Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Bc. Vojtěch Matocha

Pokročilé metody hledání diskrétního logaritmu

Katedra algebry

Vedoucí diplomové práce:	Mgr. Pavel Příhoda Ph. D.
Studijní program:	Matematika
Studijní obor:	Matematické metody informační bezpečnosti

Praha 2013

Na tomto místě bych chtěl poděkovat všem, kteří přispěli ke vzniku této práce, zejména potom svému vedoucímu doktoru Pavlu Příhodovi za cenné komentáře, kritické poznámky a všechen čas, který se strávil četnými konzultacemi a pročítáním tohoto textu.

V neposlední řadě bych chtěl poděkovat docentu Štěpánu Holubovi za rady a myšlenky týkající se problematiky rozšíření konečných těles, doktoru Petru Sombergovi za ochotu při zodpovídání mých četných dotazů ohledně algebraických křivek a docentu Stanovskému za drobné konzultace týkající se implementačních problémů v konečných tělesech.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Pokročilé metody hledání diskrétního logaritmu

Autor: Bc. Vojtěch Matocha

Katedra: Katedra algebry

Vedoucí diplomové práce: Mgr. Pavel Příhoda Ph. D.

Abstrakt: Mějme konečnou cyklickou grupu $G = \langle g \rangle$. Problém diskrétního logaritmu, tedy pro zadané $y \in G$ nalézt $x \in \mathbb{N}$ splňující $g^x = y$, představuje jeden ze základních pilířů moderních kryptografických transformací. Ve své práci podáváme přehled algoritmů, které se pro výpočet diskrétního logaritmu používají, včetně v současnosti nejrychlejšího známého algoritmu pro grupu $\mathbb{F}_{p^n}^*$: funkčního síta. Kromě funkčního síta se podrobněji zabýváme index kalkulem a jeho optimalizacemi: Coppersmithovým algoritmem a polynomiálním sítem. Hlavním přínosem práce je implementace funkčního síta v jazyce C a její aplikace na konkrétní vstupy.

Klíčová slova: diskrétní logaritmus, algoritmus, funkční síto

Title: Advanced techniques for calculations of discrete logarithm

Author: Bc. Vojtěch Matocha

Department: Department of Algebra

Supervisor: Mgr. Pavel Příhoda Ph. D.

Abstract: Let $G = \langle g \rangle$ be a finite cyclic group. Solving the equation $g^x = y$ for given $y \in G$ is called the discrete logarithm problem. This problem is at the core of many modern cryptographic transformations. In this paper we provide a survey of algorithms to attack this problem, including the function field sieve, the fastest known algorithm applicable to the group $\mathbb{F}_{p^n}^*$. We also discuss the index calculus algorithm and some techniques improving its performance: the Coppersmith's algorithm and the polynomial sieving. The most important contribution of this paper is a C-language implementation of the function field sieve and its application to real inputs.

Keywords: discrete logarithm, algorithm, function field sieve

Obsah

1 Diskrétní logaritmus **3**
1.1 Definice problému . . . 3
1.2 Triviální algoritmus výpočtu . . . 4
1.3 Význam v kryptografii . . . 4
1.3.1 RSA . . . 4
1.3.2 ElGamal . . . 5
1.3.3 Diffie-Hellman . . . 5
1.3.4 Používané cyklické grupy . . . 6

2 Implementace algoritmů **7**
2.1 Jazyk Java . . . 7
2.2 Jazyk C . . . 7
2.3 Použité knihovny . . . 7
2.4 Dokumentace . . . 8

3 Souvislost s číselnou faktorizací **9**
3.1 Úvod . . . 9
3.2 Redukce na diskrétní logaritmus v podgrupách . . . 9
3.3 Faktorizace za pomoci výpočtu zobecněného diskrétního logaritmu 10
3.4 Shrnutí . . . 12
3.5 Implementace . . . 12

4 Algoritmus Baby steps - giant steps **13**
4.1 Princip algoritmu . . . 13
4.2 Časová složitost . . . 13
4.3 Algoritmus v pseudokódu . . . 14
4.4 Optimalizace . . . 15
4.5 Implementace . . . 16

5 Pollardův ρ algoritmus **17**
5.1 Narozeninový paradox . . . 17
5.2 Princip algoritmu . . . 17
5.3 Floydův algoritmus pro hledání cyklů . . . 19
5.4 Časová složitost . . . 19
5.5 Algoritmus v pseudokódu . . . 20
5.6 Optimalizace . . . 20
5.7 Implementace . . . 20

6 Index kalkulus **22**
6.1 Princip algoritmu . . . 22
6.2 Pravděpodobnost volby b-hladkého polynomu . . . 24
6.3 Časová složitost . . . 26
6.4 Coppersmithův algoritmus pro $\mathrm{GF}(2^n)$ . . . 27
6.5 Semaevův algoritmus . . . 29
6.6 Polynomiální síta . . . 31
6.7 Další zkoumané metody . . . 34

6.8 Implementace 35

7 Funkční síto **37**
7.1 Teoretický úvod 37
7.2 Popis algoritmu 43
7.2.1 Dvojnásobně hladké páry 43
7.2.2 Výpočet normy $ry + s$ 44
7.2.3 Konstrukce polynomu H 45
7.2.4 Náhražková funkce 47
7.2.5 Morfismus ψ na hodnotách náhražkové funkce α 47
7.2.6 Částečný diskrétní logaritmus 49
7.2.7 Hladké kongruence 49
7.2.8 Diskrétní logaritmus 51
7.2.9 Optimalizace - prosívání 52
7.3 Přehled použitých struktur 53
7.4 Výpočet diskrétních valuací 54
7.5 Newtonova mnohoúhelníková metoda 57
7.5.1 Implementace 59
7.6 Časová složitost 60
7.7 Paralelizace 62
7.8 Algoritmus v pseudokódu 62

8 Srovnání algoritmů **65**
8.1 Baby steps - giant steps 65
8.2 Pollard ρ 67
8.3 Baby steps - giant steps vs. Pollard ρ 68
8.4 Index kalkulus 69
8.5 Funkční síto 74

9 Závěr **77**
9.1 Shrnutí 77
9.2 Otevřené otázky 77
9.3 Přínosy práce 77

Seznam použité literatury **79**

1. Diskrétní logaritmus

1.1 Definice problému

Nechť G je cyklická grupa řádu $n \in \mathbb{N}$, nechť g je její generátor. Problémem diskrétního logaritmu prvku $t \in G$ pak rozumíme nalezení nejmenšího přirozeného čísla l takového, že $g^l = t$. Takové l z definice generátoru grupy jistě existuje v množině $0, \ldots, n-1$ a budeme ho značit $\log_g t$ nebo pro jednoduchost jen $\log t$, je-li generátor grupy zřejmý z kontextu.

Kromě standardního diskrétního logaritmu se na několika místech v práci setkáme se zobecněným diskrétním logaritmem. Běžně je n známá hodnota. Zobecněným diskrétním logaritmem budeme rozumět problém řešení rovnice $g^l = t$ v grupě, jejíž řád neznáme na vstupu.

Souvislost s běžným logaritmem reálných čísel je patrná na první pohled, mimo jiné platí vztahy obvyklé pro běžný logaritmus:

- $\log_g 1 = 0$,
- $\log_g g = 1$,
- $\log_g a^k = k \cdot \log_g a$ pro všechna $a \in G$, $k \in \mathbb{N}$,
- $\log_g(a \cdot b) = \log_g a + \log_g b$ pro všechna $a, b \in G$, protože

  $$\log_g(a \cdot b) = \log_g(g^k \cdot g^l) = \log_g g^{k+l} = k + l = \log_g g^k + \log_g g^l = \log_g a + \log_g b.$$

Příklad: Mějme multiplikativní cyklickou grupu $\mathbb{Z}_{11}^*$ s generátorem $g = 6$. Diskrétním logaritmem prvku $t = 8$ je pak hodnota $l = 7$, neboť

$$g^l \pmod{11} = 6^7 \pmod{11} = 279936 \pmod{11} = 8.$$

Příklad: Mějme aditivní cyklickou grupu $\mathbb{Z}_{10}$ s generátorem $g = 7$. Diskrétním logaritmem prvku $t = 8$ je hodnota $l = 4$, neboť

$$g^l \pmod{10} = 7 \cdot 4 \pmod{10} = 28 \pmod{10} = 8.$$

Z výše uvedených příkladů je patrné, že diskrétní logaritmus nemusí být z výpočetního hlediska vždy náročný. Výpočet hodnoty lze v druhém případě, v němž je grupová operace definována běžným sčítáním modulo 10, snadno provést v polynomiálním čase pomocí Euklidova algoritmu. Naproti tomu v prvním případě, tedy v multiplikativní grupě tělesa $\mathbb{F}_{11}$, je výpočet výrazně obtížnější, pro obecný případ není algoritmus s polynomiální časovou složitostí v nejhorším případě dosud znám. Skutečná výpočetní složitost diskrétního logaritmu v cyklické grupě

řádu n se proto vždy odvíjí od konkrétní definice grupy a od operace, která je na ní zavedena.

V předchozím odstavci jsme zmínili, že pro výpočet diskrétního logaritmu je velmi důležitá samotná definice a následná reprezentace grupy. V obecném případě je totiž přirozená reprezentace daného prvku grupy právě pomocí příslušné mocniny pevně zvoleného generátoru. V takovém případě je ovšem diskrétní logaritmus triviální úloha. Ukázkovým příkladem je aditivní cyklická grupa $\mathbb{Z}_n$ s generátorem $g = 1$. Hodnota libovolného prvku je pak přímo hodnotou jeho diskrétního logaritmu.

Obecný model grupy, používaný pro obecné řešení problému diskrétního logaritmu, proto předpokládá, že prvek sám informaci o diskrétním logaritmu nenese; předpokládá, že lze pouze rozhodnout, zda jsou dva prvky grupy identické, a toto rozhodnutí je možno provést za jednu jednotku času. Dále obecný model předpokládá, že lze v grupě provést za jednu jednotku času výpočet grupové operace na libovolných dvou prvcích a nalezení prvku inverzního. Tento předpoklad nemusí být samozřejmý ve všech strukturách: například v podílových strukturách nemusí být porovnání dvou prvků snadné.

1.2 Triviální algoritmus výpočtu

Nejjednodušším postupem výpočtu diskrétního logaritmu v dané grupě je iterování grupové operace. Na generátor g provádíme grupovou operaci, dokud nedostaneme požadovaný prvek t. Tento postup je ovšem značně neefektivní, v nejhorším případě musíme provést tolik operací a porovnání, kolik je počet prvků celé grupy.

To vede na exponenciální složitost vzhledem k délce vstupu, která je $k = \log n$, tedy $O(2^k) = O(n)$. Výhodou může být zanedbatelná prostorová složitost, přesto se triviální algoritmus používá jen pro velmi malé grupy.

1.3 Význam v kryptografii

Na předpokladu, že je výpočet diskrétního logaritmu v některých grupách časově náročný, je vybudována celá řada moderních kryptografických transformací, zejména asymetrických šifer jako RSA a protokolů pro sdílení tajemství jako ElGamal nebo Diffie-Helman. Následující přehled slouží především jako ilustrace praktického využití diskrétního logaritmu, hlubší analýzu uvedených postupů lze nalézt například v [11].

1.3.1 RSA

RSA je asymetrický kryptosystém s veřejným klíčem, jedná se o vůbec první algoritmus vhodný pro šifrování i podepisování. Předpokládejme, že B chce odeslat A šifrovanou zprávu m.

1. A zvolí dvě různě velká náhodná prvočísla p a q.

2. A spočítá $n = pq$.

3. A spočítá hodnotu Eulerovy funkce $\phi(n) = (p-1)(q-1)$.

4. A zvolí $e < \phi(n)$ takové, že $\gcd(e, \phi(n)) = 1$.

5. A nalezne d takové, že $de = 1 \pmod{\phi(n)}$.

6. B odešle A šifrový text $c = m^e \pmod{n}$

7. A přijme c a spočítá

$$c^d = (m^e)^d \pmod{n} = m^{ed} \pmod{n}$$

Protože $ed = 1 \pmod{p-1}$ a $ed = 1 \pmod{q-1}$ a p a q jsou nesoudělná, platí $c^d = m \pmod{n}$.

Útočník E schopný počítat zobecněný diskrétní logaritmus v multiplikativní grupě $\mathbb{Z}_n$ může způsobem popsaným dále ze znalosti n efektivně dopočítat p a q, odtud $\phi(n)$ a nalézt libovolné d' splňující $d' \cdot e = 1 \pmod{\phi(n)}$ a následně vypočítat m analogicky jako A.

1.3.2 ElGamal

Pomocí tohoto algoritmu mohou dvě strany A a B bezpečně komunikovat přes nezabezpečený komunikační kanál. Předpokládejme, že A chce odeslat B šifrovanou zprávu, kterou lze vhodně reprezentovat prvkem m ve zvolené grupě G.

