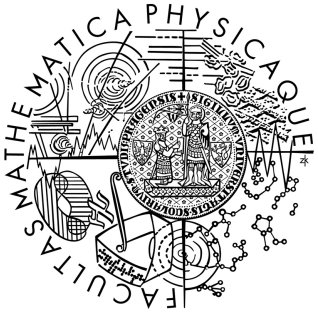
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Kadlček

Optimální strategie faktorizace menších složených čísel

Katedra algebry
Vedoucí bakalářské práce: Doc. RNDr. Aleš Drápal, CSc.

Studijní program: Matematika
Studijní obor: Matematické metody informační bezpečnosti

2006

Chtěl bych poděkovat Mgr. Janu Zvánovcovi za rady ohledně formalizace jazyka této bakalářské práce. Dále bych rád poděkoval Ondřeji Kunčarovi za pomoc se softwarovou stránkou věci.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 10.08.2006 Tomáš Kadlček

Obsah

Název práce: Optimální strategie faktorizace menších složených čísel
Autor: Tomáš Kadlček
Katedra (ústav): Katedra algebry
Vedoucí bakalářské práce: Doc. RNDr. Aleš Drápal, CSc.
e-mail vedoucího: Ales.Drapal@mff.cuni.cz
Abstrakt: Cílem práce bylo testovat tři algoritmy implementované v kvadratickém sítu, které je veřejně k dispozici na webových stránkách katedry algebry MFF (zde [3]). Jejich úkolem v rámci algoritmů MPQS/SIQS je rozkládat kladná čísla na čísla řádu nejvýše unsigned int (v C++), tj. do 32 bitů délky včetně. Tato činnost je nutná při spuštění varianty double large prime variation (DLPV), kdy rozkládáme čísla, která se ne zcela rozložila do faktorizační báze. Algoritmy dostupné pro testování byly: Pollard–ρ, Pollard $p-1$ a CFRAC. Metoda eliptických křivek nebyla dosud implementována. Porovnávání metod bylo provedeno na několika odlišných počítačích. Výsledkem plynoucím z měření je fakt, že pro rozkládání čísel delších než 70 cifer s použitím varianty DLPV je nejvhodnější nejdříve spustit $p-1$ algoritmus a pokud v rozkládání neuspěje, pak jej doplnit algoritmem ρ nebo CFRAC. Zrychlení celého algoritmu způsobené tímto optimalizovaným dílčím rozkládáním se pohybuje v řádu 5–10%.

Klíčová slova: Pollard–ρ, Pollard $p-1$, CFRAC, faktorizace, kvadratické síto

Title: Optimal Factorization Strategy for Smaller Composite Numbers
Author: Tomáš Kadlček
Department: Department of Algebra
Supervisor: Doc. RNDr. Aleš Drápal, CSc.
Supervisor's e-mail address: Ales.Drapal@mff.cuni.cz
Abstract: The goal of this paper was to test three algorithms implemented in the kvadratic sieve which is offered to be used by public on web pages of the department of algebra of MFF (see [3]). Their purpose in the MPQS/SIQS algorithms is to factor positive integers to integers of size of unsigned int (in C++), i.e.integers of the maximum size 32 bits. This factorization is necessary for the double large prime variation (DLPV), where we need to factor numbers, which haven't fully factored over the factor base. Algorithms available to be tested and compared was: Pollard–ρ, Pollard $p-1$ and CFRAC. Eliptic curve method wasn't implemented yet. The comparison of these methods was made with using several different computers. The main result we obtained from the measuring is that the partial factorization job, when factoring numbers greater than 10^{70} with variant DLPV, should be first done by $p-1$ algorithm and if it fails, than ρ or CFRAC should be run because they are successful everytime. This optimalization accelerates the whole algorithm by 5–10%.

Keywords: Pollard–ρ, Pollard $p-1$, CFRAC, factorization, quadratic sieve

Kapitola 1

Zkoumané faktorizační algoritmy

V první kapitole jsou popsány tři faktorizační metody: Pollard $p-1$, Pollard–ρ a CFRAC, které byly implementovány v rámci algoritmů MPQS/SIQS (Multiple Polynomial Quadratic Sieve / Self Initializing Quadratic Sieve) na katedře algebry MFF. Dále budeme psát QS pro označení obou algoritmů, z nichž při startu programu vybíríme pouze jeden parametrem -q (pro MPQS) nebo -s (pro SIQS). Výsledky uvedené v kapitole 2 byly naměřeny výhradně s použitím algoritmu SIQS, jenž je obecně považován za (dvakrát) rychlejší.

Předpokladem pro správný chod algoritmů je, že na vstupu dostanou složené číslo, které se dále pokusí rozložit na dva činitele. Fakt, že je číslo složené, je výpočetně snadné ověřit např. Rabin-Millerovým testem. Varianta QS používající double large prime variation nám však zaručuje, že čísla vstupující do níže popsaných faktorizačních algoritmů už složená jsou. Zdrojové texty v C++ jsou dostupné ke stažení zde [3]. Metoda eliptických křivek nebyla dosud dokončena, proto nebyla v rámci této práce ani testována a porovnána s ostatními metodami.

Alternativní popisy těchto faktorizačních metod může čtenář nalézt také v jiných českých textech, např. [2] a [6] nebo v anglicky psané učebnici teorie čísel [1].

1.1 Pollardova $p-1$ metoda

Tato metoda faktorizace není použitelná v obecném případě. Funguje rychle pouze pro čísla určitých speciálních vlastností. Nechť tedy rozkládané číslo N je rovno $N = p \cdot q$, kde p je prvočíslo a q je „zbylý“ činitel. Potom očekávanou speciální vlastností je rozložitelnost čísla $p-1$ na součin malých prvočísel. Přesněji:
$p - 1 = {r_1}^{s_1} \dots {r_k}^{s_k}$, kde $r_1, \dots, r_k$ jsou prvočísla menší nebo rovna nějaké zadané

hranici B, $s_1, \ldots, s_k$ jsou kladná celá čísla. Využívá se tedy toho, že číslo $p-1$ má jiné vlastnosti než prvočíslo p.

Definice: Jestliže pro přirozené číslo N platí, že $N = p_1 \ldots p_k$, kde $p_1, \ldots, p_k$ jsou prvočísla menší nebo rovny číslu B, potom říkáme, že číslo N je *B–hladké* (*B-smooth*). Jestliže se v rozkladu čísla N objevují i vyšší mocniny prvočísel, tj. $N = {p_1}^{l_1} \ldots {p_k}^{l_k}$ a přitom ${p_i}^{l_i} \leq B$ pro všechny $i = 1, \ldots, k$, potom říkáme, že číslo N je *B–mocné* (*B-powersmooth*).

Nejmenší společný násobek čísel od 1 po B označme $\mathrm{LCM}\,[1 \ldots B]$. Jestliže je číslo $p-1$ *B–mocné*, potom je zřejmé, že $p-1$ dělí $\mathrm{LCM}\,[1 \ldots B]$, neboť všechny prvočíselné mocniny dělící $p-1$ jsou menší nebo rovny B. Dále pro $x, y \in \mathbb{Z}$ nechť $\mathrm{GCD}(x, y)$ značí největší společný dělitel čísel x a y, kde bereme $\mathrm{GCD}(x, y) \geq 1$ a pro $x = y = 0$ hodnotu $\mathrm{GCD}(x, y)$ nedefinujeme.

Buď p prvočíslo. Potom pro libovolné $a \in \mathbb{Z}$, $\mathrm{GCD}(a, p) = 1$, platí podle Fermatovy věty:

$$a^{p-1} \equiv 1 \pmod{p}.$$

Tedy a umocněné na jakýkoliv násobek čísla $p-1$ bude opět kongruentní 1 modulo p.

Hodnotu p neznáme, a zvolíme náhodně, např. $a = 2$.

Platí (označme):

$$y := a^{\mathrm{LCM}\,[1 \ldots \mathrm{B}]} \equiv 1 \pmod{p}$$

odtud

$$y - 1 \equiv 0 \pmod{p}.$$

Vidíme, že jsme našli číslo $y-1$ dělitelné hledaným prvočíslem p. Proto v algoritmu budeme toto číslo y počítat modulo N a následně určíme $\mathrm{GCD}(y-1, N)$.

Kroky základního $p-1$ algoritmu:

1. Zvolíme a a hranici B.
2. Spočítáme $y := a^{\mathrm{LCM}\,[1 \ldots \mathrm{B}]} \bmod N$.
3. Vypočteme $\mathrm{GCD}((y-1), N)$.
4. Je-li $\mathrm{GCD}((y-1), N)$ různé od jedné a od N zároveň, pak je rovno nějakému děliteli čísla N. Algoritmus uspěl a končí. Je-li $\mathrm{GCD}((y-1), N)$ rovno jedné nebo N, pak algoritmus ohlásí neúspěch a končí.

V kroku (4) mohou nastat dvě situace, kdy algoritmus neuspěje při rozkládání čísla N:

- GCD$((y-1), N) = 1$ nastává v případě, kdy $p-1$ není *B–mocné*.
- GCD$((y-1), N) = N$ nastává v případě, kdy $y - 1 = 0$. Tři Tabulky v podkapitole 2.2 ukazují, že s vysokou pravděpodobností nastala situace, že $\varphi(N)$ je *B–mocné* (φ je Eulerova funkce). Algoritmus v tomto případě selže pokaždé, protože $y - 1$ je násobkem p i q zároveň. Jedinou možností, jak se tomu vyhnout, je zmenšit hodnotu hranice B a začít algoritmus znova (bohužel tento postup nevede stoprocentně k cíli, protože $p - 1$ i $q - 1$ mohou v rozkladu obsahovat stejné nejvyšší prvočíslo). Jiná situace vedoucí ke GCD$((y-1), N) = N$ je, když $y-1$ je (velmi nepravděpodobně) q násobkem prvočísla p. Tehdy stačí změnit volbu a a začít znova od začátku.

