algoritmy na množstvo problémov ako napr. test prvočíselnosti.

Táto práca sa zaoberá implementáciou dvoch z týchto algoritmov a to algoritmu Karacuba, ktorý je nejjednoduchší, ale má najväčšiu asymptotickú zložitosť a algoritmom Schönhageovým-Strassenovým, ktorý je najkomplikovanejší, ale má najmenšiu známu asymptotickú zložitosť. Moje zistenia sú popísané vyššie.

# Literatúra


[K] Donald E. Knuth, *The art of computer programing, Volume 2, Seminumerical algorithms*, Addison-Wesley, 1998.

[M] Roman E. Maeder, *Storage allocation for the Karatsuba integer multiplication algorithm*, Lecture Notes in Comp. Sci. **722** (1993), 59–65.

[W] Franz Winkler, *Polynomial Algorithms in Computer Algebra*, Springer, 1996.

[Y] C.K. Yap, *Fundamental Problems in Algorithmic Algebra*, Oxford University Press, 2000.

[YL] C. Yap and C. Li, *QuickMul: Practical FFT-based Integer Multiplication*, http://cs.nyu.edu/chenli/papers/qmul.ps, 2000.

DODATOK A

# Prílohy