1. A a B se veřejně dohodnou na cyklické grupě G řádu n a generujícím prvku $\alpha \in G$.

2. B zvolí náhodné celé číslo $l \in \{0, 1, \ldots, n-1\}$ a vypočítá α^l. Tuto hodnotu zašle nezabezpečeně A.

3. A zvolí náhodné celé číslo $k \in \{0, 1, \ldots, n-1\}$ a vypočítá α^k.

4. A vypočítá hodnoty $(\alpha^l)^k$ a $m\alpha^{lk}$ pro danou zprávu $m \in G$ a zašle dvojici hodnot $(\alpha^k, m\alpha^{lk})$ účastníkovi B.

5. B vypočítá $(m\alpha^{lk})((\alpha^k)^l)^{-1} = m\alpha^{lk}(\alpha^{lk})^{-1} = m$.

Případný útočník E je v průběhu komunikace schopen zachytit přenášené hodnoty α, α^l, α^k a $m\alpha^{lk}$. Je-li schopen efektivně počítat diskrétní logaritmus v grupě G, snadno spočítá l a zprávu m dopočítá stejně jako účastník B.

1.3.3 Diffie-Hellman

Pomocí tohoto algoritmu mohou dvě komunikující strany A a B, používající nezabezpečený komunikační kanál, vygenerovat sdílené tajemství.

1. A a B zvolí veřejně známou grupu G a její prvek α.

2. A zvolí náhodné celé číslo a a spočítá α^a v grupě G.

3. A odešle α^a nezabezpečeným kanálem straně B.
4. B zvolí náhodné celé číslo b a spočítá α^b v grupě G.
5. A přijme α^b a spočítá $(\alpha^b)^a = \alpha^{ab}$
6. B přijme α^a a spočítá $(\alpha^a)^b = \alpha^{ab}$

A a B tak sdílí tajemství α^{ab}, které může sloužit například jako tajný klíč pro další komunikaci symetrickou blokovou šifrou. V případě, že útočník E umí efektivně počítat diskrétní logaritmus v G, může odposlechnout α^a a α^b, vypočítat a a b a z nich také α^{ab}.

1.3.4 Používané cyklické grupy

Bezpečnost výše uvedených algoritmů závisí do značné míry na zvolené cyklické grupě. Obvyklou volbou bývají cyklické grupy $\mathbb{Z}_p^*$ pro dostatečně velké prvočíslo p nebo z implementačních důvodů $\mathrm{GF}_{2^n}^*$ pro dostatečně velké n. V praxi se v závislosti na požadované bezpečnosti sdíleného klíče pro Diffie-Hellmanův algoritmus používají grupy $\mathrm{GF}_{2^{768}}^*$, $\mathrm{GF}_{2^{1024}}^*$ nebo $\mathrm{GF}_{2^{2048}}^*$.

Kromě multiplikativních grup číselných těles se v praxi používají také cyklické podgrupy grupy bodů eliptické křivky. Oproti číselným tělesům jsou považovány za bezpečnější [11], nevýhodou může být složitost výpočtu jedné grupové operace a z ní vyplývající skutečná časová složitost konkrétní implementace algoritmu. Přehled standardních používaných eliptických křivek je k vidění na [12].

2. Implementace algoritmů

Součástí této práce je implementace šesti popisovaných algoritmů: algoritmu Baby steps - giant steps, algoritmu Pollard ρ, základní verze index kalkulu, Coppersmithova algoritmu bez prosívání, Coppersmithova algoritmu s prosíváním a funkčního síta. Jedním z cílů práce je otestovat tyto algoritmy na přiměřeně velkých vstupech a ověřit jejich očekávanou složitost testováním. Při implementaci byl kladen důraz na časovou složitost asymptotickou, nikoli na časovou složitost v reálném průměrném případě. Je téměř jisté, že některé již existující implementace výše uvedených algoritmů dokáží spočítat diskrétní logaritmus daného vstupu v závislosti na použitém programovacím jazyce výrazně rychleji.

2.1 Jazyk Java

Některé algoritmy byly implementovány v jazyce Java (použitá verze 1.5). Jazyk Java byl pro naprogramování zvolen coby moderní nástroj podporující počítání s velkými čísly, polynomiální a modulární aritmetikou a bohatou knihovnou standardních matematických funkcí. Java pracuje s referencemi na objekty v paměti, umožňuje tedy udržovat spojové struktury, a nabízí sadu kolekcí pro snadnou implementaci seznamů, množin a hešovacích map. Nevýhodou Javy je i přes použití just-in-time compilation její absolutní rychlost; překladač nevytváří nativní kód, pouze bytecode, který musí být znovu interpretován pomocí virtual machine v závislosti na konkrétní používané platformě. Na časové náročnosti se to však projeví pouze zanedbatelnou multiplikativní konstantou.

2.2 Jazyk C

Pokročilé algoritmy jako index kalkulus a jeho optimalizace byly implementovány v jazyce C. Jeho výhodou je maximální možná rychlost; zdrojové kódy v jazyce C jsou kompilovány přímo do nativního kódu daného operačního systému, nemusí být již znovu interpretovány, a dosahují proto skvělých výsledků v zátěžových testech.

Jazyk C a jeho knihovny `NTL` a `GMP` podporují výpočty s velkými čísly a polynomy. Pro výpočty nad obory charakteristiky 2 jsou tyto knihovny optimalizované a plně využívají možností hardwaru. Pro závěrečnou fázi index kalkulu a funkčního síta, v níž je nutné spočítat soustavu rovnic definovanou řídkou maticí, byla použit matematický software Mathematica, který implementuje Wiedemannovu metodu. Lineární fáze těchto algoritmů tedy není výstupem práce.

2.3 Použité knihovny

Pro práci s velkými čísly a složitějšími algebraickými strukturami, jako například konečnými tělesy nebo polynomy s koeficienty v konečných tělesech, byla použita open-source knihovna projektu `JAS` - Java Algebra System. Knihovna `JAS` posky-

tuje objektově orientovaný vícevláknový přístup k počítačové algebře, instance prvků používaných struktur jsou zpravidla instancemi tříd v balíku `edu.jas.arith`.

Algoritmus funkčního síta a některé další algoritmy pro výpočet diskrétního logaritmu používají interně pokročilé algebraické a numerické podpůrné algoritmy:

Pro ověření, zda je dané přirozené číslo prvočíslem, byla proto do zdrojového kódu zahrnuta knihovna pro polynomiální deterministický algoritmus AKS. Jeho zdrojový kód se nachází v balících začínajících na `aks`. Algoritmus AKS, jak byl popsán v [8], na vstupu pracuje s bezčtvercovým přirozeným číslem. Implementace, použitá v rámci této práce, proto nejprve ověřuje bezčtvercovost, následně se pokusí prokázat složenost pomocí Rabin-Millerova testu [9], a teprve pokud se složenost po stanoveném počtu iterací neprokáže, rozhodne o prvočíselnosti pomocí AKS.

Jelikož neexistuje žádná veřejně přístupná a pro účely práce použitelná implementace metody pro výpočet diskrétních valuací založené na Newtonově mnohoúhelníku, byla implementována v rámci této práce a je jedním z jejích dílčích výstupů.

V závěru funkčního síta se používá Wiedemannův numerický algoritmus pro výpočet soustavy rovnic nad konečným tělesem zadané řídkou maticí. Vzhledem ke složitosti tohoto numerického algoritmu a ke skutečnosti, že již existuje kvalitní implementace v rámci projektu Mathematica, rozhodl jsem se existující implementace využít.

2.4 Dokumentace

Celý kód implementace se nachází v příloze. Konkrétní technické detaily jsou podrobněji vysvětleny v dokumentaci ke zdrojovému kódu. Ta byla vygenerována jako standardní javadoc a je rovněž k nahlédnutí v příloze, v programech v jazyce C jsou komentáře přímo ve zdrojových souborech.

3. Souvislost s číselnou faktorizací

3.1 Úvod

Úspěšnost hledání rychlého algoritmu pro výpočet diskrétního logaritmu se v průběhu let do značné míry odvíjela od úspěšnosti hledání algoritmu pro číselnou faktorizaci. Názorným příkladem tohoto vztahu budiž úzká souvislost mezi číselným sítem pro výpočet faktorizace a funkčním sítem pro výpočet diskrétního logaritmu, o němž bude pojednávat samostatná kapitola.

V následující kapitole proto uvedeme některá tvrzení, která dokazuje Eric Bach v [5], osvětlující přímou souvislost mezi faktorizací a zobecněným diskrétním logaritmem.

3.2 Redukce na diskrétní logaritmus v podgrupách

Mějme cyklickou grupu $G = \langle g \rangle$ řádu n a předpokládejme, že pro n platí:

$$n = uv,$$

a zároveň $gcd(u, v) = 1$. G lze pak psát jako direktní součin dvou podgrup:

$$G = G_1 \times G_2,$$

kde $|G_1| = u$ a $|G_2| = v$. Z Lagrangeovy věty navíc jistě

$$\langle g^v \rangle = G_1, \langle g^u \rangle = G_2.$$

Chceme-li vypočítat x, že $g^x = t$, má smysl uvažovat následující dílčí diskrétní logaritmy l_u a l_v v podrupách G_1 a G_2:

$$(g^u)^{l_u} = t^u$$

$$(g^v)^{l_v} = t^v.$$

Známe-li hodnoty l_u a l_v, můžeme dopočítat celkový diskrétní logaritmus x pomocí Euklidova algoritmu. Nalezneme a,b taková celá čísla, že $au + bv = 1$. Pak

$$t = t^{au+bv} = (t^u)^a (t^v)^b = g^{ul_u a} g^{vl_v b} = g^{aul_u + bvl_v},$$

a tedy

$$\log_g t = aul_u + bvl_v \pmod n$$

Pomocí výše popsané techniky jsme schopni pomocí celočíselné faktorizace redukovat problém diskrétního logaritmu na diskrétní logaritmus v grupách řádu mocnin prvočísel.

Předpokládejme nyní, že řád grupy G je p^a. Jako obvykle hledáme x takové, že $g^x = t$. x pak můžeme zapsat v bázi p takto:

$$x = b_0 + b_1 p + \cdots + b_{a-1} p^{a-1},$$

kde b_i jsou celá čísla,

$$0 \leq b_i \leq p - 1.$$

Naším cílem bude postupně vypočítat jednotlivé číslice x, tedy hodnoty b_i. Platí:

$$t^{p^{a-1}} = (g^x)^{p^{a-1}} = g^{xp^{a-1}} = g^{b_0 p^{a-1}} = (g^{p^{a-1}})^{b_0}.$$

Hodnota b_0 je tedy diskrétním logaritmem v grupě generované $g^{p^{a-1}}$, která je prvočíselného řádu p.

Předpokládejme nyní, že hodnoty $b_0, \ldots, b_{j-1}$ jsme již vypočítali. Uvažme

$$t_j = t g^{b_0 - b_1 p - \ldots - b_{j-1} p^{j-1}}.$$

Víme, že t_j je p^j-tá mocnina, takže $t_j^{p^{a-j-1}}$ je v podgrupě p^{a-1}-tých mocnin. Hodnotu b_j tak můžeme vypočítat vyřešením diskrétního logaritmu

$$t_j^{p^{a-j-1}} = (g^{p^{a-1}})^{b_j}.$$

Známe-li všechny číslice v zápisu x, hodnotu x snadno dopočítáme.

3.3 Faktorizace za pomoci výpočtu zobecněného diskrétního logaritmu

Pro přehlednost nejprve definujme pro prvočíslo p následující značení:

- $\nu_p(x) = \max\{k : p^k | x\}$
- $\nu_p(0) = \infty$

Symbolem $\nu_p(x)$ tedy budeme rozumět nejvyšší mocninu prvočísla p takovou, která ještě dělí x.

V následujících úvahách předpokládejme, že n je liché přirozené číslo dělitelné alespoň dvěma různými prvočísly. Zda n tuto vlastnost splňuje, můžeme ověřit v polynomiálním čase, ať už pravděpodobnostním Rabin-Millerovým algoritmem nebo deterministickým AKS [8]. Nechť tedy

$$n = p_1^{e_1} \ldots p_k^{e_k}.$$

Podle čínské věty o zbytcích existuje isomorfismus

$$\mathbb{Z}_n^* \cong \mathbb{Z}_{p_1^{e_1}}^* \times \cdots \times \mathbb{Z}_{p_k^{e_k}}^*.$$

Toho nyní využijeme. Nechť ϕ_l značí řád l-tého faktoru,

$$\phi_l = (p_l - 1) p_l^{e_l - 1}.$$

Odtud plyne, že řád každého prvku původní „velké“ grupy $\mathbb{Z}_n^*$ dělí

$$\lambda = \mathrm{lcm}(\phi_1, \ldots, \phi_k).$$

Předpokládejme nyní, že a je náhodně zvolené číslo ze $\mathbb{Z}_n^*$, symbolem K budeme označovat

$$K = \{a \in \mathbb{Z}_n^* : a^{\lambda/2} \equiv \pm 1 \pmod{n}\}.$$

Množina K je zřejmě vlastní podgrupou $\mathbb{Z}_n^*$. Předpokládáme-li totiž, že

$$\nu_2(p_1 - 1) \geq \nu_2(p_i - 1) \forall i = 2, \ldots, k,$$

pak můžeme volit a řádu ϕ_1 modulo $p_1^{e_1}$ a řádu $\phi_i/2$ modulo ostatní mocniny prvočísel. Takto zkonstruované a jistě neleží uvnitř podgrupy K. Protože K je vlastní podgrupa, je takových a alespoň polovina, neboť faktorgrupa $\mathbb{Z}_n^*/K$ má velikost alespoň 2. Prvek $a \notin K$ proto můžeme snadno nalézt náhodnou volbou.