Existuje ale i vylepšení, které používá vedle hranice B také hranici B_1. Tento rozšířený algoritmus lze použít pro rozklad takových čísel N, která mají dělitele p s vlastností: $p - 1 = fg$, kde f je *B–mocné* a g je prvočíslo, $B < g \leq B_1$.

Podobně jako v předchozím případě platí:

$$a^{g \cdot \mathrm{LCM}\,[1\ldots\mathrm{B}]} \equiv 1 \pmod{p}.$$

Nejprve spočítejme $y := a^{\mathrm{LCM}\,[1\ldots\mathrm{B}]} \bmod N$. Dále potřebujeme zjistit, zda pro některé prvočíslo g, $B < g \leq B_1$, platí: $y^g \equiv 1 \bmod N$. Nechť $p_1 < \cdots < p_k \leq B$ jsou všechna prvočísla menší nebo rovna B. Tato máme v paměti uložena a používáme je pro výpočet y. Dále nechť $B < p_{k+1} < \cdots < p_{k+l} \leq B_1$ jsou všechna prvočísla mezi B a B_1 včetně. Tato jsou v paměti uložena ve formě rozdílů $d_{k+i} = p_{k+i} - p_{k+i-1}$, $i = 1, \ldots, l$. Zabírají tak v počítači méně paměti než kdybychom měli uložena jednotlivá prvočísla.

Předpočítejme hodnoty $y^{d_{k+i}} \bmod N$ pro všechna $i = 1, \ldots, l$. Nyní stačí hodnotu $b := y^{p_k} \pmod N$ postupně měnit násobením hodnotami $y^{d_{k+1}}$, $y^{d_{k+2}}$, ... (vše počítáme (mod N)), dokud nebude $b = y^g$. Vzhledem k tomu, že prvočíslo g neznáme, poznáme tento stav tak, že po každé změně hodnoty b vynásobením testujeme, zda GCD$(y-1, N)$ je zároveň různé od 1 a od N. Pokud ano, máme hledaného dělitele.

Kroky rozšířeného $p-1$ algoritmu:

1. Zvolíme a, hranice B a B_1.
2. Spočítáme $y := a^{\mathrm{LCM}\,[1\ldots\mathrm{B}]} \bmod N$.
3. Vypočteme GCD$((y-1), N)$.

4. Je-li $\mathrm{GCD}((y-1), N)$ různé od jedné a od N zároveň, pak je rovno nějakému děliteli čísla N. Algoritmus uspěl a končí. Je-li $\mathrm{GCD}((y-1), N)$ rovno jedné, pak algoritmus pokračuje krokem (5). Je-li $\mathrm{GCD}((y-1), N)$ rovno N, pak algoritmus ohlásí neúspěch a končí.

5. Předpočítáme hodnoty $y^{d_{k+i}} \bmod N$ pro všechna $i = 1, \ldots, l$, kde d_{k+i} jsou tabelované rozdíly prvočísel. Index i nastavíme roven jedné.

6. Pro dané i násobíme hodnotu b předpočítanou hodnotou z kroku (3), tj. $b := b \cdot y^{d_{k+i}} \bmod N$ a dále spočítáme $\mathrm{GCD}(y-1, N)$.

7. Je-li $\mathrm{GCD}(y-1, N)$ různé zároveň od jedné a od N, pak už je rovno nejakému děliteli N a algoritmus končí. Je-li $\mathrm{GCD}(y-1, N)$ rovno N pro některé i, algoritmus ohlásí selhání a skončí. Je-li $\mathrm{GCD}(y-1, N)$ rovno 1 a $i \leq l$, zvyšíme index i o jedna a pokračujeme od kroku (6). Je-li $\mathrm{GCD}(y-1, N)$ rovno 1 a $i = l$, algoritmus ohlásí selhání a skončí.

Selhání algoritmu v kroku (4) je popsáno výše v poznámce pro jednodušší verzi $p-1$ algoritmu. Selhání algoritmu v kroku (7) může nastat ze dvou důvodů:

- Jestliže v průběhu algoritmu bylo stále $\mathrm{GCD}(y-1, N)$ rovno jedné, pak víme jistě, že rozkládané číslo N nemá žádného dělitele p s požadovanými vlastnostmi, co se týká hodnot f a g.

- Jestliže v kroku (7) vyjde největší společný dělitel roven N, pak jsme narazili na situaci, kdy řád a v $\mathbb{Z}_N$ dělí (exponent) $p_{k+i} \cdot \mathrm{LCM}\,[1 \ldots \mathrm{B}]$, kde i je index, při kterém došlo k ukončení algoritmu. Tento případ se dá řešit návratem na krok (1) a změnou hodnoty a. Hranice B a B_1 není třeba měnit.

Obě verze (základní i rozšířená) $p-1$ algoritmu bývají v literatuře popisovány trochu jinak. Zde jsou popisy uvedeny ve formě, která odpovídá optimalizovanému kódu $p-1$ algoritmu v rámci QS, tedy ve formě, ke které se vztahují měření v podkapitole 2.2.

Tato metoda faktorizace dokáže být velmi rychlá i pro velká čísla N, protože záleží jenom na *B–mocnosti* jejich dělitelů zmenšených o jedna. V nejhorším případě může být číslo N rovno součinu $p \cdot q$, kde $p = (2a+1)$, $q = (2b+1)$, p, q, a, b jsou prvočísla, p, q jsou zhruba stejné velikosti. V tomto případě má algoritmus složitost $\mathrm{O}(N^{\frac{1}{2}})$, protože je třeba volit hranici B vyšší než p nebo q, a není rychlejší než metoda náhodného dělení.

Budeme-li v základní verzi algoritmu předpokládat pouze *B–hladkost* $p-1$, potom musíme v kroku (2) provést několikrát mocnění $a^{\mathrm{LCM}\,[1\ldots\mathrm{B}]} \bmod N$ za sebou (to odpovídá jednomu umocnění $a^{(\mathrm{LCM}\,[1\ldots\mathrm{B}])^{\mathrm{r}}} \bmod N$, pro nějaké zvolené $r \geq 2$),

čímž v exponentu získáme $r--$násobné mocniny oproti původní variantě s jedním umocněním. Toto jistě zpomalí celý algoritmus, ale mnohdy je to výhodnější, než kdybychom museli zvolit hranici B tak, aby $p-1$ bylo B–*mocné*, a následně jednou umocnili $a^{\mathrm{LCM}\,[1...B]} \mod N$.

Existují i variace na Pollardovu $p-1$ metodu, kdy se využívá např. hodnot $p+1$, p^2+p+1, p^2-p+1 atd. Zde se bere prvek a jako prvek těles $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^3}$, ...

V literatuře se uvádí, že v praxi je schůdná hranice řádově $B = 10^6$ a $B_1 = 10^8$. Výsledky měření rychlosti $p-1$ algoritmu implementovaného a optimalizovaného v SIQS v závislosti na hodnotách B a B_1 jsou uvedeny v podkapitole 2.2.

1.2 Pollardova ρ–metoda

Mějme zobrazení $f : A \longmapsto A$, A je konečná množina, o kterém předpokládejme, že je zcela náhodné. Zvolme prvek $x_0 \in A$ a definujme posloupnost $x_1 = f(x_0)$, $x_2 = f(x_1) = f(f(x_0))$, ..., $x_k = f^k(x_0)$ pro všechny $k \in \mathbb{N}$.

Definice: Pro dané zobrazení f, zvolený prvek x_0 a posloupnost $\{x_k\}_{k=0}^{\infty}$ (viz předchozí odstavec) definujeme periodu m jako nejmenší přirozené číslo takové, že $x_j = x_{j+m}$ pro nějaké přirozené číslo j. Dále definujeme preperiodu l jako nejvyšší přirozené číslo takové, že $x_{l-1} \neq x_{(l-1)+m}$. V případě, že takové l neexistuje, tj. $x_0 = x_m$ (x_0 je „uvnitř“cyklu), definujme $l = 0$.

Podstata časové složitosti Pollard–ρ algoritmu spočívá v tom, že průměrná délka periody náhodného zobrazení na množině A je (s rostoucí velikostí A) asymptoticky rovna $\sqrt{\frac{\pi.|A|}{8}}$. Průměrná (asymptoticky) délka preperiody nás zajímat nebude, protože složitost algoritmu je ovlivněna délkou periody.

Budeme sice počítat funkci $f \mod N$, ale pro faktorizační účely je zajímavější její chování mod$p$, kde p je prvočíslo dělící N. Najdeme-li periodu (myšleno modulo p) posloupnosti $\{x_k\}_{k=0}^{\infty}$, dokážeme ji využít k rozkladu N. Odtud je patrné, že pokud $N = p.q$, kde p, q jsou prvočísla řádově stejné velikosti, pak složitost algoritmu roste s $\sqrt{p}$ a tudíž je rovna $O\left(2^{(\frac{n}{4})}\right)$, kde n je binární délka N.

Vztahujme tedy preperiodu a periodu k nějakému zvolenému $x_0 \in \mathbb{Z}_N$ a k funkci $f_p = f \pmod p$, kde p je neznámý dělitel čísla N, a $f : \mathbb{Z}_N \longmapsto \mathbb{Z}_N$ je náhodná fce.