Mějme takové $a \notin K$, buď x jeho exponent, tedy nějaké přirozené číslo takové, že $a^x = 1 \pmod{n}$. Předpokládejme nyní, že toto x umíme efektivně spočítat, přesný postup založený na diskrétním logaritmu osvětlíme později.

Pak pro nějaké k splňující $0 < k < \log |x|$ platí, že

$$a^{x/2^k} \not\equiv \pm 1 \pmod{n}$$

a zároveň k splňuje, že:

$$(a^{x/2^k})^2 \equiv 1 \pmod{n}.$$

Nechť je α řád prvku a v $\mathbb{Z}_n^*$. Jistě α dělí hodnotu λ, $\lambda = \alpha \cdot \beta_0$ pro nějaké β_0 liché. Podobně pro nějaké β_1 liché platí $x = \alpha \cdot \beta_1 \cdot 2^\nu$. Odtud

$$a^{\lambda/2} \equiv a^{\beta_0 \lambda/2} \equiv a^{\alpha/2} \pmod{n}.$$

Příslušné k potom snadno zvolíme jako $k = \nu + 1$. Z výše uvedeného pak vyplývá, že $\gcd(n, (a^{x/2^k} + 1) \pmod{n})$ je netriviální dělitel n.

Nyní předpokládejme, že umíme efektivně řešit zobecněný diskrétní logaritmus $a^y = b \pmod{n}$ pro pevně dané a a b v $\mathbb{Z}_n^*$, kde $\phi(n)$ neznáme. Oproti obvyklému zadání diskrétního logaritmu je tento problém obecnější, neznáme řád grupy, který můžeme standardně při výpočtu využít.

Za tohoto předpokladu ukážeme, jak nalézt exponent a. Řešit zobecněný diskrétní logaritmus $a^x = 1 \pmod{n}$ nestačí, ten je vždy 0. Nejprve nalezneme prvočíslo p, které nedělí $\phi(n)$. Takové jistě existuje někde mezi prvními $\lfloor \log n \rfloor + 1$ prvočísly. Pak existuje y, že

$$(a^p)^y = a \pmod{n}.$$

Zvolíme-li $x = py - 1$, dostáváme hledaný exponent.

Jsme-li navíc schopni vypočítat x v polynomiálním čase, nebudou se ve výpočtu vyskytovat vysoká čísla a ani nalezení největšího společného dělitele časovou složitost nezhorší.

3.4 Shrnutí

Výsledky této kapitoly lze shrnout do následujícího tvrzení:

Věta 1. *Výpočet zobecněného diskrétního logaritmu v cyklické grupě řádu* $n = p_1^{e_1} \cdot \dots \cdot p_k^{e_k}$*, kde* p_i *jsou prvočísla, je polynomiálně ekvivalentní s problémem faktorizace čísla* n *a výpočtu diskrétních logaritmů v cyklických grupách řádů* p_i *pro* $i = 1, \dots, k$.

Důkaz. Plyne přímo z úvah v předchozím textu. □

Máme-li tedy k dispozici algoritmus, který efektivně počítá diskrétní logaritmus v grupě prvočíselného řádu, a algoritmus, který efektivně rozkládá čísla na prvočísla, můžeme s jejich pomocí zkonstruovat efektivní algoritmus pro výpočet obecného diskrétního logaritmu. Opačně známe-li efektivní algoritmus výpočtu zobecněného diskrétního logaritmu, můžeme ho pomocí konstrukcí uvedených výše v textu využít pro konstrukci efektivního algoritmu pro faktorizaci.

3.5 Implementace

Pollardův ρ algoritmus pro výpočet diskrétního logaritmu předpokládá, že pracuje v grupě prvočíselné velikosti. Není-li tento předpoklad zajištěn, může dojít k výraznému snížení efektivity; algoritmus je často nutné spouštět opakovaně od začátku.

Proto je výhodné využít techniku výpočtu diskrétního logaritmu popsanou v této kapitole i v reálné implementaci. Problém diskrétního logaritmu v dané grupě se redukuje na problém diskrétního logaritmu v podgrupách prvočíselné velikosti a z dílčích výsledků se poté dopočítá výsledek celkový.

V implementaci v jazyce Java tuto funkčnost zajišťuje třída `DLReductor` v balíku `alg`. Ta dostává kromě standardního vstupu také instanci algoritmu, kterou má použít pro finální výpočet v „malých“ podgrupách prvočíselné velikosti.

Její součástí musí být nějaký faktorizační algoritmus, který umožní rozpoznat vyskytující se podgrupy. Zdrojový kód v příloze proto obsahuje také implementaci kvadratického síta pro faktorizaci celých čísel. To sice není asymptoticky nejrychlejší známý algoritmus pro číselnou faktorizaci, v celkové časové složitosti Pollard ρ algoritmu, v němž je použita, je však složitost kvadratického síta zanedbatelná.

4. Algoritmus Baby steps - giant steps

4.1 Princip algoritmu

Algoritmus „Baby steps - giant steps“ byl publikován v roce 1971 matematikem Danielem Shanksem.

Mějme grupu $G = \langle g \rangle$ řádu n, nechť $t \in G$. Z vlastností cyklické grupy jistě existuje diskrétní logaritmus $l \in \{0, \ldots, n-1\}$ splňující $g^l = t$. Položme $m = \lceil \sqrt{n} \rceil$. Potom můžeme l psát jako číslo v bázi m ve tvaru $l = b_0 + b_1 m$. Této vlastnosti využijeme při hledání l, namísto l totiž budeme hledat hodnoty b_0 a b_1.

Provést algoritmus znamená především vytvořit dva seznamy prvků z G, jeden bude obsahovat prvky vzniklé takzvanými „dětskými kroky (baby steps)“, druhý takzvanými „obřími kroky (giant steps)“. První vytvořený seznam bude tvaru

$$L_b = \{1, g, \ldots, g^{m-1}\},$$

druhý seznam bude tvaru

$$L_g = \{t, tg^{-m}, \ldots, tg^{-(m-1)m}\}.$$

V dalším průběhu využijeme následující vlastnost: z definice b_0 a b_1 víme, že

$$t = g^l = g^{b_0 + b_1 m}.$$

Odtud

$$g^{b_0} = tg^{-b_1 m}.$$

Jelikož $0 \leq b_0, b_1 \leq m-1$, nachází se zcela jistě levá strana výše uvedené rovnosti v seznamu L_b a pravá strana v seznamu L_g. Protože se rovnají, leží zároveň v průniku obou seznamů. Úkol nalezení b_0 a b_1 se tím redukuje na nalezení společného prvku ve dvou stejně velkých seznamech.

Vzhledem k volbě m coby horní celé části $\sqrt{n}$ se může stát, že průnik obou seznamů nebude jednoprvkový, protože se v něm kromě hledané hodnoty l vyskytne ještě $l + kn$ pro nějaké malé k takové, že $n \leq l + kn < m^2$. Pokud tedy v průniku seznamů nalezneme taková b_0 a b_1, že $b_0 + b_1 m \geq n$, je hledaným diskrétním logaritmem hodnota $b_0 + b_1 m \mod n$.

4.2 Časová složitost

První část algoritmu, vytvoření obou seznamů, jsme zjevně schopni provést v čase $O(m)$. K vygenerování „baby steps“ seznamu L_b je totiž zapotřebí $m-2$ grupových operací, k výpočtu g^{-m} potom dvou dalších (jedna grupová operace a inverze), vygenerování „giant steps“ seznamu L_g dalších $m-1$. Výhodou algoritmu je, že seznam L_b stačí pro výpočet diskrétního logaritmu různých prvků v téže

grupě vytvořit právě jednou; na t nijak nezávisí. S vypočítáním každého nového prvku z L_g můžeme rovnou ověřit jeho náležení do L_b; průměrný počet prvků, které musí být vypočítány před nalezením prvku společného pro oba seznamy, je tedy oproti m přibližně poloviční.

Jelikož předpokládáme, že jsme schopni na prvcích grupy zavést linární uspořádání a pro dva prvky za jednu jednotku času rozhodnout, který je v tomto uspořádání větší a který menší, můžeme oba seznamy setřídit. To nám zabere $O(m \cdot \log m)$ času s použitím některého efektivního třídícího algoritmu (například algoritmu merge sort). V setříděných seznamech jsme již schopni pro daný prvek rozhodnout náležení pomocí binárního vyhledávání, tedy v čase $O(\log m)$.

Další výhodou algoritmu „Baby steps - giant steps“ je skutečnost, že řád grupy G nemusí být dopředu vůbec znám; stačí po každém neúspěšném běhu algoritmu zdvojnásobit m. Jakmile m dosáhne hranice $\sqrt{n}$, algoritmus jistě nalezne správné řešení.

Celková časová náročnost algoritmu „Baby steps - giant steps“ je tedy

$$O(m \log m) = O(\sqrt{n} \cdot \log n).$$

Algoritmus je tedy exponenciální v délce vstupu $\log n$. Zároveň má bohužel také nezanedbatelnou prostorovou složitost: udržovat seznam L_b vyžaduje paměť $O(\sqrt{n})$.

4.3 Algoritmus v pseudokódu

Obecnou verzi popisovaného algoritmu můžeme zachytit následujícím pseudokódem. Symbolem n v něm značíme řád grupy, g zvolený generátor a $t_1, \ldots t_k$ hodnoty na vstupu diskrétního logaritmu. Výstupem bude k-tice $s_1, \ldots, s_k$ taková, že

$$g^{s_i} = t_i \forall i = 1, \ldots k.$$

Verze zachycená pseudokódem plně využívá možnosti předpočítat L_b jednou pro všechny hodnoty na vstupu.

```
function BabyStepsGiantSteps(n, g, t_1, t_2, ..., t_k)
    L_b ← {}
    m ← ⌈√n⌉
    i ← 0
    h ← 1
    while i < m do
        L_b ← h
        h ← h · g
        i ← i + 1
    end while
    L_b ← sort(L_b)
    i ← 0
    r ← h^{-1}
    for j = 1 → k do
        h ← t_j
```

```
        while i < m do
            if (h = g^{b_0}) ∈ L_b then
                s_j ← b_0 + i · m
                break
            end if
            h ← h · r
            i ← i + 1
        end while
    end for
    return (s_1, ..., s_k)
end function
```

4.4 Optimalizace

Hlavní slabinou algoritmu je jeho prostorová složitost. Optimalizační snahy proto cílí především na efektivní uchovávání seznamu L_b, tedy seznamu „baby steps“. Jednou z možností je použít strukturu trie, která se obecně používá pro udržování množiny řetězců podobné délky a k rychlému rozhodování, zda daný řetězec v množině leží či nikoli.

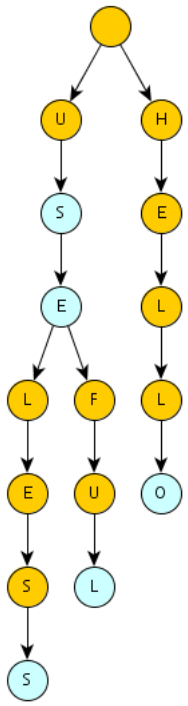
Předpokládejme, že jsme schopni každému prvku v grupě přiřadit konečný řetězec znaků, pro cyklickou grupu $\mathbb{Z}_n$ nebo $\mathbb{Z}_p^*$ to standardně může být zápis celého čísla ve zvolené soustavě. Tyto řetězce pak budeme držet ve stromové struktuře. Oproti běžnému vyhledávacímu stromu nebude jeden vrchol ve stromě odpovídat jednomu řetězci, ale jednomu znaku. Každý řetězec je ve struktuře trie reprezentován jednou souvislou cestou začínající v kořeni stromu a končící v některém z vrcholů stromu, přičemž jeho hodnota je zřetězením znaků v vrcholech na cestě od kořene do koncového vrcholu.

V každém vrcholu navíc musíme držet informaci, zda v něm končí některý z řetězců. Za platná slova v trii považujeme pouze cesty začínající v kořeni a končící v listech nebo některém z označených vrcholů.

Zavedení nového řetězce délky n do trie lze provést v čase $O(n)$, tedy v logaritmickém čase vzhledem k velikosti řetězcem reprezentovaného čísla. Podobně lze vyhledání řetězce v množině provést v čase $O(n)$.

V nejhorším případě, tedy pokud žádné dva řetězce nemají společný prefix, je prostorová složitost udržování množiny řetězců v trii stejná jako v běžném seznamu. Výhoda trie spočívá právě v uchovávání sdílených prefixů: je-li prefix sdílený k různými řetězci, je v trii uchováván pouze jednou, nikoli k-krát jako v případě běžného seznamu.

Chceme-li například v trii uchovat slova „useful“, „use“, „useless“, „us“ a „hello“, bude výsledná trie tvaru:



Vrcholy, ve kterých končí některý z držených řetězců, jsou označeny modře.

4.5 Implementace

V přiloženém zdrojovém Java kódu je algoritmus implementován v balíku `alg` ve třídě `BabyStepGiantStep`, která je instancí rozhraní `DLAlgorithmZp`. Podobně jako v ostatních implementovaných algoritmech byla pro pro práci s velkými čísly a pro efektivní modulární aritmetiku použita knihovna `JAS`.

Oproti pseudokódu je v reálné implementaci jeden rozdíl. Jelikož Java ve svých standardních knihovnách nabízí hešovací mapy (třídu `HashMap`), nebyl seznam „baby steps“ realizován jako klasický seznam, ale právě jako hešovací mapa. To vede ke zvýšení efektivity. Hešování umožňuje ověřovat, zda se daný prvek již nachází v množině, průměrně v konstantním čase. Odpadá tak složitost s případným řazením „baby steps“ a vyhledáváním v konstantním čase. Na druhou stranu tak vzrostou časové náklady o konstantní čas nutný k výpočtu heše, ten je ovšem v asymptotické složitosti algoritmu zanedbatelný.

5. Pollardův ρ algoritmus

Algoritmus Johna Pollarda bývá nazývan podle řeckého písmene ρ, jelikož jeho provedení svým principem písmeno připomíná: budeme hledat bod, v němž se zacyklí jistá posloupnost generovaných hodnot.

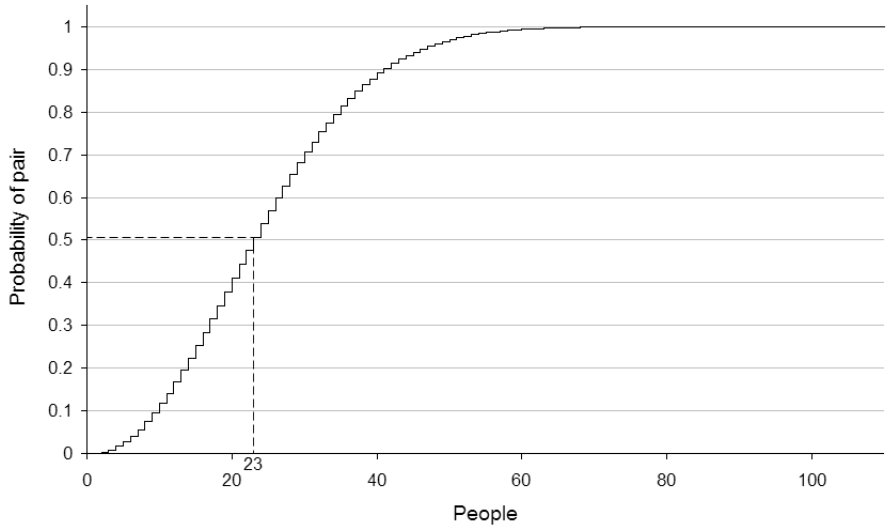
5.1 Narozeninový paradox

Heuristika ρ algoritmu vychází z takzvaného narozeninového paradoxu: Mějme n kuliček a k přihrádek. Nechť je každá kulička rovnoměrně náhodně zařazena do jedné přihrádky. A buď náhodný jev, který nastává, setkají-li se alespoň dvě kuličky v některé z přihrádek. Pak pro $n < k$ platí:

$$p(A) = 1 - (1 - \frac{1}{k})(1 - \frac{2}{k}) \cdots (1 - \frac{n-1}{k}).$$

První kulička může skončit v libovolné přihrádce, aniž by došlo ke kolizi, u druhé nenastane kolize s pravděpodobností $1 - \frac{1}{k}$, protože může skončit v libovolné přihrádce kromě té, ve které skončila první kulička, atd. Pro $n > k$ je pochopitelně $p(A) = 1$.

Narozeninovým paradoxem potom bývá nazýván následující překvapivý výsledek: A nastává s pravděpodobností $p \geq \frac{1}{2}$ již pro $n \geq \sqrt{k}$. Jev bývá ilustrován na následujícím příkladu: sejde-li se 23 lidí, pak (za předpokladu, že pravděpodobnost narození v daný den je pro každého člověka $\frac{1}{365}$ a data narození jsou ve sledované množině lidí nezávislé náhodné jevy) s nadpoloviční pravděpodobností mají alespoň dva z těchto lidí narozeniny ve stejný den.



5.2 Princip algoritmu

Mějme cyklickou grupu $G = \langle g \rangle$ řádu n. Uvážíme-li náhodnou posloupnost prvků

$$x_1, \ldots, x_k \in G,$$

podle narozeninového paradoxu pro $k \sim \sqrt{n}$ nastane první kolize, tedy $x_i = x_j$ pro $i \neq j$.

Pollardův ρ algoritmus této vlastnosti využívá. Posloupnost, která je v jeho průběhu generována, však není ryze náhodná. Naopak je nezbytné, aby posloupnost byla generována deterministicky, aby tedy každé x_{i+1} jednoznačně záviselo na x_i. Posloupnost však musí být pseudonáhodná, abychom se mohli opírat o narozeninový paradox. Právě pseudonáhodnost posloupnosti brání tomu, aby byl Pollardův algoritmus z hlediska časové složitosti rigorózně analyzován.

Generujme tedy posloupnost

$$S = \{g, f(g), f(f(g)), \dots\},$$

kde funkce $f : G \to G$ je snadno počitatelná pseudonáhodná funkce. Stejně jako v minulé kapitole předpokládáme, že prvky grupy můžeme seřadit, a tedy také rozdělit na tři třetiny podle velikosti. V [4] se pak jako příklad takové psuedonáhodné funkce uvádí

- $f(x) = t \cdot x$ pro x v první třetině
- $f(x) = x^2$ pro x ve druhé třetině
- $f(x) = g \cdot x$ pro x ve třetí třetině.

Symbolem x_i označme i-tý prvek posloupnosti S. Dále definujme pomocnou posloupnost dvojic (a_i, b_i) takto:

- $(a_0, b_0) := (0, 1)$
- $(a_{i+1}, b_{i+1}) := (a_i + 1 \mod n, b_i)$ pro x_i v první třetině pro $i \geq 0$
- $(a_{i+1}, b_{i+1}) := (2a_i \mod n, 2b_i \mod n)$ pro x_i ve druhé třetině pro $i \geq 0$
- $(a_{i+1}, b_{i+1}) := (a_i, b_i + 1 \mod n)$ pro x_i ve třetí třetině pro $i \geq 0$

Lze snadno nahlédnout, že takto definovaná posloupnost dvojic splňuje

$$x_i = t^{a_i} \cdot g^{b_i}.$$

Jakmile tedy nalezneme $x_i = x_j$, nutně platí $g^{b_i - b_j} = t^{a_j - a_i}$. Z pseudonáhodnosti P se navíc po dvojicích shodují všechny prvky v posloupnosti následující po x_i a x_j. Je-li tedy $(a_j - a_j)$ invertibilní v $\mathbb{Z}_n$, můžeme vypočítat diskrétní logaritmus takto:

$$g^{(b_i - b_j)(a_j - a_i)^{-1}} = (t^{a_j - a_i})^{(a_j - a_i)^{-1}} = t,$$

a tedy

$$\log_g t = (b_i - b_j)(a_j - a_i)^{-1} \mod n.$$

Podmínka invertibility není omezující; v předchozích kapitolách je popsán postup, jak převést problém diskrétního logaritmu v grupě řádu n na problém diskrétního logaritmu v podgrupách řádu prvočíselného. Můžeme tedy bez újmy na

obecnosti předpokládat, že řád grupy je prvočíslo, a všechny nenulové prvky jsou proto invertibilní. V případě $a_i = a_j$ algoritmus selhává a je nutné zvolit jinou počáteční hodnotu: například g^r pro náhodně zvolené $r \in \mathbb{N}$.

Zbývá vyřešit problém, jak efektivně rozpoznat, že došlo ke kolizi, tedy k zacyklení posloupnosti S. Případné porovnávání každého nového prvku v posloupnosti se všemi předchozími by totiž bylo časově velmi náročné. Řešením je použití Floydova algoritmu pro hledání cyklů v posloupnosti.

5.3 Floydův algoritmus pro hledání cyklů

Předpokládejme, že máme posloupnost prvků množiny M takovou, v níž je hodnota $(n + 1)$. prvku posloupnosti deterministicky stanovena hodnotou prvku n - tého. Takovou posloupností je také posloupnost $\{x_i\}_0^\infty$, funkce f, která je pro výpočet dalšího prvku použita, má sice pseudonáhodné vlastnosti, ale je ve skutečnosti deterministická.

Triviální postup by znamenal postupně projít všechny prvky posloupnosti a pro každý prvek ověřit, zda se již nenacházel mezi prvky předchozími. To vyžaduje buď značné požadavky paměťové na udržení množiny již nalezených prvků, anebo časové na zpětné procházení předchozích prvků.

Floydův algoritmus postupuje při hledání cyklů v posloupnosti jinak: prochází postupně všechny prvky posloupnosti od prvním počínaje a každý porovnává s prvkem na pozici dvojnásobné. Využívá faktu, že Pollardův ρ algoritmus nutně nemusí nalézt první kolizi v posloupnosti, stačí nalézt kolizi libovolnou. Algoritmus proto hledá kolizi speciálního tvaru

$$x_k = x_{2k}.$$

Floydův algoritmus tak oproti triviálnímu algoritmu musí projít více prvků, než nalezne dva shodné, odpadá však náročné ověřování. Díky tomu má Floydův algoritmus pouze lineární složitost $O(a + b)$, kde a je délka úvodní neperiodické části posloupnosti a b je délka jedné periody [3].

5.4 Časová složitost

Podobně jako algoritmus „Baby steps - giant steps“ také Pollardův algoritmus má heuristicky exponenciální složitost vzhledem k velikosti vstupu, konkrétně $O(\sqrt{n})$. Průměrně totiž potřebujeme vygenerovat $\sqrt{n}$ členů posloupnosti, než dojde k první kolizi. Floydův algoritmus vyžaduje pouze lineární čas, horní odhad složitosti se tím proto nezhorší.

Hlavní výhodou oproti algoritmu „Baby steps - giant steps“ je složitost prostorová. Zatímco „Baby steps - giant steps“ vyžadoval nezanedbatelných $O(\sqrt{n})$ paměti, Pollarův ρ algoritmus má paměťové nároky nepatrné: v každém okamžiku běhu je nutno udržovat pouze informaci o jednom či dvou členech posloupnosti, z nichž bude v následujícím kole vypočítán člen další, a prostorová složitost je proto pouze $O(1)$.

5.5 Algoritmus v pseudokódu

V níže uvedeném pseudokódu je Pollardův algoritmus přehledně zachycen. Na vstupu očekává řád grupy n, generátor g, funkci na trojici $(\langle g \rangle, \mathbb{Z}_n, \mathbb{Z}_n)$ `pseudorand`, která je na $\langle g \rangle$ pseudonáhodnou funkcí, a logaritmovanou hodnotu t. Výstupem je diskrétní logaritmus t. Předpokládá se, že algoritmus má přístup ke generátoru náhodných čísel `rand(k)`, který s rovnoměrným rozložením pravděpodobnosti vrací hodnotu z $\{0, \ldots, k-1\}$.

function PollardRho$(n, g, \text{pseudorand}, t)$
  label start
  $r \leftarrow \text{rand}(n)$
  $x \leftarrow g^r$
  $a \leftarrow 0$
  $b \leftarrow r$
  $(x, a, b) \leftarrow \text{pseudorand}(x, a, b)$
  $(X, A, B) \leftarrow \text{pseudorand}(\text{pseudorand}(g^r, 0, r))$
  while $x \neq X$ **do**
    $(x, a, b) \leftarrow \text{pseudorand}(x, a, b)$
    $(X, A, B) \leftarrow \text{pseudorand}(\text{pseudorand}(X, A, B))$
  end while
  if $\gcd(A - a, n) \neq 1$ **then**
    goto start
  end if
  return $(b - B) \cdot (A - a)^{-1}$
end function

5.6 Optimalizace

Pollardův algoritmus je heuristický a vychází z těžko ověřitelných předpokladů, zejména z pseudonáhodnosti zvolené funkce. Pokud pro nějakou pevně danou grupu selže, je možné volit jinou pseudonáhodnou funkci a doufat v lepší výsledek.

5.7 Implementace

Algoritmus je implementován v přiloženém zdrojovém kódu ve třídě `PollardRo` v balíku `alg`. Třída `PollardRo` implementuje výpočet diskrétního logaritmu v tělesech prvočíselné velikosti. Pro práci s velkými čísly a pro modulární aritmetiku využívá knihovnu `JAS`.

Co se dělení na tři skupiny při výpočtu další hodnoty pseudonáhodné posloupnosti týče, používá standardní dělení intervalu. Dalším možným přístupem by bylo dělení modulární: rozřadit prvky grupy do skupiny podle jejich příslušnosti do zbytkové třídy modulo 3. Jsou ovšem grupy, ve kterých tento přístup selhává a takto definovaná funkce f nesplňuje požadavky na pseudonáhodnost.