Myšlenka Pollard–ρ metody spočívá v tom, že pokud najdeme $x_i, x_j \in \mathbb{Z}_N$ takové, že $x_i \neq x_j$ a $x_i \equiv x_j \pmod p$, potom víme, že $x_i - x_j$ je násobkem p. Toho využijeme při počítání $\mathrm{GCD}(x_i - x_j, N)$, neboť tento největší společný dělitel je roven nějakému děliteli N. Potřebné k nalezení takovéto dvojice je, aby x_i a x_j byly „za“ preperiodou, tj. $i, j \geq l$, a pak už stačí pro jedno x_i projít asymptoticky $\sqrt{\frac{\pi p}{8}}$ prvků x_j k tomu, abychom našli dvojici s popsanou vlastností. Ještě dodejme,

že rozpoznání vhodné dvojice (x_i, x_j) od ostatních, které nevedou k faktorizaci, se provádí tak, že spočítáme $\mathrm{GCD}(x_i - x_j, N)$ a pokud je větší než jedna, potom jsme našli jak hledanou dvojici, tak i dělitel čísla N (stále platí předpoklad, že $x_i \neq x_j$)).

Pollard a Floyd navrhli postup, kdy se postupně počítají pouze dvojice tvaru (x_i, x_{2i}) a tím se šetří místo v paměti počítače. Je-li x_i v periodické části, tj. $i \geq l$, pak v jistém okamžiku nastane případ, kdy $x_{2i} \equiv x_{i+m} \equiv x_i \pmod{p}$ a tedy testem na $\mathrm{GCD}(x_i - x_{2i}, N)$ obdržíme dělitele čísla N. Počítá se zde třikrát fce f, a to: $x_{i+1} = f(x_i)$, $x_{2(i+1)} = f^2(x_{2i})$. Tomu se lze vyhnout užitím Brentova vylepšení.

Ten navrhnul, aby se za x_i volilo postupně x_{2^s-1}, pro $s = 1, 2, 3, \ldots$, a ke každému takovému x_i se počítaly hodnoty x_j, kde $\frac{3}{2} \cdot 2^s \leq j < 2^{s+1}$. Tvrzení (včetně důkazu) říkající, že existují taková i, j, která splňují předchozí podmínku a přitom jsou vhodná pro Pollard–ρ algoritus, můžete najít např. v [2]. Experimentálně se potvrdilo, že Brentovo vylepšení původního algoritmu navrženého Pollardem a Floydem je rychlejší asi o 25%.

Ještě se vrátíme k odhadu asymptotického chování periody m v závislosti na rostoucím p.

Nechť $\mathrm{P}[m, l]$ značí pravděpodobnost, že posloupnost $\{x_i\}_{i=0}^{\infty}$ získaná iterováním má preperiodu l a periodu m. Je zřejmé, že $x_0, \ldots, x_{m+l-1}$ jsou navzájem různé a $x_{m+l} = x_l$. Předpokládáme-li náhodnost f_p, pak $\mathrm{P}[m, l]$ nezávisí na volbě počátečního x_0.

$$\mathrm{P}[m, l] = \frac{(p-1)\ldots(p-(m+l-1))}{p^{m+l}}$$

$$\mathrm{P}[m, l] = \frac{1}{p} \prod_{1 \leq k < m+l} \left(1 - \frac{k}{p}\right)$$

Pro pevně zvolené p označme $m = \mu \cdot \sqrt{p}$ a $l = \lambda \cdot \sqrt{p}$. Z předchozí rovnice a z Taylorova rozvoje pro $\ln(1+x)$ plyne:

$$\begin{aligned}\ln(p.\mathrm{P}[m, l]) &= \sum_{1 \leq k < (\mu+\lambda)\sqrt{p}} \left(-\frac{k}{p} - \frac{1}{2}\left(-\frac{k}{p}\right)^2 + \frac{1}{3}\left(-\frac{k}{p}\right)^3 - \ldots\right) \\ &= \sum_{1 \leq k < (\mu+\lambda)\sqrt{p}} \left(-\frac{k}{p} + O\left(\frac{k^2}{p^2}\right)\right) = -\frac{(\lambda+\mu)^2}{2} - \frac{\lambda+\mu}{2\sqrt{p}} + O\left(\frac{(\lambda+\mu)^3}{\sqrt{p}}\right).\end{aligned}$$

Odtud hustota rozdělení $\mathrm{P}[m, l]$ je (předpokládáme, že $\lambda + \mu \ll \sqrt{p}$)

$$\frac{1}{p} e^{-\frac{(\lambda+\mu)^2}{2}}.$$

A proto průměrná hodnota dvojice (perioda, preperioda) je

$$\begin{aligned}(m_{\text{prum}}, l_{\text{prum}}) = &\ \sum_{m,l} \mathrm{P}[m,l] \cdot (m,l) \approx \int_0^\infty \int_0^\infty \mathrm{P}[m,l] \cdot (m,l)\,\mathrm{d}m\,\mathrm{d}l \\ \approx &\ \int_0^\infty \int_0^\infty (\mu\sqrt{p}, \lambda\sqrt{p}) \cdot \tfrac{1}{p} e^{-\frac{(\lambda+\mu)^2}{2}} \cdot \sqrt{p}\mathrm{d}\mu \cdot \sqrt{p}\mathrm{d}\lambda. \end{aligned} \tag{1.1}$$

Důkaz asymptotické časové složitosti algoritmu sice předpokládá náhodnost zobrazení, které iterujeme, ale v praxi užívaným zobrazením bývá polynom $x^2+1 \pmod N$ nebo $x^2-1 \pmod N$ (který zřejmě není náhodným zobrazením) a vzniklá posloupnost $\{x_i\}_{i=0}^{\infty}$ se z hlediska délky periody chová obdobně jako posloupnost vzniklá iterováním náhodného zobrazení. Toto bylo ověřeno experimentálně.

Tvrzení 1.2.1: Pro $p \to \infty$ je průměrná hodnota periody m asymptoticky rovna

$$\sqrt{\frac{\pi}{8}} \cdot \sqrt{p}.$$

Důkaz: Ze vztahu (1.1) plyne pro periodu m následující:

$$m_{\text{prum}} \approx \int_0^\infty \int_0^\infty \mu \cdot \sqrt{p} \cdot e^{-\frac{(\lambda+\mu)^2}{2}}\,\mathrm{d}\mu\,\mathrm{d}\lambda = \sqrt{\frac{\pi}{8}} \cdot \sqrt{p}$$

□

Tento integrál byl vyčíslen s použitím školní licence programu Maple7.

1.3 CFRAC

Princip faktorizace pomocí řetězových zlomlů objevili pánové Lehmer a Powers v roce 1931, dnes používanou podobu algoritmu CFRAC publikovali roku 1975 pánové Morrison a Brillhart. Faktorizační metoda pomocí řetězových zlomků je historicky první, která má subexponenciální časovou složitost. I když důkaz byl proveden pouze heuristicky a následně prakticky ověřená časová složitost odhadu odpovídala. Přesněji jde o složitost $\mathrm{O}\left(e^{\sqrt{2 \cdot n \cdot ln(n)}}\right)$, kde n je binární délka faktorizovaného N. Ještě v 70. letech byl CFRAC nejpoužívanější faktorizační metodou. Stejně jako novější a rychlejší metody, kvadratické síto a síto nad obecným číselným tělesem , je založena na fermatově metodě, kdy se hledá kongruence tvaru

$$x^2 \equiv y^2 \pmod N. \tag{1.2}$$

Všechny tyto metody využívají stejný způsob hledání této kongruence, a to tak, že se snaží najít velké množství vztahů

$$x_i{}^2 \equiv p_0{}^{\epsilon_0} \dots p_k{}^{\epsilon_k} \pmod N, \tag{1.3}$$

kde $p_0 < \cdots < p_k$ jsou všechna prvočísla z vybrané faktorizační báze (FB), ϵ_i jsou celé nezáporné exponenty. Speciálně volíme $p_0 = -1$, a také je vhodné mít $p_1 = 2$, viz komentář k lemmatu 1.3.6.

Z výsledné matice exponentů je potom možno vynásobením vybraných řádků získat na pravé straně součin prvočísel z faktorizační báze, kdy každému prvočíslu přísluší sudý exponent. Pak stačí v rovnici (1.2) za x dosadit součin vybraných x_i a za y vzít součin prvočísel z pravé strany, ale pouze s polovičními exponenty. Všechno počítáno mod$N$. Takto získáme x a y splňující vztah (1.2) a s alespoň 50% pravděpodobností jedno z čísel $x-y$ nebo $x+y$ má společný netriviální dělitel s rozkládaným číslem N. Rovnice 1.2 má (za předpokladu, že číslo N má právě dva různé prvočíselné dělitele) čtyři různá řešení, z nichž dvě řešení tvaru $x \equiv \pm y \pmod{N}$ nejsou vhodná k rozkladu N, zatímco dvě řešení tvaru $x \not\equiv \pm y \pmod{N}$ vedou k nalezení netriviálního dělitele N.

CFRAC využívá pro generování kongruencí typu (1.3) rozvoj $\sqrt{kN}$ do řetězového zlomku. Význam konstanty k je vysvětlen v komentáři k lemmatu 1.3.6.

Tedy

$$\sqrt{N} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

Označme $\frac{p_n}{q_n} = [a_0, a_1, \ldots, a_n]$, kde $a_i \in \mathbb{Z}$ pro všechna $i = 1, \ldots, n$. Důkazy tvrzení uvedených v této podkapitole nebudeme rozepisovat, protože jsou uvedeny v mnoha textech zabývajících se řetězovými zlomky. Uvedená tvrzení a lemmata byla vybrána ze sady textů [2], konkrétně z části o faktorizaci čísel pomocí řetězových zlomků, za důležitá pro pochopení jednotlivých kroků algoritmu CFRAC, jak je popsán v tomto textu na straně 12.