Obecný `PollardRo` algoritmus může ve své finální fázi selhat, pokud hodnota $a_i - a_j$ není invertibilní v multiplikativní struktuře okruhu exponentů. Tvůrce

algoritmu navrhuje tuto slabinu řešit opakovaným výpočtem pseudonáhodné posloupnosti pro jiný počáteční prvek.

Implementace, která je součástí této práce, přistupuje k problému jinak: třída `PollardRo` je při testování používána zásadně prostřednictvím třídy `DLReductor`, která zajistí rozklad problému do podgrup prvočíselné velikosti a následné dopočítání celkového výsledku z dílčích výsledků pro jednotlivé podgrupy.

Pravděpodobnost možného selhání algoritmu je tak minimalizována, může k němu dojít pouze v případě, že hodnota, jejíž inverzi hledáme, bude 0. V takovém případě je zvoleno náhodné r a algoritmus je spuštěn znovu pro počáteční prvek g^r.

6. Index kalkulus

Algoritmus index kalkulus pro výpočet diskrétního logaritmu nefunguje v obecné grupě, využívá struktury celých čísel nebo polynomů, a je proto aplikovatelný pouze v jisté třídě grup, konkrétně v $\mathbb{F}_n^*$, kde n je mocnina prvočísla. Pro některé často používané grupy není aplikace známa, pro grupu bodů na eliptické křivce například zůstává nejlepším známým algoritmem obecný „Baby steps - giant steps“. Pro $n = p^m$, kde p je prvočíslo, existují algoritmy s lepší asymptotickou složitostí: metoda založená na číselném sítu a funkční síto. Index kalkulus byl představen v [13] a podrobněji analyzován v [14].

6.1 Princip algoritmu

Základní myšlenkou index kalkulu je zavedení takzvaných hladkých prvků. Každý prvek $\mathrm{GF}(p^m)$ lze reprezentovat jako polynom nad $\mathbb{F}_p$ stupně menšího než m. Pro stanovenou mez hladkosti b rozumíme b-hladkým prvkem takový prvek, jehož polynomiální reprezentace $f \in \mathbb{F}_p[x]$ se rozkládá do mocnin faktorů stupně menšího než b, tedy

$$f = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k},$$

kde $\deg(g_i) < b$ pro všechna $i \in \{1, \ldots, k\}$.

Fixujme tedy b mez hladkosti, její přesná hodnota bude určena později. Dále definujme takzvanou bázi hladkých prvků

$$S := \{g \in \mathbb{F}_p[x] | \deg(g) < b, g \text{ ireducibilní}\}.$$

V úvodní fázi algoritmu budou spočítány diskrétní logaritmy všech prvků z báze S. Tato fáze nezávisí na vstupním prvku, pouze na zvolené grupě. Má-li tedy být spočítáno více diskrétních logaritmů v téže grupě, stačí provést úvodní fázi pouze jednou.

Cílem index kalkulu je získání dostatečného počtu lineárních vztahů mezi diskrétními logaritmy bázových prvků z množiny S. Jakmile je dostatečný počet vztahů znám, je možné soustavu rovnic vyřešit a získat pro každý bázový prvek jednoznačný výsledek. V dalším odstavci ukážeme, jak získat jeden takový lineární vztah.

Zvolme náhodné celé číslo $s \in \{1, \ldots, p^m - 1\}$ a spočítejme $g^s \pmod f$, kde g je primitivní prvek a f je zvolený ireducibilní polynom stupně m takový, že $\mathrm{GF}(p^m) = \mathbb{F}_p[x]/(f)$. Dále ověříme, zda je vypočítaný polynom b-hladký. Pokud ne, pokračujeme novou volbou s. V opačném případě dostáváme vztah

$$g^s \pmod f = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k} \pmod f,$$

kde $g_i \in S$ pro $i = 1, \ldots, k$. Odtud zlogaritmováním dostáváme

$$s = \sum_{i=1}^{k} e_i \log g_i \pmod{p^m - 1}.$$

Každá úspěšná volba s tedy znamená jednu lineární rovnici. Získáme-li takových rovnic více než $|S|$ lineárně nezávislých, můžeme logaritmus všech prvků z S jednoznačně určit.

Jelikož je matice nalezené soustavy lineárních rovnic řídká (je tvořena převážně nulami) a soustava je počítána nad konečným tělesem, je možné použít některý z rychlých algoritmů, například Wiedemannovu metodu popsanou v [15].

V další fázi zvolme náhodné celé číslo s takové, že $1 \leq s \leq p^m - 1$ a spočítejme

$$h^* = h \cdot g^s \pmod{f}.$$

h^* nyní rozložíme na ireducibilní polynomy a ověříme, zda všechny faktory leží v množině S, tedy zda je h^* b-hladký. Pokud ne, pokračujeme novou volbou s. V opačném případě dostáváme:

$$h^* = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k},$$

a tedy

$$\log_g h = \sum_{i=1}^{k} e_i \log_g g_i - s \pmod{p^m - 1}.$$

Metoda index kalkulu probíhá ve třech krocích:

1. Získáme dostatečný počet lineárních vztahů mezi prvky z množiny S.
2. Vyřešíme vzniklou soustavu lineárních rovnic a získáme diskrétní logaritmy prvků z S.
3. Spočítáme diskrétní logaritmus daného prvku za pomoci diskrétních logaritmů prvků z množiny S.

Kromě posledního kroku nezávisí žádný z kroků na vstupním prvku. Díky tomu můžeme považovat všechny kroky kromě posledního za předvýpočet, který může pro více vstupních prvků z téhož tělesa běžet právě jednou.

Podrobněji lze jednotlivé kroky psát jako:

1. Krok 1

   Nalezněme všechny prvky stupně menšího než zvolené b, $S = \{g_1, \ldots, g_k\}$.

   Pro $i = 1, \ldots, t$ spočítejme $g^{s_i} = g_1^{e_{i1}} \cdot g_2^{e_{i2}} \cdots g_k^{e_{ik}}$.

2. Krok 2

   Odvoďme lineární rovnice $s_i = e_{i1} \log g_1 + \ldots + e_{ik} \log g_k$.

   Vyřešme soustavu rovnic zadanou maticí

$$\begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1k} \\ e_{21} & e_{22} & \cdots & e_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ e_{t1} & e_{t2} & \cdots & e_{tk} \end{pmatrix} \begin{pmatrix} \log g_1 \\ \log g_2 \\ \vdots \\ \log g_k \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_t \end{pmatrix} \pmod{p^m - 1}$$

3. Krok 3

   Zvolme s náhodné a spočítejme $h^* = hg^s$

   Pokud $h^* = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k}$, dostáváme $\log h = e_1 \log g_1 + \ldots + e_k \log g_k - s$.

Počet hledaných vztahů t lze zpravidla volit jako $2|S|$. Při této volbě t s vysokou pravděpodobností dostáváme množinu, v níž je alespoň $|S|$ lineárně nezávislých rovnic.

6.2 Pravděpodobnost volby b-hladkého polynomu

Pravděpodobnost, že pro náhodně zvolené s je h^* nebo g^s b-hladký, zřejmě závisí na mezi hladkosti b. Předpokládejme, že pro náhodně zvolené s je v obou případech výsledkem vždy náhodný polynom z $\mathbb{F}_p[x]$ stupně menšího než m. Pro jednoduchost můžeme rovněž předpokládat, že h^* je monický; pokud není, vydělíme ho vždy jeho vedoucím koeficientem a v algoritmu budeme pracovat výhradně s monickými polynomy. Nechť $p(k, b)$ značí pravděpodobnost, že náhodně zvolený monický polynom z $\mathbb{F}_p[x]$ stupně právě k je b-hladký. $N(k, b)$ buď počet takových polynomů. Pak

$$p(k, b) = \frac{N(k, b)}{N(k, k)} = \frac{N(k, b)}{p^k}.$$

Náhodně zvolený monický polynom je tedy b-hladký s pravděpodobností alespoň $p(m, b)$.

Symbolem $I(k)$ označme počet monických ireducibilních polynomů nad $\mathbb{F}_p$ stupně k. Lze ukázat [14], že

$$I(k) = \frac{1}{k} \sum_{d|k} \mu(d) p^{k/d},$$

kde μ je Möbiova funkce, tedy:

- $\mu(n) := 1$ pro n bezčtvercové celé číslo se sudým počtem různých prvočíselných dělitelů,
- $\mu(n) := -1$ pro n bezčtvercové celé číslo s lichým počtem různých prvočíselných dělitelů,
- $\mu(n) := 0$ pro n, které není bezčtvercové.

Z výše uvedené formulky plyne, že

$$I(k) = k^{-1} p^k + O(k^{-1} p^{k/2}).$$

Nyní dodefinujme $N(0, 0) = 1$, $N(k, 0) = 0$ pro $k \neq 0$, $N(k, m) = 0$ pro $k < 0$ a $m \geq 0$.

Pro uvedené hodnoty brzy dokážeme následující rekurzivní vztah:

$$N(n,m) = \sum_{k=1}^{m} \sum_{r\geq 1} N(n-rk, k-1) \binom{r+I(k)-1}{r}.$$

Než dokážeme uvedené tvrzení, proveďme pozorování: každý monický polynom f stupně m, který je b-hladký, lze psát jednoznačně jako

$$f(x) = g(x) \prod_{u(x)} u(x)^{a(u(x))},$$

kde $u(x)$ jsou všechny monické ireducibilní stupně b pro nějaké $k \in \{1, \ldots, m\}$ a platí

$$\sum a(u(x)) = r$$

pro r kladné celé číslo. $g(x)$ je monický polynom stupně $m - rb$, který je $(b-1)$-hladký. V polynomu $g(x)$ jsou tedy obsaženy ty ireducibilní faktory f, které jsou stupně nejvýše $(b-1)$, zatímco faktory stupně b jsou právě polynomy $u(x)$.

Máme-li pevné r a b, existuje $N(m-rb, b-1)$ takových polynomů $g(x)$. Zvolit $\prod_{u(x)} u(x)^{a(u(x))}$ znamená zvolit $I(b)$-tici nezáporných celých čísel, jejichž součet je r. Takových $I(b)$-tic je právě

$$\binom{r+I(b)-1}{r}.$$

Dokázaný rekurzivní vztah lze využít k výpočtu přesného počtu b-hladkých monických polynomů daného stupně nad konečným tělesem. Asymptotický odhad plyne z následujících vět, důkazy těchto tvrzení jsou detailně vyloženy v [14].

Věta 2. *Nechť*

$$f_m(z) = \prod_{k=1}^{m} (1-z^k)^{I(k)}$$

a

$$b(z) = \left(\frac{f_m'}{f_m}(z)\right)' = \sum_{k=1}^{m} I(k)\left(\frac{k(k-1)z^{k-2}}{1-z^k} + \frac{k^2 z^{2k-2}}{(1-z^k)^2}\right).$$

Pak pro $n^{1/100} \leq m \leq n^{99/100}$ platí

$$N(n,m) \approx (2\pi b(r_0))^{-\frac{1}{2}} f_m(r_0) {r_0}^{-n} \text{ pro } n \to \infty,$$

kde $r = r_0 = r_0(m,n)$ je řešením rovnice

$$r\frac{f_m'}{f_m}(r) = n.$$

Věta 3. *Pro $m^{1/100} \leq b \leq m^{99/100}$ platí $N(m,b) = p^m (\frac{b}{m})^{(1+o(1))\frac{m}{b}}$ pro $m \to \infty$.*

Věta 4. *Platí*

$$p(m,b) = \exp((1+o(1))\frac{m}{b}\log\frac{b}{m}).$$

6.3 Časová složitost

Efektivita index kalkulu závisí na vhodně zvolené mezi hladkosti. Vysoká mez hladkosti b znamená vysokou pravděpodobnost volby b-hladkého polynomu, ale zároveň nutnost nalézt takových polynomů více, protože generovaná soustava bude mít více neznámých. Naopak nízká mez hladkosti matici zmenšuje, a šetří tak čas v závěrečné, lineární fázi, ale výrazně snižuje pravděpodobnost nalezení hladké relace. Nejprve definujme značení, takzvanou L-notaci, pomocí níž lze přehledně zachytit složitost subexponenciálních algoritmů.

Definice. *Buď $N \in \mathbb{N}$, $s \in \mathbb{R}$, $c \in \mathbb{R}$, $0 \leq s \leq 1$. Pak třídou funkcí $L_N[s;c]$ rozumíme třídu funkcí*

$$L_N[s;c] = \exp((c + o(1))(\log N)^s(\log\log N)^{1-s}).$$

L-notace slouží k odhadování složitostí algoritmů od polynomiálních v $\log N$, pro $s = 0$, až po exponenciální v $\log N$ pro $s = 1$. Konstanta c je méně významná, všechny třídy se stejným parametrem s jsou nezávisle na c polynomiálně ekvivalentní.

Odlyzko se v [14] problémem zabývá podrobněji a dochází k závěru, že vhodnou volbou meze hladkosti je

$$b = \frac{1}{2}cm^{\frac{1}{2}}(\log m)^{\frac{1}{2}}$$

pro $c = (\log 2)^{-\frac{1}{2}}$.