Tvrzení 1.3.1: Nechť $C_0 = 0$ a $D_0 = 1$. Nechť je dále definována posloupnost $\{C_i\}_{i=0}^{\infty}$ a $\{D_i\}_{i=0}^{\infty}$ rekurzivně:

$$C_{n+1} = a_n \cdot D_n - C_n \qquad D_{n+1} = \frac{N - C_{n+1}^2}{D_n}. \tag{1.4}$$

Potom pro a_n platí:

$$a_n = \left\lfloor \frac{C_n + \sqrt{N}}{D_n} \right\rfloor \tag{1.5}$$

a navíc C_n a D_n jsou celá čísla pro všechna $n \in \mathbb{N}$.

Důkaz: viz např. [2]

Tvrzení 1.3.2: Pro každé přirozené n platí:

$$p_{n-1}^2 - Nq_{n-1}^2 = (-1)^n D_n. \tag{1.6}$$

Důkaz: viz např. [2]

Z předchozího plyne skutečnost, že

$$p_{n-1}^2 \equiv (-1)^n D_n \pmod{N} \tag{1.7}$$

a pokud se dá rozložit pravá strana kongruence do FB, potom získáme vztah typu (1.3).

Při počítání $pn-1$ použijeme vztah $p_{n+1} \equiv a_{n+1} \cdot p_n + p_{n-1} \pmod{N}$. Dále je třeba počítat C_n, D_n a a_n. Je snadné počítat C_n, jde jenom o násobení. Problém s náročnějším dělením vyvstává při počítání D_n a a_n. Můžeme ale využít fintu, která je popsaná např. v [2].
Nechť $g = \left\lfloor \sqrt{N} \right\rfloor$, potom platí (úpravou (1.5))

$$C_n + g = a_n D_n + r_n, \qquad 0 \leq r_n < D_n. \tag{1.8}$$

Čili s užitím předchozího a (1.4) máme

$$C_{n+1} = g - r_n. \tag{1.9}$$

Dále pro výpočet D_n získáme rychlejší algoritmus s pomocí následujícího lemmatu:

Lemma 1.3.3: Pro všechna $n \in \mathbb{N}$, $n \geq 2$ platí:

$$D_{n+1} = D_{n-1} + a_n(r_n - r_{n-1}). \tag{1.10}$$

Důkaz: viz např. [2]

Uvádíme kroky algoritmu pro výpočet hodnot C_k, D_k, a_k, p_k, r_k a g. Inicializace faktorizační báze stejně jako postup generování rovnic ze vztahů tvaru 1.7 a následné řešení soustavy lineárních rovnic, to jsou kroky CFRACu, které zde popisovat nebudeme, protože jsou intuitivně zřejmé.

Kroky algoritmu CFRAC:

1. Stanovíme malý koeficient k (viz komentář k lemmatu 1.3.6.) a mez M pro počet opakování kroků (4)–(8).

2. $C_0 := 0$, $D_0 := 1$, $p_{-1} := 1$, $r_0 := 0$ a $g := \left\lfloor \sqrt{N} \right\rfloor$.

3. $D_1 := kN - g^2$, $a_0 := g$, $p_0 := g$, $C_1 := g$, $n := 1$.

4. Dělením se zbytkem spočítáme a_n a r_n ze vztahu (1.8).

5. $p_n := a_n \cdot p_{n-1} + p_{n-2} \bmod N$.

6. podle (1.9) $C_{n+1} = g - r_n$.

7. podle (1.10) $D_{n+1} = D_{n-1} + a_n(r_n - r_{n-1})$.

8. $n := n + 1$ a pokud $n < M$, pokračujeme od kroku (4), jinak ukončíme výpočet řetězového zlomku a s ním i výpočet výše uvedených proměnných.

Podstatné pro implementaci je také vědět, v jakých rozmezích se jednotlivé proměnné pohybují. Např. p_n počítáme modulo N, takže bude jistě menší než N. Ostatní proměnné můžeme odhadnout následujícím lematem:

Lemma 1.3.4: Hodnoty $C_n + g$, D_n, a_n a r_n jsou z intervalu $\left\langle 0,\ 2\sqrt{kN} \right\rangle$.

Důkaz: viz např. [2]

V průběhu algoritmu registrujeme, že a_n je velmi malé číslo, často je rovno jedné, proto můžeme v krocích 4) a 5) nahradit dělení se zbytkem rychlejším odečítáním.

Vraťme se k volbě malého koeficientu k. Některá prvočísla totiž v žádném případě nemohou dělit D_n. Tato vlastnost je závislá na hodnotě kN, k čemuž se snadno přijde pomocí vztahu (1.6).

Lemma 1.3.5: Nechť liché prvočíslo p dělí hodnotu D_n. Potom Legendrův symbol $\left(\frac{p}{kN}\right)$ je roven 0 nebo 1.

Důkaz: viz např. [2]

Pro implementaci algoritmu na počítači je vhodné, aby D_n bylo dělitelné nějakou mocninou dvojky, protože dělení dvěma je velice snadné (=posun bitů v paměti). Můžeme využít následující lemma:

Lemma 1.3.6: Nechť 2 nedělí kN. Potom:
pokud 4 dělí D_n, pak $kN \equiv 1 \pmod 4$,
pokud 8 dělí D_n, pak $kN \equiv 1 \pmod 8$.

Důkaz: viz např. [2]

Kapitola 2

Měření na kvadratickém sítu

Algoritus QS hledá rovnice tvaru (1.3) stejně jako CFRAC. V rámci varianty double large prime variation jde i rovnice typu:

$$x_i^2 = p_1^{\epsilon_1} \dots p_k^{\epsilon_k} \cdot R \cdot S \pmod{N},$$

kde p_i jsou z faktorizační fáze (FB), ϵ_i jsou celá nezáporná čísla a R,S jsou prvočísla mimo FB (tedy jsou vyšší než hranice FB zadaná při startu programu na příkazové řádce - označme si ji mez_{FB}). Takto získáme vyšší počet rovnic a pokud se nám podaří najít několik rovnic takových, že po jejich vynásobení budou na pravé straně výsledné rovnice všechna prvočísla v sudé mocnině, potom dostaneme další rovnici typu (1.3). Přítomnost prvočísel mimo FB nijak neovlivňuje krok algoritmu QS, ve kterém se řeší soustava lineárních rovnic za účelem získání rovnice typu (1.2).

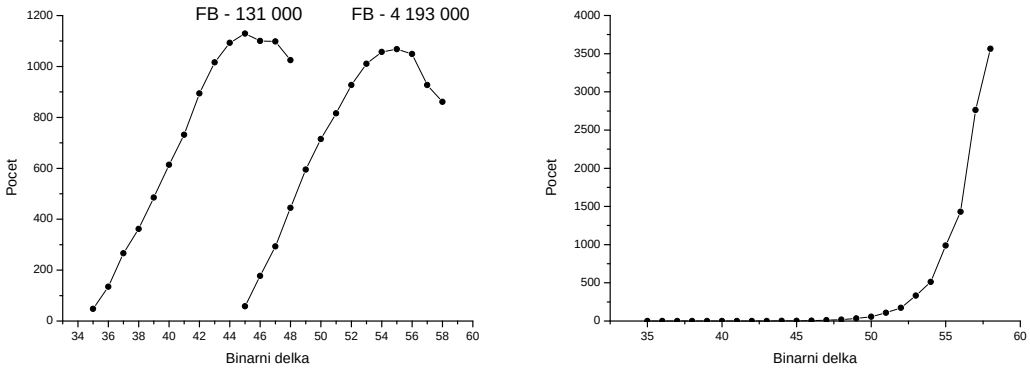
Součin prvočísel R, S nazývejme „zbytek“. Pokud „zbytek“ náleží do intervalu $I = \langle \text{mez}_{\text{FB}}{}^2, (128 \cdot \text{mez}_{\text{FB}})^2 \rangle$, potom se jej algoritmus QS pokusí rozložit. Zde je volba dolní hranice zřejmá, a horní hranice byla zvolena tak, aby čísla R, S byla v rozmezí 14 bitů délky. Maximální, pro QS přípustná, binární délka prvočísel je však omezena obvyklou velikostí integeru, tzn. 32 bitů (do budoucna toto není problém, i kdyby měl integer 64 bitů, algoritmus bude fungovat dál bez problémů). Rozmezí 14 bitů se v praxi ukázalo být dostatečné.

2.1 Porovnání Pollard–ρ a CFRAC

Následující uvedené grafy a diskuse výsledků jsou vztahovány k měření provedenému na počítači, který je k dispozici v počítačové laboratoři v budově MFF na Malé Straně. Konkrétně procesor AMD Opteron 144 (1.0 GHz, 1024 KB cache), paměť 1024 MB RAM, operační systém GNU/Linux.

Množství rozkládaných čísel dané binární délky je ovlivněno parametrem - *checkX* z příkazové řádky, který ovlivňuje citlivost té funkce algoritmu QS, jež rozhoduje, zda-li bude „zbytek“ dále rozkládán některým z testovaných algoritmů. Při měření byla zvolena hodnota $checkX = -1$, doporučená autory QS.

Bereme-li binární délku „zbytků“ generovaných prosíváním jako náhodnou veličinu s hodnotami v intervalu I, potom její rozdělení je znázorněno grafem na obrázku 2.1 vlevo, který ukazuje počet vygenerovaných „zbytků“ dané binární délky. Data byla nasbírána při rozkládání 65 ciferného čísla N s využitím 10000 „zbytků“ pro obě velikosti FB zvlášť a s hodnotou parametru $checkX = -1$.



Obrázek 2.1: Rozdělení „zbytků“

Porovnejme tento graf s grafem vpravo, který odpovídá náhodnému rozdělení deseti tisíc čísel tvaru $N = p \cdot q$, pro p, q prvočísla, v intervalu 35 až 58 bitů délky. S využitím vzorce pro asymptotickou hustotu prvočísel $\pi(n) = \frac{n}{\ln(n)}$, kde $\pi(n)$ je počet prvočísel menších než n, snadno odvodíme, že $\xi(k)$:=počet prvočísel binární délky k je asymptoticky roven: $\xi(k) = \pi(k) - \pi(k-1) = 2^{k-1}\left[\frac{2}{n \cdot \ln 2} - \frac{1}{(n-1)\cdot \ln 2}\right]$. Každý „zbytek“ binární délky n se rozkládá na součin dvou prvočísel binárních délek k a $n-k-1$ pro nějaké $k \geq 2$. Tedy počet „zbytků“ délky n je dán binomickým součtem $\sum_{i+j-1=n} \xi(i)\xi(j)$. Ten má evidentně exponenciální růst. Graf na obrázku 2.1 vlevo však takový růst nemá a vyplývá z něj, že je nutné umět rychle rozkládat „zbytky“ všech délek a ne soustředit se na rozkládání těch, které se vyskytují nejčastěji, jako by tomu bylo v případě grafu vpravo na tomtéž obrázku. Tvar křivky na grafu 2.1 je dán tím, že postup od rozkládaného čísla prosíváním až ke „zbytku“ probíhá po krocích, kdy toto číslo dělíme jeho prvočíselnými děliteli zastoupenými v FB. Proto se často stává, že „zbytek“ má nižší binární délku než by měl při náhodném

výběru z intervalu 14 bitů délky. Shrnutí: algoritmus QS prosíváním neprodukuje „zbytky“ rozložené náhodně v uvažovaném intervalu.

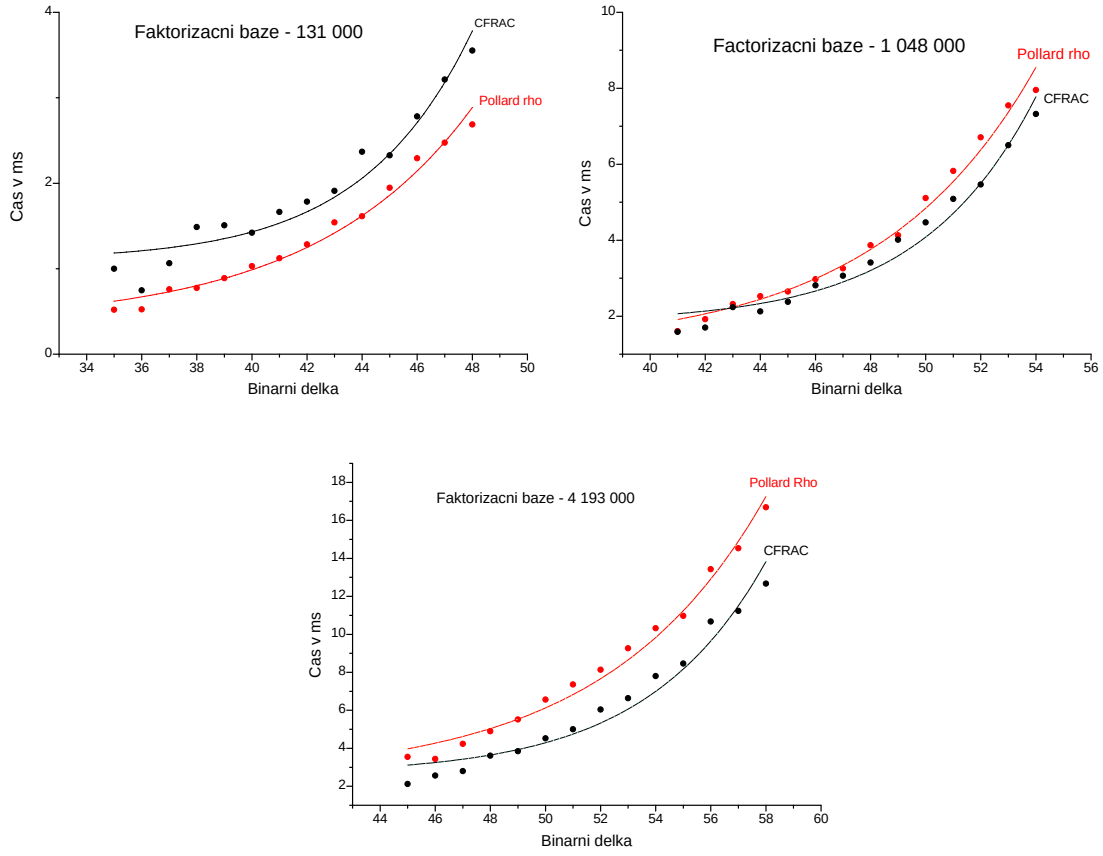
Samotné měření rychlosti algoritmů Pollard–ρ a CFRAC probíhalo při rozkládání čísla N řádu 10^{65}, při různých velikostech faktorizační báze. Použito vždy pět tisíc číslech („zbytků“). Aproximace výsledků byla spočtena a vykreslena do grafů s využitím programu Origin7, kdy byla pro Pollard–ρ použita funkce typu $y = A_0 + A_1 \cdot \exp(\frac{x}{4})$ a pro CFRAC funkce typu $y = A_0 + A_1 \cdot \exp(2x \ln x)$. Tyto funkce odpovídají asymptotickým složitostem algoritmů. Origin7 při aproximaci určoval jednotlivé koeficienty metodou nejmenších čtverců. Aproximace parabolou vycházela jen o trochu lépe, což je ale stále ještě pochopitelné, protože interval 14 bitů je relativně krátký na to, aby se výrazněji projevily rozdíly v přesnosti při aproximaci parabolou a aproximaci výše uvedenými funkcemi. Samozřejmě bylo průběžně kontrolováno, že jde o jediný aktivní proces na počítači, čímž se eliminovala možnost zkreslení výsledků způsobená případnými dalšími aktivitami procesoru.

Ze tří grafů na obrázku 2.2 popisujících závislost času rozkládání na binární délce rozkládaného „zbytku“ je možné odhadnout hranici, která určuje, zda-li je výhodnější používat Pollard–ρ nebo CFRAC. Silné body v grafu odpovídají průměrům naměřených hodnot pro danou binární délku rozkladaného čísla.

Z prvního grafu na obrázku 2.2 vyplývá, že CFRAC se nevyplácí používat, hledáme-li rozklad obsahující relativně malá prvočísla (tj. řádově 10^5). Druhý graf naopak ukazuje, že od 44 bitů délky rozkládaných čísel je v průměru CFRAC rychlejší než Pollard–ρ. Tady se už očekává, že rozklad bude obsahovat čísla řádu $2^{22} (\approx 10^6)$ a vyšší. Třetí graf ukazuje, že rozdíly mezi oběma algoritmy se citelně objevují na vstupech o délce přesahující 50 bitů (víc než 25% rozdíl v časech).

Vyvstává zde otázka, zda je měření při tak rozdílných velikostech FB účelné. V praxi totiž velikost FB zásadně ovlivňuje rychlost celého algoritmu QS a optimální hranice $\mathrm{mez}_{\mathrm{FB}}$ je pro 65 ciferné číslo zhruba sto tisíc, zatímco pro stociferné číslo je kolem jeden a půl milionu. Nezávisle na tom, jak je rozkládané číslo N velké, do dílčích rozkladů vstupují pouze „zbytky“, a tedy hraniční délka „zbytků“ rovná 44 bitů musí být stejná pro všechna rozkládaná čísla N. Faktory ovlivňující rychlost dílčích rozkladů jsou velikost FB (čím vyšší je $\mathrm{mez}_{\mathrm{FB}}$, tím větší je FB) a velikost „zbytků“. Nikoliv tedy velikost rozkládaního čísla N, jak by se na první pohled mohlo zdát.

Při podrobnějším pohledu na grafy zjišťujeme, že pro jednu délku a různé velikosti báze se časy algoritmů liší. Proč? Máme-li větší bázi, hledáme při rozkládání vyšší prvočísla. Zatímco s bází ohraničenou mezí 131 000 a délkou rozkládaného čísla 44 bitů je možný i rozklad obsahující jedno prvočíslo menší než 200 000 a druhé zákonitě mnohem vyšší ($\approx$ 90 000 000), s bází 1 048 000 už hledáme dvě prvočísla, která jsou si relativně bližší ($\approx$ např. 1 a 16 milionů).



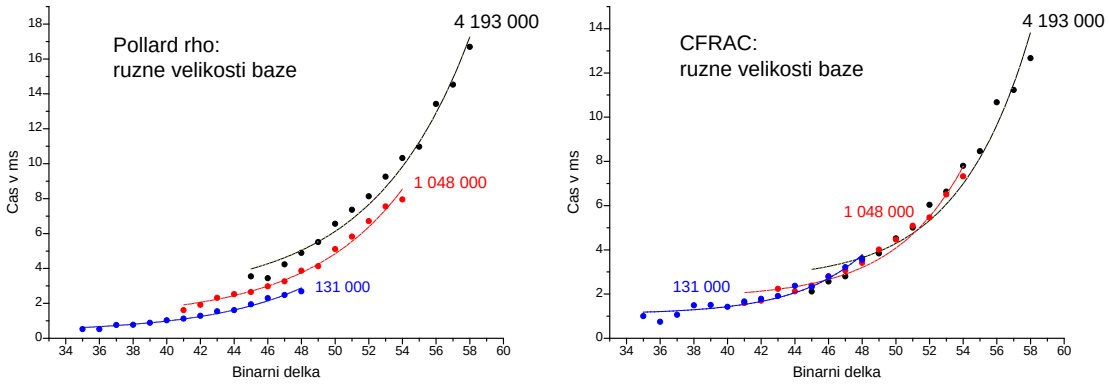
pt

Obrázek 2.2: porovnání metod Pollard–ρ a CFRAC

Podívejme se, jak toto ovlivňuje oba algoritmy.