Polynomů v hladké bázi je pak nejvýše $L_n[\frac{1}{2};c]$, hledáme tedy $2L_n[\frac{1}{2};c]$ hladkých relací. Předpokládáme, že h^* se v závislosti na s chová jako náhodný monický polynom stupně nejvýše m nad $\mathbb{F}_p$. Abychom nalezli $2L_n[\frac{1}{2};c]$ hladkých relací, vyzkoušíme přibližně $p(m,b)^{-1} \cdot 2L_n[\frac{1}{2};c]$ polynomů, protože pravděpodobnost, že je náhodný monický polynom stupně nejvýše m b-hladký, je alespoň taková jako pravděpodobnost, že je náhodný polynom stupně právě m b-hladký.

Pro pevně stanovené $b = \frac{1}{2}cm^{\frac{1}{2}}(\log m)^{\frac{1}{2}}$ lze $p(m,b)$ odhadnout:

$$p(m,b) = p(m, \frac{1}{2}cm^{\frac{1}{2}}(\log m)^{\frac{1}{2}}) \approx \exp(m^{\frac{1}{2}}(\log m)^{-\frac{1}{2}} \cdot (\log m + \log\log m)) = L_n[\frac{1}{2}].$$

Každý hladký polynom jsme schopni nad konečným tělesem rozložit a otestovat v polynomiálním čase. Vyzkoušíme tedy přibližně celkem

$$p(m,b)^{-1} \cdot 2L_n[\frac{1}{2};c] = L_n[\frac{1}{2}] \cdot L_n[\frac{1}{2};c] = L_n[\frac{1}{2}]$$

polynomů, než nalezneme dostatečný počet hladkých relací.

6.4 Coppersmithův algoritmus pro $\mathrm{GF}(2^n)$

Algoritmus popsaný v předchozím textu lze do značné míry optimalizovat [14], asymptoticky významné zlepšení ovšem přináší teprve Coppersmithův algoritmus. Ten využívá některých vlastností tělesa charakteristiky 2 a v ostatních konečných tělesech ho není možné použít. Podle [14] dosahuje heuristické složitosti

$$L[\frac{1}{3}; 1],$$

což znamená výrazné zrychlení oproti standardnímu index kalkulu.

Jednou z klíčových myšlenek je vhodná volba tělesa tak, aby se v něm „dobře počítalo“. Násobení v tělese $\mathrm{GF}(2^n)$ je proto realizováno pomocí primitivního polynomu f, který je tvaru

$$f(x) = x^n + f_1(x),$$

kde f_1 je nízkého stupně, konkrétně $\deg(f_1(x)) \leq \log n$. V běžném index kalkulu jsme volili náhodná přirozená čísla s a ověřovali, zda je $x^s \pmod f$ hladký. Pokud byl, získali jsme jeden nový lineární vztah mezi diskrétními logaritmy bázových prvků. Coppersmith vylepšil krok 1 běžného index kalkulu, lineární vztahy získává efektivnější cestou, což mu umožní zvýšit pravděpodobnost úspěšného nalezení hladké kongruence.

V průběhu Coppersmithova algoritmu budeme pracovat se dvěma polynomy u_1 a u_2 nad $\mathbb{F}_2$, které mají nízký stupeň (nižší než pevně zvolené d) a jsou nesoudělné. Předpokládejme, že u_1 a u_2 byly zvoleny náhodně a požadavek na nesoudělnost byl ověřen Euklidovým algoritmem. Z polynomů u_1 a u_2 sestavíme polynom w_1 takto:

$$w_1 = u_1 x^h + u_2,$$

kde h je nějaké pevně zvolené přirozené číslo.

Takto vytvořený polynom w_1 má pro libovolné přirozené číslo k, pro které platí $h2^k \geq n$, následující vlastnost:

$$w_1^{2^k} = (u_1 x^h + u_2)^{2^k} \pmod f = u_1^{2^k} x^{h2^k} + u_2^{2^k} \pmod f = u_1^{2^k} f_1 x^{h2^k - n} + u_2^{2^k} \pmod f.$$

V první rovnosti využíváme faktu, že výpočet probíhá v tělese charakteristiky 2, v poslední potom vhodné volby $f(x) = x^n + f_1(x)$. Označme nyní

$$w_2 = u_1^{2^k} f_1 x^{h2^k - n} + u_2^{2^k}.$$

Ze vztahu výše tak dostáváme

$$w_1^{2^k} = w_2 \pmod f.$$

Takto získané kongruence poslouží k získání lineárních vztahů mezi diskrétními logaritmy podobně jako x^s pro náhodné s v klasickém index kalkulu. Počáteční

ověření nesoudělnosti u_1 a u_2 činí volbu efektivnější - cíleně se vyhýbáme nadbytečným, už spočítaným vztahům, které snižují efektivitu běžného index kalkulu. Předpokládejme pro ilustraci případ, kdy u_1 a u_2 nesoudělné nejsou, tedy existuje netriviální $e \in \mathbb{F}_2[x]$ takové, že $\gcd(u_1, u_2) = e$. Pak rovněž platí

$$w_1^{2^k} = w_2 \pmod{f},$$

ale e je nyní dělitelem obou stran, a tedy dostáváme tentýž lineární vztah jako pro $\frac{u_1}{e}$ a $\frac{u_2}{e}$ (na obou stranách rovnice jsou tytéž členy, které se odečtou).

Zbývá zodpovědět otázku, jak v tomto případě vhodně volit parametry d, 2^k, h a b. Podrobná analýza, uvedená v [14], ukazuje, že je vhodné volit:

$$d = n^{\frac{1}{3}}(\log n)^{\frac{2}{3}}$$

$$2^k = (\frac{n}{\log n})^{\frac{1}{3}}$$

$$h = \lfloor \frac{n}{2^k} \rfloor + 1$$

$$b = n^{\frac{1}{3}}(\log n)^{\frac{2}{3}}.$$

Shrnutí algoritmu v krocích:

1. Krok 1

   Nalezněme všechny ireducibilní prvky stupně menšího než zvolené b, $S = \{g_1, \ldots, g_k\}$.

   Procházíme možné dvojice u_1 a u_2 a počítáme prvky w_1 a w_2, dokud nenalezneme alespoň t takových dvojic, že $w_1^{2^k}$ a w_2 jsou b-hladké, tedy

$$g_1^{e_{i1}} \cdot g_2^{e_{i2}} \cdots g_k^{e_{ik}} = w_1^{2^k} = w_2 = g_1^{f_{i1}} \cdot g_2^{f_{i2}} \cdots g_k^{f_{ik}}.$$

2. Krok 2

   Odvoďme lineární rovnice

$$(e_{i1} - f_{i1}) \log g_1 + \ldots + (e_{ik} - f_{ik}) \log g_k = 0.$$

   Vyřešme soustavu rovnic zadanou maticí

$$\begin{pmatrix} (e_{11} - f_{11}) & (e_{12} - f_{12}) & \cdots & (e_{1k} - f_{1k}) \\ (e_{21} - f_{21}) & (e_{22} - f_{22}) & \cdots & (e_{2k} - f_{2k}) \\ \vdots & \vdots & \ddots & \vdots \\ (e_{t1} - f_{t1}) & (e_{t2} - f_{t2}) & \cdots & (e_{tk} - f_{tk}) \end{pmatrix} \begin{pmatrix} \log g_1 \\ \log g_2 \\ \vdots \\ \log g_k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \pmod{2^n - 1}$$

3. Krok 3

   Zvolme s náhodné a spočítejme $h^* = hg^s$

   Pokud $h^* = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k}$, dostáváme $\log h = e_1 \log g_1 + \ldots + e_k \log g_k - s$.

Heuristická analýza, vycházející z předpokladu, že se oba volené polynomy chovají jako náhodné polynomy nad $\mathbb{F}_2$ téhož stupně, vede podle [14] na odhad složitosti

$$\exp((1+o(1))n^{\frac{1}{3}}(\log n)^{\frac{2}{3}}) = L_{2^n}[\frac{1}{3};1],$$

což je dramatické vylepšení ve srovnání s běžným index kalkulem. Pro předpoklad náhodnosti ovšem existují pouze statistické dohady, odhad je tedy odhadem v nematematickém slova smyslu a má výpovědní hodnotu pouze pro „malá“ tělesa (všechna tělesa v praxi použitelná), u nichž byly konkrétní hodnoty přesně spočítány.

6.5 Semaevův algoritmus

Možností, jak vylepšit základní myšlenku index kalkulu, existuje celá řada, jednou z nich je řešení navržené Semaevem v roce 1994. Základní princip je podobný jako u algoritmu Coppersmithova. Semaev se snaží získávat lineární vztahy mezi diskrétními logaritmy bázových prvků efektivně, tedy s co nejvyšší pravděpodobností, že získaná kongruence mezi polynomy bude b-hladká.

Podobně jako Coppersmith, předpokládá Semaev vhodnou volbu ireducibilního polynomu f. Proto je také kladen požadavek na těleso, ve kterém výpočet probíhá. Za tělesa vhodná pro Semaevovu metodu považujeme tělesa $\mathrm{GF}(2^n)$, kde 2^n-1 je takové, že existuje r splňující $r \mid (2^n-1)$ a zároveň $r \not| (2^k-1)$ pro $k<n$. Jedním z případů, kdy je tato podmínka splněna, je volba $r=n+1$, pokud je takto zvolené r prvočíslo a 2 je primitivním prvkem $\mathbb{Z}_r$. Pro taková n je cyklotomický polynom

$$f(x) = x^n + x^{n-1} + \cdots + 1$$

ireducibilní a může být použit pro konstrukci tělesa.

Mějme n vyhovující Semaevovým předpokladům. Volme $u = 2^k \pmod r$ a $m = \lfloor n^{\frac{2}{3}} \rfloor$. Pro tato čísla definujme množiny:

$$T_u = \{0 \le l < r | l < m, (lu \mod r) \le m\}$$

$$T_{i_u} = \{ui \mod r | i \in T_u\}.$$

Nechť $w \in \mathrm{GF}(2^n)^*$ generuje podgrupu velikosti r. Taková podgrupa podle Lagrangeovy věty jistě existuje, neboť $r \mid (2^n-1)$.

V dalším kroku z množin T_u a T_{i_u} zkonstruujeme polynomiální kongruenci, kterou (bude-li b-hladká) využijeme k nalezení lineárního vztahu mezi diskrétními logaritmy bázových prvků. Definujme

$$c = \sum_{i \in T_u} a_i w^i,$$

kde $a_i \in \mathbb{F}_2$, a

$$d = \sum_{i \in T_u} a_i w^{i_u},$$

kde a_i jsou stejné a $i_u = ui \mod r$.

Pak platí

$$c^{2^k} = (\sum_{i \in T_u} a_i w^i)^u = \sum_{i \in T_u} a_i w^{ui} = \sum_{i \in T_u} a_i w^{i_u} = d.$$

V rovnostech výše používáme postupně definici c, skutečnost, že pracujeme nad tělesem charakteristiky 2, definici i_u a definici d.

Velkými písmeny C a D označme polynomy z $\mathbb{F}_2[x]$ odpovídající prvkům c a d tělesa $\mathrm{GF}(2^n)$. Z rovnosti výše dostáváme hledanou polynomiální kongruenci

$$C^{2^k} = D \pmod f.$$

Polynomy C a D byly zkontruovány tak, aby měly stupeň nejvýše m. Díky tomu je nalezená kongruence b-hladká s vyšší pravděpodobností než u zcela náhodné volby. Je důležité si uvědomit, že Semaevova metoda dává značné omezení na počet párů, které můžeme využít při hledání kongruencí. Konkrétně jich může být nejvýše $2^{|T_u|}$ podle volby u, neboť můžeme volit a_i pro každý koeficient polynomu jako 0 nebo 1. Je-li u zvoleno nevhodně, může se stát, že množina T_u bude malá a v prvním kroku nebudeme schopni nalézt dostatečné množství lineárních vztahů, aby měla výsledná soustava jednoznačné řešení.

Semaev ve své práci ukázal, že pro naprostou většinu voleb u je množina T_u dostatečně velká. Rigorózní důkaz však neexistuje, podobně jako rigorózní důkaz, že je tato metoda efektivnější než běžný index kalkulus. Jako obvykle se při analýzách vychází z nedokázaného předpokladu, že se takto vytvořené polynomy C a D chovají jako náhodně zvolené polynomy jistého omezeného stupně.

Další nevýhodou Semaevovy metody je fakt, že pro takto zvolený polynom f už prvek x není primitivním prvkem. Umocňování nového generátoru multiplikativní grupy v tělese tak může být z hlediska hardwaru časově výrazně složitější.

Shrnutí algoritmu v krocích:

1. Krok 1

   Nalezněme všechny ireducibilní prvky stupně menšího než zvolené b, $S = \{g_1, \ldots, g_k\}$.

   Nalezneme t dvojic C a D, pro které platí, že C^{2^k} a D jsou oba b-hladké, tedy

   $$g_1^{e_{i1}} \cdot g_2^{e_{i2}} \cdots g_k^{e_{ik}} = C^{2^k} = D = g_1^{f_{i1}} \cdot g_2^{f_{i2}} \cdots g_k^{f_{ik}}.$$

   Pokud byly vyčerpány všechny možné páry a nebylo nalezeno t hladkých kongruencí, pokračujme znovu pro jinou volbu u.

2. Krok 3

   Odvoďme lineární rovnice

$$(e_{i1} - f_{i1}) \log g_1 + \ldots + (e_{ik} - f_{ik}) \log g_k = 0.$$

   Vyřešme soustavu rovnic zadanou maticí

$$\begin{pmatrix} (e_{11} - f_{11}) & (e_{12} - f_{12}) & \cdots & (e_{1k} - f_{1k}) \\ (e_{21} - f_{21}) & (e_{22} - f_{22}) & \cdots & (e_{2k} - f_{2k}) \\ \vdots & \vdots & \ddots & \vdots \\ (e_{t1} - f_{t1}) & (e_{t2} - f_{t2}) & \cdots & (e_{tk} - f_{tk}) \end{pmatrix} \begin{pmatrix} \log g_1 \\ \log g_2 \\ \vdots \\ \log g_k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \pmod{2^n - 1}$$

3. Krok 3

   Zvolme s náhodné a spočítejme $h^* = hg^s$

   Pokud $h^* = g_1^{e_1} \cdot g_2^{e_2} \cdots g_k^{e_k}$, dostáváme $\log h = e_1 \log g_1 + \ldots + e_k \log g_k - s$.

6.6 Polynomiální síta

Klíčovou (a z hlediska časové složitosti velmi náročnou) částí všech tří uvedených verzí index kalkulu je test hladkosti. Faktorizace polynomů je sice možná v polynomiálním čase vzhledem k velikosti vstupu, i tak však její provedení znamená nezanedbatelnou časovou složitost. Proto by bylo vhodné nějak usnadnit rozhodování, zda je polynom b-hladký, aniž by musel být faktorizován. Jde o přístup analogický k sítům pro hledání prvočísel, například k všeobecně známému sítu Eratosthenovu.

V následujícím odstavci se budeme zabývat polynomiálním sítem, které představili McCurley a Gordon roce 1992. Jde o zlepšení Coppersmithovy metody popsané v jednom z předchozích odstavců. Otázka, na kterou McCurley a Gordon odpovídají, zní: jak volit u_1 a u_2 tak, aby byl w_1 s vysokou pravděpodobností hladký? Nebo přesněji: jak pro dané u_1 rozpoznat vhodný polynom u_2? Metoda vychází z náhodné pevné volby polynomu u_1, ke kterému pomocí přesívání nalezneme taková u_2, aby byl w_1 b-hladký.

Podaří-li se nám odfiltrovat většinu neperspektivních kandidátů na w_1, výrazně ubyde testů hladkosti: pro každé b-hladké w_1 budeme na hladkost obvyklým způsobem testovat pouze w_2.

Podobně jako čísla v sítu Eratosthenově je zapotřebí polynomy nějak seřadit a očíslovat, abychom pak mohli označovat ty, které má smysl faktorizovat a které naopak už zcela jistě b-hladké nejsou. Přirozeným seřazením je chápat polynom jako vektor koeficientů a tento vektor jako zápis čísla v binární soustavě. Můžeme tedy snadno sestrojit pole prvků očíslované polynomy.

Předpokládejme tedy, že máme pole očíslované polynomy, na každé pozici v tomto poli je kladné celé číslo, na počátku 0. Mějme u_1 pevné. Samotné prosívání polynomů potom probíhá takto:

1. Zvolme možné u_2 a spočítejme w_1.
2. Projděme všechny dosud nepoužité ireducibilní polynomy g_j stupně menšího než b a ověřme, zda dělí w_1. Pokud ano, navyšme prvek v poli na pozici aktuálního u_2 o stupeň g_j.
3. Navyšme o stupeň g_j rovněž všechny ostatní prvky v poli, které odpovídají polynomům dělitelným g_j. Polynom g_j označme jako použitý a postup opakujme pro další nepoužitý polynom g_j.

Symbolem h nyní budeme rozumět hodnotu z popisu Coppersmithova algoritmu, pomocí které je z u_1 a u_2 definován w_1. Kandidáti na u_2, u nichž bude v poli po dokončení síta hodnota vyšší než

$$\deg(w_1) - b = \deg(u_1 \cdot x^h) - b = \deg(u_1) + h - b,$$

budou podrobeni dalším testům a využiti pro hledání hladkých kongruencí. Ostatní budou vyřazeni, k hladké kongruenci zcela jistě nevedou, mají nějakého dělitele stupně většího než b.

Naskýtá se otázka, jak ze znalosti, že g_j dělí jeden možný polynom u_2, rozeznat, které další potenciální u_2 polynom g_j také dělí. V Eratosthenově sítu se řeší podobný problém: víme, že prvočíslo p dělí celé číslo n a chceme nalézt ostatní celá čísla, která p rovněž dělí. V číselném případě je řešení jednoduché: p dělí kromě n také $n + p$, $n + 2p$, $n + 3p$ atd. V Erathostenově sítu jsou od pozice v poli, odpovídající číslům dělitelným stejným prvočíslem, rovnoměrně vzdáleny. V polynomiálním případě je situace o něco složitější.

Gordon a McCurley se rozhodli chápat polynomy nad dvouprvkovým tělesem do určitého stupně d jako vrcholy d-rozměrné krychle a pro průchod polem, s cílem nalézt polynomy dělitelné stejným ireducibilním polynomem, použít Grayův kód. Grayův kód je speciální uspořádání binárních slov pevné délky d. Pro číslo $k \in \{1, \ldots, 2^d\}$ označme

$$o(k) = \max(i, 2^i | k),$$

tedy délka nejdelšího souvislého řetězce nul coby přípony binárního zápisu k. Každému číslo v intervalu $1, \ldots, 2^d$ můžeme pomocí Grayova kódování přiřadit binární řetězec délky d následujícím způsobem:

- $G(1) = 000 \ldots 0$
- Známe-li $G(n)$, získáme $G(n+1)$ z $G(n)$ negací $o(n)$-tého bitu (bity číslujeme od 0 do $d - 1$ zleva).

Pomocí výše uvedené definice jsme schopni Grayův kód definovat pro libovolné $d \in N$. Pro $d = 5$ vypadá Grayův kód takto:

k	o(k)	G(k)	k	o(k)	G(k)
1	0	00000	17	0	00011
2	1	10000	18	1	10011
3	0	11000	19	0	11011
4	2	01000	20	2	01011
5	0	01100	21	0	01111
6	1	11100	22	1	11111
7	0	10100	23	0	10111
8	3	00100	24	3	00111
9	0	00110	25	0	00101
10	1	10110	26	1	10101
11	0	11110	27	0	11101
12	2	01110	28	2	01101
13	0	01010	29	0	01001
14	1	11010	30	1	11001
15	0	10010	31	0	10001
16	4	00010	32	5	00001

Pomocí Grayova kódování tedy můžeme zavést vzájemně jednoznačné zobrazení mezi čísly $1, \ldots, 2^d$ a binárními polynomy stupně nejvýše d. Každé dva binární vektory po sobě jdoucí v Grayově kódu se od sebe liší v právě jednom koeficientu. Chápeme-li tedy binární slovo délky d jako souřadnice vrcholů d-rozměrné krychle, vidíme, že existuje cesta po jejích hranách, která projde každým vrcholem právě jednou.

Předpokládejme nyní, že máme g_j ireducibilní polynom, o kterém víme, že dělí v Grayově kódování nejmenšího kandidáta na polynom u_2. Chceme-li nalézt všechny ostatní kandidáty, které polynom g_j rovněž dělí, procházíme polem pomocí přičtení $g_j x^{o(i)}$ v i-tém kroku k původnímu indexu. Po prvních k krocích tak dostáváme polynom

$$g_j + g_j x^{o(2)} + g_j x^{o(3)} + \cdots + g_j x^{o(k)} = g_j(1 + \sum_{i=2}^{k} x^{o(i)})$$

.

Takový polynom je očividně násobkem g_j a $1 + \sum_{i=2}^{k} x^{o(i)}$ je z definice Grayova kódu a z definice očíslování binárních polynomů k-tým nejmenším polynomem. Postupně tak projdeme všechny možné násobky polynomu g_j.

Shrnutí postupu v krocích:

1. Krok 1

   Inicializujme pole A délky 2^t hodnotami 0 pro všechny indexy.

   Pro každý ireducibilní polynom g_j až do stupně b proveďme:

   $q := \max(t - d, 0)$

   Spočítejme

$$u_2 := u_1 x^h \mod g_j.$$

Pro $\deg(u_2) < t$ proveďme navýšení hodnoty $A[u_2]$ o $\deg(u_2)$ a pro $i = 1, \ldots, 2^q$ proveďme:

$$A[u_2] := A[u_2] + \deg(u_2)$$

$$u_2 := u_2 + gx^{o(i)}$$

2. Krok 2

   Dále berme v úvahu pouze taková u_2, pro která platí

$$A[u_2] \geq (\deg(u_1) + h - b).$$

Přínos polynomiálního síta je patrný na první pohled. Většina nevhodných polynomů je sítem odfiltrována a vůbec je není třeba faktorizovat. Operace prováděné uvnitř síta jsou navíc velmi jednoduché a snadno implementovatelné, jedná se především o sčítání a posun indexu pole.

Ani polynomiální síto nám ovšem nedokáže zaručit, že polynom u_2 opravdu povede k použitelné b-hladké kongruenci. Situací, kdy může sítem proklouznout nevhodný polynom, je obecně několik:

1. Algoritmus popsaný výše neřeší situaci, kdy je některý ireducibilní polynom v rozkladu ve vyšší mocnině.

2. Nemáme zajištěno, že nalezená dvojice polynomů u_1 a u_2 je nesoudělná, jak požaduje Coppersmithův algoritmus. Každý polynom u_2, který sítem projde, musíme ještě před ověřením hladkosti w_2 podrobit testu nesoudělnosti s u_1.

6.7 Další zkoumané metody

Index kalkulus je coby obecný princip dále rozvíjen. Kromě funkčního síta, které z myšlenek index kalkulu přímo vychází, jsou stále studovány možnosti optimalizace. Kromě zmíněného prosívání, které filtruje neperspektivní hladké relace dřív, než dojde k faktickému ověření hladkosti, se pro tělesa středních velikostí používá technika takzvaného „vypichování“, poprvé zmíněná v [25].

Oproti klasickému prosívání je asymptoticky rychlejší: zatímco u prosívání je i každému vyřazenému kandidátu na hladkou relaci věnován jistý nezanedbatelný čas, vypichování umožňuje vyřadit kandidáty, aniž by se jimi výpočet musel zabývat.

Další možností, jak dosáhnout u index kalkulu vyšší rychlosti, je paralelizace. Index kalkulus lze paralelizovat snadno; hledání hladkých relací může provádět v jednom okamžiku více vláken současně.

6.8 Implementace

Implementace index kalkulu se nachází v přiložených souboruech `indexCalculus.cpp`, `coppersmith.cpp` a `coppersmithSieving.cpp`. Uvedený zdrojový kód interně používá některé procedury z knihovny `NTL`, zejména `NTL/GF2X.h`, `NTL/ZZ.h`, `NTL/GF2EX.h` a `NTL/vec_ZZ.h` pro práci s polynomy nad dvouprvkovým tělesem, velkými celými čísly a vektory celých čísel.

Samotný kód se skládá z pěti kroků, které jsou implementovány v oddělených procedurách, jejichž hlavičkami začíná samotný kód. Jsou to v obou programech:

1. `preparePrimitivePolynomial()`
2. `prepareField()`
3. `computeSmoothnessBound()`
4. `prepareBase()`
5. `findEquations()`

Krok `preparePrimitivePolynomial()` zajistí primitivní polynom požadovaných vlastností, tedy stupně n tvaru

$$f(x) = x^n + g(x),$$

kde $g(x)$ je polynom co nejmenšího stupně.

Krok `prepareField()` vypočítá na základě dodaného primitivního polynomu velikost tělesa a vygeneruje požadovanou strukturu do proměnné typu `NTL::GF2XModulus`.

Krok `computeSmoothnessBound()` vypočítá mez hladkosti podle vzorce

$$b = \lceil n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \rceil,$$

respektive

$$b = \lceil n^{\frac{1}{3}} (\log n)^{\frac{2}{3}} \rceil.$$

Krok `prepareBase()` zajistí požadovanou množinu bázových prvků, tedy ireducibilních polynomů stupně menšího než je mez hladkosti b z předchozího kroku. K tomu použije předpočítanou tabulku ireducibilních polynomů nad dvouprvkovým tělesem, která je výstupem pomocného programu `irredPolynomialGen.exe`. Předpokládá, že vygenerovaný soubor s polynomy se nazývá `polynomials.txt` a nachází se v adresáři, kde byl program spuštěn.

V kroku `findEquations()` se nachází samotné jádro algoritmu. Dokud není nalezeno dostatečně mnoho lineárních rovnic, hledá program pomocí generátoru náhodných čísel další mocniny generujícího prvku a ověřuje jejich hladkost, respektive pro Coppesmithův algoritmus generuje polynomy w_1 a w_2. Všechny nalezené rovnice jsou zapsány do výstupního souboru `icalc_equations.txt`, respektive `copper_equations.txt` či `copper_sieve_equations.txt`.