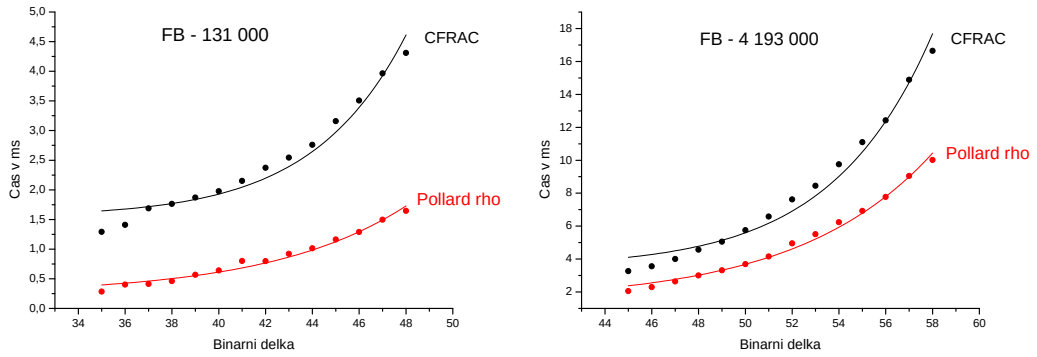
Na obrázku 2.3 vlevo vidíme, že velikost mez_{FB} (=spodní odhad na velikosti obou součinitelů) nepopiratelně ovlivňuje rychlost Pollardova–ρ algoritmu. To je způsobeno tím, že jeho složitost roste s odmocninou z nejmenšího dělitele rozkládaného čísla. Naopak u CFRACu nic takového nepozorujeme. Tedy rychlost CFRACu není ovlivněna velikostmi jednotlivých dělitelů rozkládaného čísla, ale pouze jeho binární délkou.

Dva grafy na obrázku 2.4 znázorňují výsledky měření na clusteru v budově MFF v Karlíně. Konfigurace tamních počítačů je: 64 bitový procesor AMD Opteron 246 (2.0 GHz, 1024 KB cache), paměť 2048 MB RAM, operační systém GNU/Linux. Ukázalo se, že dvakrát rychlejší procesor zvládá Pollard–ρ až dvakrát rychleji, jak se dalo očekávat. Problém nastal u CFRACu, kde nody clusteru vykazovaly do-



Obrázek 2.3: metody Pollard–ρ a CFRAC pro různé velikosti bází

konce pomalejší časy (o 20%-30%) než počítače na Malé Straně. Paradoxní situace, že na rychlejším počítači běží algoritmus pomaleji než na počítači dvakrát pomalejším, může být způsobena implementační chybou algoritmu CFRAC, rozdílem verzí operačního systému a překladače na nodech clusteru v Karlíně a na počítačích v laboratoři na Malé Straně nebo také špatnou kompatibilitou knihovny GMP pro počítání s dlouhými čísly a operačního systému na clusteru. Jiný problém se objevil při spouštění QS pod systémem Windows 2000, kdy algoritmus CFRAC v jednu chvíli „odletí“. GMP bylo původně vytvořeno pro Linux, takže problém pod Windows může pramenit odtud. Bylo by vhodné toto v budoucnu podrobněji prozkoumat. V tomto textu se tím dále zabývat nebudeme.



Obrázek 2.4: metody Pollard–ρ a CFRAC pro různé velikosti bází na clusteru

2.2 Výhody a nevýhody p-1 algoritmu

V této podkapitole použijeme stejné značení jako v podkapitole 1.1.

V testovaném programu, dottupném zde [3], je Pollardova $p-1$ metoda implementovaná jako dvoufázová funkce. V první fázi se vypočítá $y := a^{\mathrm{LCM}\,[1\ldots\mathrm{B}]} \bmod N$ a zjistí se, zda $\mathrm{GCD}(y-1, N)$ je zároveň různé od 1 a od N. Pokud ano, máme dělitel N, pokud ne, začne druhá fáze, využívající rozšířenou verzi $p-1$ algoritmu popsanou v podkapitole 1.1. Časová náročnost první fáze je ovlivněna (téměř) jenom velikostí hranice B. Velikost „zbytku“ sice také ovlivňuje rychlost první fáze, ale jenom minimálně. Prakticky můžeme tvrdit, že „zbytky“ z rozmezí 14 bitů délky jsou rozkládány $p-1$ algoritmem za stejný čas. Svým způsobem vyjímečná situace nastává v případě, kdy rozkládaný „zbytek“ je tvaru $p \cdot q$ a obě $p-1$ i $q-1$ jsou *B–mocná*. Potom $y-1$ je násobkem p a q zároveň, což znamená, že je násobkem N a tedy $y-1=0$. Algoritmus v tomto případě končí, nemá šanci uspět při procházení druhou fázi.

Časová náročnost algoritmu ve druhé fázi závisí na několika faktorech. Velikost hranice B_1 určuje počet prvočísel mezi B a B_1 a tím i množství operací během (neúspěšného) hledání prvočísla g. Proč při neúspěšném? Pokud se prvočíslo g nalezne, druhá fáze skončí. Prvočísla mezi g a B_1 se už do algoritmu nezapojují, a tedy hranice B_1 už dál neovlivní tento jeden běh algoritmu. Naopak, pokud algoritmus projde všechna prvočísla mezi B a B_1 a nenalezne g, potom ohlásí selhání. Hranice B_1 proto ovlivňuje čas množstvím prvočísel, která je nutno projít před oznámením o selhání. V původní verzi se základ pro mocnění a volil několikrát (konkrétně 16 krát), aby algoritmus uspěl i v případech, kdy je $\mathrm{GCD}(y-1, N) = N$. To je situace, kdy $y-1$ vyjde rovno $q-$násobku prvočísla p. Potom, po změně a, je velice pravděpodobné, že nově vzniklé $y-1$ bude $k-$násobkem ($k \neq q$) prvočísla p a testem na největšího společného dělitele $y-1$ a N nalezneme dělitele p. Kolikrát však změníme hodnotu a, tolikrát se prodlouží čas běhu algoritmu. Poslední zajímavý případ je také spojený se změnou a. Často se stává, že $p-1$ není *B–mocné* a tedy $\mathrm{LCM}\,[1\ldots\mathrm{B}]$ není dělitelné $p-1$. Může se však stát, že řád a (myšleno v $\mathbb{Z}_p$) dělí $\mathrm{LCM}\,[1\ldots\mathrm{B}]$. Pak by platilo: $y := a^{\mathrm{LCM}\,[1\ldots\mathrm{B}]} \equiv 1 \pmod{p}$ a dělitele N opět najdeme testem na $\mathrm{GCD}(y-1, N)$. Tento postup je ovšem časově náročný, protože nelze předem určit, pro které a bude řád a dělit $\mathrm{LCM}\,[1\ldots\mathrm{B}]$.

Následující tabulka obsahuje procentuální úspěšnosti rozkládání „zbytků“ $p-1$ metody pro různé hodnoty hranic B a B_1. Hranice B_1 je uvedena formou $k-$násobku hodnoty B. Pro $k=1$ to znamená, že nebyla použita druhá fáze algoritmu. Měření probíhalo při rozkládání 2000 čísel, zvolena $\mathrm{mez_{FB}} = 131\ 000$. Poslední sloupec tabulky vyjadřuje pravděpodobnost, že oba dělitelé „zbytku“ p a q mají tu nepříznivou vlastnost, že $p-1$ a $q-1$ jsou *B–mocné* zároveň. Tato pravděpodobnost, označme ji $\mathrm{P_{oba\ faktory\ }}_{B\text{–}mocné}$, závisí pouze na velikosti hranice B.

$B\backslash k$	1	2	3	4	5	6	7	$\mathrm{P}_{\text{oba faktory } B\text{–}mocné}$
1000	43,5	53,5	59,5	64	67	69	70,5	8
2000	48	58,5	63	66	68,5	71	71,5	14
3000	50	59	63,5	66,5	68	69,5	70,5	18,5
4000	51	59	63,5	65,5	67,5	69	70	21,5
5000	52	59,5	63	65,5	67	68	68,5	24
6000	52,5	59,5	62,5	65	66,5	67	67,5	25,5
7000	51	58	61,5	63,5	64,5	65	66	28
8000	51	57,5	61	62,5	63,5	64	65	30

Obrázek 2.5: tabulka úspěšnosti rozkladu $p-1$ algoritmu pro $\mathrm{mez}_{\mathrm{FB}} = 131\ 000$

Na první pohled by měla úspěšnost při rozkladu růst se zvyšující se hranicí B, protože čím vyšší je tato hranice, tím vyšší je šance, že rozkládané číslo má dělitele p s vlastností, že $p-1$ je *B–mocné*. Toto však v tabulce 2.5 nepozorujeme. Naopak, procentuální úspěšnost s rostoucí hranicí B nejdříve roste a posléze klesá. Jak je to možné? Odpověď nalezneme mezi výše popsanými situacemi, které mohou pro $p-1$ nastat. Konkrétně je to situace, kdy oba dělitelé „zbytku“ p i q mají vlastnost, že $p-1$ a $q-1$ jsou *B–mocná* (v algoritmu vyjde $\mathrm{GCD}(y-1, N) = N$). Porovnáním prvního a posledního řádku tabulky zjišťujeme, že zatímco pro $B = 1000$ se zvyšováním hranice B_1 až na sedmínásobek hodnoty B úspěšnost algoritmu zvýší o 27%, pro $B = 8000$ se takto zvýší jenom o 14%. Více než 21% čísel, která by se rozložila s nastavením algoritmu $B = 1000,\ B_1 = 7000$, se nerozloží s nastavením $B = B_1 = 8000$, protože pro jejich dělitele platí výše popsaná nepříznivá situce.

Uveďme pro úplnost ještě dvě tabulky odpovídající faktorizačním bázím s mezemi $\mathrm{mez}_{\mathrm{FB}} = 1\ 048\ 000$ (na obrázku 2.6) a $\mathrm{mez}_{\mathrm{FB}} = 4\ 193\ 000$ (na obrázku 2.7). Čím větší je faktorizační báze, tím delší „zbytky“ síto generuje k částečným rozkladům $p-1$ algoritmu. Proto s rostoucí $\mathrm{mez}_{\mathrm{FB}}$ je méně pravděpodobné, že vygenerovaný „zbytek“ má oba dělitele p i q s vlastností, že $p-1$ a $q-1$ jsou *B–mocné*. Tuto úvahu potvrzují i údaje v posledních sloupcích všech tří tabulek. Důsledkem klesající $\mathrm{P}_{\text{oba faktory } B\text{–}mocné}$ jsou vyšší rozdíly v úpěšnosti rozkladu $p-1$ algoritmem pozorovatelné v rámci jednotlivých sloupců.

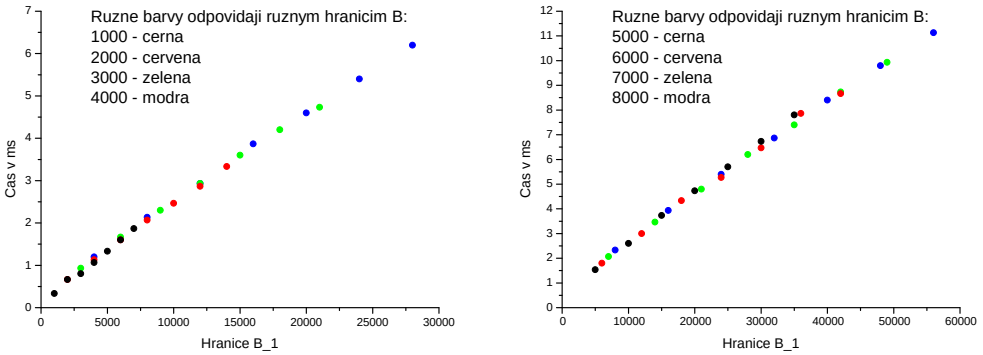
$B\backslash k$	1	2	3	4	5	6	7	$\mathrm{P}_{\text{oba faktory } B\text{–}mocné}$
1000	29,5	40,5	46	50,5	53,5	55,5	58	3
2000	38,5	49	54,5	58,5	61,5	63,5	65,5	6
3000	42,5	52,5	58	61,5	64,5	66,5	68,5	8
4000	45	55	60	63,5	66	68,5	70	10
5000	47	56,5	61	65	67,5	69	70,5	12
6000	48	57	62	65,5	67,5	69	70	13,5
7000	49	58	62,5	66	67,5	68,5	70	15
8000	49,5	58,5	63,5	66	67,5	69	70	16

Obrázek 2.6: tabulka úspěšnosti rozkladu $p-1$ algoritmu pro $\mathrm{mez}_{\mathrm{FB}} = 1\ 048\ 000$

$B\backslash k$	1	2	3	4	5	6	7	$\mathrm{P}_{\text{oba faktory } B\text{–}mocné}$
1000	24,5	34	39,5	42,5	45	47,5	50	1,5
2000	33	43	48	51,5	54	56,5	58,5	3
3000	37,5	47	52	55,5	58,5	61,5	63,5	5
4000	40	49	54,5	58,5	62	64	65	6
5000	42	51,5	57	61	63,5	65	66,5	8
6000	43,5	52,5	59	62	64	65,5	67	9
7000	45	54	59,5	62	64,5	66	67,5	10
8000	45	55	60	63	64,5	66	67,5	11

Obrázek 2.7: tabulka úspěšnosti rozkladu $p-1$ algoritmu pro $\mathrm{mez}_{\mathrm{FB}} = 4\ 193\ 000$

Grafy na obrázku 2.8 barevně znázorňují růst časové náročnosti $p-1$ algoritmu v závislosti na zvolených hranicích B a B_1 během neúspěšného běhu oběmi fázemi algoritmu. Nejde tedy o případ, kdy $p-1$ i $q-1$ jsou *B–mocná* zároveň. Tato nepříznivá situace je stejně časově náročná jako případ, kdy se algoritmu podaří číslo rozložit už po první fázi, protože do druhé fáze algoritmus vstupuje pouze v případě, že $\mathrm{GCD}(y-1, N) = 1$. Barevné body v grafech odpovídají průměru naměřených hodnot pro dané dvojice hranic (B, B_1). Z obrázků je patrné, že složitost roste lineárně v závislosti na hranici B_1, zhruba 0,25ms na každých 1000 jednotek hranice B_1. Měření bylo provedeno na počítačích v laboratoři budovy MFF na Malé Straně.



Obrázek 2.8: Časy neúspěšného běhu $p-1$ metody pro různé hranice B a B_1

Chceme-li najít optimální použití $p-1$ algoritmu při rozkládání „zbytků“ generovaných algoritmem QS, musíme uvážit úspěšnost rozkládání v závislosti na hodnotě $\mathrm{mez}_{\mathrm{FB}}$, tj. kterou ze tří uvedených tabulek budeme uvažovat pro výběr dvojice hranic (B, B_1), pak samotný výběr této dvojice, a nakonec pravděpodobnost neúspěchu algoritmu a jeho průměrný dopad na celkovou časovou náročnost

algoritmu. V případě neúspěchu je pak možné spustit jiný faktorizační algoritmus, který uspěje vždy. V rámci QS dostupného zde [3] se jedná o algoritmy Pollard–ρ a CFRAC. Je třeba rozhodnout, zda-li se vyplatí kombinovat $p-1$ metodu s jinými metodami a docílit tak toho, že všechny „zbytky“ budou rozloženy, nebo je rychlejší používat jenom $p-1$ metodu a prosíváním generovat více „zbytků“, čímž se vykompenzuje nižší úspěšnost $p-1$ metody při rozkládání. Známe-li princip prosívání v QS, je nám hned jasné, že „plýtvání se zbytky“ si můžeme dovolit jenom při rozkládání relativně menších čísel (např. 65 cifer), ale pro čísla řádu 10^{100} musíme každý „zbytek“ rozložit, protože jejich generování je časově velmi náročné. Je to stěžejní práce celého QS, která zásadně ovlivňuje časovou náročnost algoritmu.

Tabulka na obrázku 2.9 obsahuje průměrné časy potřebné pro vygenerování jednoho „zbytku“. Tento čas je ovlivněn velikosti rozkládaného čísla N (čím vyšší N, tím náročnější je prosívání), dále velikostí faktorizační báze a velikostí prosívacího intervalu. Optimálním nastavením těchto dvou faktorů v závislosti na velikosti N se, mimo jiné, ve své bakalářské práci [5] zabývá Lukáš Perůtka. Jeho výsledky pro čísla v rozmezí 50 až 70 decimálních cifer ukazují, že velikost prosívacího intervalu není natolik důležitá pro rychlost programu jako velikost faktorizační báze. Také se ukázalo, že používat double large prime variation (DLPV) je výhodné až pro čísla přesahující 70 cifer. Data v tabulce jsou proto uváděna jenom pro čísla delší než 70 cifer, hodnoty uváděné v milisekundách odpovídají nejrychlejšímu měření se zvolenými prosívacími intervaly velikosti 100 000, 200 000, 400 000 a 800 000 (rozdíly v časech se pohybovaly od nuly do deseti procent). Měření probíhalo na nodech clusteru v budově MFF v Karlíně.

délka N \ $\mathrm{mez}_{\mathrm{FB}}$	65k	131k	262k	524k	1048k	2097k	4193k
71	4,5	3	4	6,5	12	23	73
79	49	16	10	9,5	14	25,5	78
86	99	113	41	21,5	19	27	79
91	-	-	129	54	31	34	88
94	-	-	620	183	80	51	97
101	-	-	-	867	297	127	136

Obrázek 2.9: tabulka průměrné časové náročnosti (v milisekundách) pro vygenerování jednoho „zbytku“ v závislosti na velikosti rozkládaného čísla N a velikosti faktorizační báze

Perůtka uvádí, že pro 70 ciferná čísla je optimální hranice faktorizační báze řádově 60 000. To však není ve sporu s hodnotami v předchozí tabulce, podle nichž nejrychlejší generování „zbytků“ probíhá při hranici faktorizační báze zvolené řádově 130 000. Důvodem je fakt, že generování vztahů tvaru (1.3) není jenom

záležitostí DLPV, ale podílí se na tom také single large prime variation (popis je dostupný např. zde [5]) a generování „hladkých relací“ , tj. situace, kdy rozkládané číslo je mez_{FB}–*hladké*. Obecně platí, že optimální velikost faktorizační báze při DLPV se s rostoucí velikostí rozkládaného čísla N blíží té velikosti, pro níž je nejrychlejší proces generování „zbytků“. To proto, že pomocí DLPV se získává čím dál vyšší procento potřebných vztahů typu (1.3).

Nechť T značí průměrný čas potřebný ke vygenerování jednoho „zbytku“ a jeho rozložení. K výpočtu T použijeme následující intuitivní vzorec:

$$T = \frac{1}{\text{pravděpodobnost úspěšného rozkladu}} \cdot \begin{pmatrix} \text{průměrný čas} \\ \text{generování „zbytku“} \end{pmatrix} + \begin{matrix} \text{průměrný čas} \\ \text{rozkladu} \end{matrix}$$

Porovnáním tabulky 2.9 s grafy na obrázcích 2.2 a 2.4 zjistíme, že čas T je ovlivněn zejména časovou náročností prosívání. Samotné dílčí rozklady v porovnání s prosíváním ovlivňují čas T jen minimálně. Už pro 70 ciferná čísla se nevyplácí používat pro rozklady jenom $p-1$ algoritmus a muset tak generovat více „zbytků“. Ukazuje se vhodnější nejdříve použít $p-1$ algoritmus a pokud neuspěje, použít dodatečně ještě ρ–algoritmus nebo CFRAC.

Například při rozkládání 71 ciferného čísla N se zvolenou faktorizační bází s hranicí $\text{mez}_{\text{FB}} = 131\ 000$, s použitím jenom $p-1$ metody s parametry $B = 1000$ a $B_1 = 5000$, bude hodnota T rovna:

$$T = \frac{1}{0,67} \cdot (3 + 0,35) = 5\,\text{ms},$$

zatímco při kombinování s ρ–algoritmem čas T bude roven:

$$T = \frac{1}{1} \cdot (3 + (0,35 + 0,33 \cdot 1,5)) = 3,8\,\text{ms}.$$

Pozn: hodnota 0,35ms je průměrný čas úspěšného rozkládání $p-1$ algoritmem, který vypočteme z tabulky 2.5, hodnota 0,67 je pravděpobnost úspěchu $p-1$ algoritmu pro dané parametry B a B_1, průměrný čas generování jednoho „zbytku“ je 3ms a průměrný čas běhu ρ–algoritmu při $\text{mez}_{\text{FB}} = 131\ 000$ je 1,5ms, což můžeme odhadnout z grafu 2.4. Pro úplnost dodejme, že tento příklad je vztahován k faktorizaci na nodech clusteru v Karlíně.

Hodnota T slouží pouze pro orientační účely při výběru hranic B a B_1 algoritmu $p-1$ a při rozhodování, zda použít pouze $p-1$ algoritmus nebo jej kombinovat s jiným. Hodnota T se nedá přesně měřit, protože rychlost celého QS nezávisí jenom na rychlosti DLPV. Také pokud známe celkový počet úspěšně rozložených zbytků, neznamená to, že je roven počtu provedených rozkladů, protože některé rozklady sice proběhly úspěšně, ale získaný dělitel měl délku více než 32 bitů. To často nastává pro větší FB ($\text{mez}_{\text{FB}} \approx 10^6$).

2.3 Optimální využití algoritmů v rámci QS

V této kapitole předpokládáme u čtenáře znalost algoritmů $p-1$, ρ a CFRAC, stejně tak jako jejich úlohu v rámci algoritmu QS. Budeme také používat některá označení zavedená v předchozích dvou kapitolách.

Cílem celé práce bylo zkoumat rychlosti algoritmů $p-1$, ρ a CFRAC implementovaných v QS (dostupné zde [3]) při rozkládání složených čísel, která QS generuje v rámci varianty DLPV, a navrhnout rozhodovací proceduru, která určí, jaký algoritmus se použije pro rozklad „zbytků“. Toto rozhodování by mělo být prováděno na základě typu počítače, rychlostech jednotlivých algoritmů na tomto počítači, délce rozkládaného čísla N a velikosti FB.

Pro čísla řádu 10^{70} a méně se nevyplatí používat DLPV, jak uvádí L. Perůtka v textu [5]. Proto pro taková čísla žádnou optimalizaci DLPV neprovádíme. Výsledkem této práce je poznatek, že optimálním způsobem rozkládání „zbytků“ při použití DLPV (uvažujme použití DLPV pro čísla N delší než sedmdesát cifer) je kombinovat použití $p-1$ algoritmu s jiným faktorizačním algoritmem, který při rozkládání uspěje stoprocentně. Při použití DLPV se totiž rozkládají tak vysoká čísla, že samotné generování „zbytků“ prosíváním algoritmem QS zabírá nejvíce času. Proto je třeba každý „zbytek“ rozložit a ne s nimi „plýtvat“, čemuž bychom se nevyhnuli při použití jenom $p-1$ algoritmu, který je často v rozkládání neúspěšný.

Jednoduchými kroky jsme z tabulek 2.5, 2.6 a 2.7 vypočetli optimální volby hranic B a B_1 algoritmu $p-1$:

- pro FB odpovídající $\text{mez}_{\text{FB}} \leq 131\ 000$ je nejlepší volbou $B = 1000$ a $B_1 = 5000$
- pro FB odpovídající $\text{mez}_{\text{FB}} \geq 4\ 193\ 000$ je nejlepší volbou $B = 3000$ a $B_1 = 21000$
- pro ostatní velikosti FB je nejlepší volbou $B = 1000$ a $B_1 = 7000$

Jde však o tak malé rozdíly v časech, že v porovnání s náročností generování „zbytků“ (viz tabulka 2.9) hraje (i špatná) volba hranic B a B_1 zanedbatelnou roli.

Optimalizovaný algoritmus používá pro všechny rozklady „zbytků“ nejprve $p-1$ algoritmus a pokud tento při rozkladu selže, začneme rozkládat algoritmem ρ a měříme jeho rychlost v závislosti na binární délce vstupu (měříme 5000×) a totéž pak proběhně i pro algoritmus CFRAC. Pro všechny další rozklady, při nichž $p-1$ algoritmus neuspěl, se volí faktorizační algoritmus (ρ nebo CFRAC) podle toho, zda byl v průměru rychlejší pro danou binární délku aktuálně rozkládaného „zbytku“.

Přechod mezi volbami algoritmů se ukazuje být u binární délky vstupů rovné 44. Pro „zbytky“ delší už je výhodnější používat CFRAC. To platí pro velikost FB odpovídající $\text{mez}_{\text{FB}} \approx 524\,000$ a vyšší.

Problém s rychlostí CFRACu, který se objevil na nodech clusteru v Karlíně, bude třeba do budoucna vyřešit. Ať už revizí zdrojového kódu CFRACu kvůli implementačním chybám, nebo také měřením rychlostí jednotlivých částí algoritmu CFRAC kvůli jejich časové náročnosti a porovnáním s očekávanými hodnotami.

Při porovnání, o kolik je rychlejší QS s výše popsanou optimalizací a QS dostupné zde [3], jsme rozkládali čísla se 71, 79 a 86 decimálními ciframi a byly zvoleny báze postupně 131 000, 262 000 a 524 000. Optimalizovaná verze byla vždy rychlejší. Měření probíhala na počítačích v laboratoři na Malé Straně.

- V případě 71 ciferného čísla bylo potřeba nalézt asi 130 tisíc rozkladů a zrychlení rozkládání by mělo podle intuitivního vzorce pro hodnotu T být asi o jednu milisekundu na každý rozklad. To odpovídá celkovému zrychlení o dvě minuty. Měření ukázalo zrychlení o jeden a půl minuty ($\approx 7{,}5\%$). Bohužel i tak to trvalo asi dvakrát déle než kdyby se použila pouze single large prime variation a báze 320 000.

- V případě 79 ciferného čísla bylo potřeba nalézt asi 930 tisíc rozkladů a zrychlení rozkládání by mělo podle intuitivního vzorce pro hodnotu T být asi o půl milisekundy na každý rozklad. To odpovídá celkovému zrychlení o osm minut. Měření ukázalo zrychlení o devět minut ($\approx 9\%$).

- V případě 86 ciferného čísla bylo potřeba nalézt asi 1,7 milionu rozkladů a zrychlení rozkládání by mělo podle intuitivního vzorce pro hodnotu T být asi o dvě milisekundy na každý rozklad. To odpovídá celkovému zrychlení o 57 minut. Měření ukázalo zrychlení o 41 minut ($\approx 6\%$).

Naměřená zrychlení QS v řádu 5-10% v důsledku optimalizace volby faktorizačních algoritmů pro dílčí rozklady jsou hlavním přínosem celé této práce. Dodejme však, že důležitějším parametrem z hlediska rychlosti QS je velikost faktorizační báze. Dalšími důležitými parametry jsou velikost prosívacího intervalu a hodnota parametru checkX. Zvolené hodnoty pro mez_{FB} užívané k měření (131 000 až 4 193 000) slouží pouze k ilustraci závislostí některých jevů v QS a upřesnění volby mez_{FB} pro rozkládání čísel delších než 70 cifer by mohlo zrychlit celý algoritmus v řádu i desítek procent.

Literatura

[1] Cohen, H.: *A Course in Computational Algebraic Number Theory,* Springer-Verlag, Berlin, (1993)

[2] Drápal, A.: texty ze semináře o kryptologii, http://adela.karlin.mff.cuni.cz/∼krypto/seminar.php

[3] Drápal, A.: webové stránky katedry algebry MFF, sekce kryptologie, http://adela.karlin.mff.cuni.cz/∼krypto/mpqs.php

[4] archiv webových stránek o matematice, http://mathworld.wolfram.com/FermatsFactorizationMethod.html

[5] Perůtka, L.: *Měření na kvadratickém sítu,* bakalářská práce, Univerzita Karlova, Praha, (2006)

[6] Zvánovec, J.: *Testy prvočíselnosti a rozklady celých čísel,* diplomová práce, Univerzita Karlova, Praha, (2006)