Součástí programu je také několik pomocných procedur technického rázu: procedura `to_number` převádí polynom na jeho číselnou reprezentaci v konečném tělese a procedura `is_smooth` ověřuje hladkost daného polynomu. Jak index kalkulus, tak Coppersmithův algoritmus byly psány odděleně; přestože používají podobné procedury a mají podobnou strukturu, nesdílí žádný kód a všechny procedury nutné k překladu jsou v obou zdrojových souborech.

7. Funkční síto

Funkční síto je nejrychlejší dosud známý algoritmus pro výpočet diskrétního logaritmu v cyklické multiplikativní grupě konečného tělesa $\mathrm{GF}(p^n)$ za podmínky:

$$p^6 \leq n.$$

Funkční síto bylo poprvé popsáno v článku Leonarda M. Adlemana Function Field Sieve [1]. Svou podstatou se odvíjí od myšlenky, kterou využívá index kalkulus: nejprve vypočítáme diskrétní logaritmus prvků v jisté pevně definované množině „malých“ prvků ze získaných lineárních vztahů a potom z diskrétního logaritmu „malých“ prvků dopočítáme diskrétní logaritmus libovolného prvku zvoleného.

7.1 Teoretický úvod

Mějme p prvočíslo, n přirozené číslo a x generátor multiplikativní grupy tělesa $\mathrm{GF}(p^n)$. Pro účely algoritmu budeme prvky tohoto tělesa reprezentovat polynomy s koeficienty v tělese $\mathbb{F}_p$ stupně nejvýše n. Minimální polynom prvku x nad $\mathbb{F}_p$ budeme značit f. $\mathrm{GF}(p^n)$ je vektorovým prostorem dimenze n nad $\mathbb{F}_p$ a x je generátorem multiplikativní grupy tělesa, f je tedy primitivní ireducibilní polynom stupně n a násobení v tělese je realizováno jako násobení polynomů modulo polynom f.

Před samotným popisem algoritmu je zapotřebí definovat některé klíčové pojmy z algebraické geometrie a teorie algebraických funkčních těles. Definice hlavních pojmů a značení je převzato z [19] a [20]. Tvrzení, u kterých nebude uveden důkaz, jsou dokázána v [19].

Definice. *Nechť je dáno těleso K, nechť $V \subset A^n(K)$ je podmnožina afinního prostoru dimenze n. Řekneme, že V je* algebraická množina, *pokud existuje $M \subset K[x_1, \ldots, x_n]$ taková, že*

$$V = \{P \in A^n(K) | F(P) = 0 \forall F \in M\}.$$

Definice. *Řekneme, že algebraická množina V je* ireducibilní, *jestliže není sjednocením dvou vlastních algebraických podmnožin.*

Definice. *Afinní varietou rozumíme ireducibilní algebraickou množinu.*

Definice. *Nechť K je těleso, $H \in K[x_1, \ldots, x_n]$, nechť V je afinní varieta definovaná vztahem*

$$V = \{P \in A^n(K) | H(P) = 0\}.$$

Pak V nazveme hyperplochou, *pro $n = 2$ afinní rovinnou algebraickou křivkou.*

Ve zbytku práce budeme pro jednoduchost místo „afinní rovinná algebraická křivka“ psát jen „afinní křivka“, křivky vyšších dimenzí se v práci nevyskytují.

Za pomoci křivky dvou proměnných můžeme definovat pojem funkčního tělesa. V této struktuře se bude odehrávat značná část popisovaného algoritmu.

Definice. *Nechť K je těleso, nechť $H \in K[x_1, x_2]$ je afinní křivka dvou proměnných, nechť $x_1 + (H)$, $x_2 + (H)$ jsou transcendentní prvky nad K. Pak těleso*

$$quot(K[x_1, x_2]/(H))$$

nazveme funkčním tělesem asociovaným křivce H.

Abychom mohli vyložit některé hlubší teoretické výsledky algebraické geometrie, zavedeme ještě definici algebraického funkčního tělesa. Následně ukážeme, že mezi funkčními tělesy asociovanými křivkám a algebraickými funkčními tělesy existuje úzká souvislost.

Definice. *Nechť K je těleso. Pak algebraickým funkčním tělesem jedné proměnné nad K rozumíme rozšíření $K \leq F$ takové, že $[F : K(x)] < \infty$, kde x je transcendentní stupně transcendence 1 nad K.*

V dalších úvahách, nebude-li řečeno jinak, budeme algebraickým funkčním tělesem myslet algebraické funkční těleso jedné proměnné.

Definice. *Nechť F je algebraické funkční těleso nad K. Valuačním okruhem $O \subset F$ budeme rozumět okruh s vlastnostmi:*

1. $K \subsetneq O \subsetneq F$
2. $\forall z \in F, z \neq 0 : z \in O$ *nebo* $z^{-1} \in O$

Abychom mohli definovat další důležité pojmy, uvedeme následující tvrzení:

Věta 5. *Nechť F je algebraické funkční těleso nad K, nechť O je valuační okruh. Pak $P := O \setminus O^*$ je hlavní maximální ideál. Je-li $t \in P$ generujícím prvkem P, lze každý $0 \neq z \in F$ psát jednoznačně ve tvaru $z = u \cdot t^n$, kde $u \in O^*$ a $n \in \mathbb{Z}$.*

Předchozí věta nás opravňuje k definici:

Definice. *Nechť F je algebraické funkční těleso nad K, nechť O je valuační okruh. Pak ideál $P := O \setminus O^*$ nazveme místo a každý jeho generující prvek t nazveme uniformizujícím prvkem. Množinu všech míst tělesa F označme $\mathbb{P}_F$.*

Platí, že mezi místy a valuačními okruhy v tělese F existuje vzájemně jednoznačná korespondence: v definici je místo P jednoznačně zkonstruováno za pomoci nějakého valuačního okruhu O a opačně platí, že bylo-li místo P zkonstruováno pomocí valuačního okruhu O, lze O z P zpětně zkonstruovat jako

$$O_P = \{z \in F | z^{-1} \notin P\}.$$

Definice. *Nechť F je algebraické funkční těleso nad K. Diskrétní valuací F nad K rozumíme funkci $v : F \to \mathbb{Z} \cup \{\infty\}$, pro kterou platí:*

1. $v(z) = \infty \Leftrightarrow z = 0$
2. $v(x \cdot y) = v(x) + v(y)$
3. $v(x + y) \geq \min\{v(x), v(y)\}$
4. $\exists z \in F : v(z) = 1$

5. $v(a) = 0$ *pro* $a \in K \setminus \{0\}$.

Věta 6. *Nechť F je algebraické funkční těleso nad K, nechť P je místo s uniformizujícím prvkem t. Pak zobrazení $v_P : F \to \mathbb{Z}$ definované předpisem*

$$v_P(z) = n \text{ pro } z = u \cdot t^n, u \in O_P^*, n \in \mathbb{Z}$$

je diskrétní valuací tělesa F nad K. v_P navíc nezávisí na volbě uniformizujícího prvku.

Mezi pojmy místo a diskrétní valuace opět existuje vzájemně jednoznačné zobrazení. Máme-li v diskrétní valuaci F nad K, je množina

$$P_v = \{z \in F | v(z) > 0\}$$

místem a množina

$$O_v = \{z \in F | v(z) \geq 0\}$$

valuačním okruhem tělesa F. Pojmy valuační okruh, místo a diskrétní valuace jsou tedy úzce spjaty.

Definice. *Nechť F je algebraické funkční těleso nad K, nechť P je místo příslušné valuačnímu okruhu O_P. Těleso O_P/P značíme F_P a stupněm místa P rozumíme stupeň rozšíření F_P nad K, tedy $[F_P : K]$.*

F_P je dobře definované těleso, neboť P je maximální ideál okruhu O_P. Stupeň místa je navíc vždy konečný.

Definice. *Řekneme, že $P \in \mathbb{P}_F$ je nulou $z \in F$, jestliže $v_P(z) > 0$. Obdobně řekneme, že P je pólem z, jestliže $v_P(z) < 0$.*

Věta 7. *Nechť $0 \neq z \in F$, kde F je algebraické funkční těleso. Pak*

$$\sum_{P \in \mathbb{P}_F} v_P(z) \deg(P) = 0.$$

Dalším pojmem, který je nezbytné zadefinovat pro výklad algoritmu funkčního síta, je pojem divisor:

Definice. *Mějme K těleso algebraicky uzavřené v algebraickém funkčním tělese F. Nechť $\mathbb{P}_F$ je množina všech míst F. Pak volnou abelovskou grupu generovanou $\mathbb{P}_F$ značíme D_F a nazýváme ji grupou divisorů F. Budeme značit divisor*

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P,$$

kde $n_P = 0$ až na konečně mnoho míst P.

Definice. *Nechť $D \in D_F$ je divisor. Řekneme, že D je prvodivisor, jestliže existuje $P \in \mathbb{P}_F$ takové, že $D = P$. Řekneme, že D je hlavním divisorem prvku $z \in F$, jestliže*

$$D := \mathrm{Div}(z) = \sum_{P \in \mathbb{P}_F} v_P(z) \cdot P.$$

Věta 8. *Nechť pro $z_1, z_2 \in F$ platí, že*

$$\mathrm{Div}(z_1) = \mathrm{Div}(z_2).$$

Pak existuje $k \in K$, že $z_1 = kz_2$.

Pojem stupně místa lze přirozeným způsobem rozšířit na stupeň divisoru:

Definice. *Nechť $D \in D_F$ je divisor,*

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P,$$

kde $n_P \in \mathbb{Z} \forall P \in \mathbb{P}_F$. Pak definujeme stupeň D jako

$$\deg(D) = \sum_{P \in \mathbb{P}_F} n_P \cdot \deg(P).$$

Z definice vyplývá, že zobrazení $\deg : D_F \to \mathbb{Z}$ je grupový homomorfismus, tedy

$$\deg(D_1 + D_2) = \deg(D_1) + \deg(D_2).$$

Věta 9. *Stupeň každého hlavního divisoru je roven* 0.

Opačné tvrzení ovšem neplatí; divisor stupně 0 nemusí být hlavní. Množinu všech divisorů stupně 0 budeme značit D_F^0.

Definice. *Grupou hlavních divisorů budeme rozumět*

$$\underset{F}{\mathrm{Div}} := \{\mathrm{Div}(z) | 0 \neq z \in F\}.$$

Definice je korektní, hlavní divisory skutečně tvoří podgrupu grupy všech divisorů, neboť

$$\mathrm{Div}(xy) = \mathrm{Div}(x) + \mathrm{Div}(y).$$

Definice. *Třídovým číslem tělesa F budeme rozumět velikost grupy D_F^0 / Div_F a budeme ho značit symbolem h.*

Somberg v lemmatu 106 v [19] dokazuje, že h je nad konečným tělesem, tedy v případě, který je důležitý pro funkční síto, vždy konečné.

Věta 10. *Nechť F nad K je algebraické funkční těleso s konečným třídovým číslem, D je divisor stupně* 0*. Pak hD je hlavní divisor.*

Důkaz. h je velikost grupy D_F^0 / Div_F, uvážíme-li tedy projekci D do D_F^0 / Div_F, jistě $hD \in \mathrm{Div}_F$. □

Z poslední věty je patrné, že je-li třídové číslo rovno 1, jsou divisory stupně 0 právě všechny hlavní divisory.

Definice. *Nechť K je těleso, nechť E je jeho rozšíření konečného stupně. Pak normou tělesa E nad K rozumíme zobrazení $N : E \to K$ definované vztahem:*

$$N(a) = \det(L_a),$$

kde se symbolem L_a rozumí matice zobrazení

$$L_a : E \to E$$

$$L_a(b) = a \cdot b$$

coby automorfismu vektorového prostoru E nad tělesem K.

Ve funkčním sítu hrají důležitou úlohu rozšíření funkčních těles:

Definice. *Nechť F je algebraické funkční těleso nad tělesem K, nechť $F \leq E$, $K \leq L \leq E$, kde L je algebraické rozšíření K. Pak algebraické funkční těleso E nad L nazveme rozšířením tělesa F nad K.*

Definice. *Nechť E nad L je rozšířením F nad K, nechť $P' \in \mathbb{P}_E$, $P \in \mathbb{P}_F$. Řekneme, že P' leží nad P, jestliže $P = P' \cap F$, značíme $P'|P$.*

Lze ukázat a v [27] je dokázáno, že mezi místy v rozšíření a v původním tělese existuje jistý vztah:

Věta 11. *Nechť E nad L je rozšířením F nad K, $P \in \mathbb{P}_F$. Pak existuje konečně mnoho míst $Q \in \mathbb{P}_E$ ležících nad P. Navíc pro P a Q existuje $e(Q|P) \in \mathbb{N}$ takové, že pro každé $x \in F$ platí*

$$v_Q(x) = e(Q|P)v_P(x).$$

Hodnotu $e(Q|P)$ z předchozí věty nazveme větvicí index Q nad P. Stupeň $[E_Q : F_P]$ označme $f(Q|P)$. Dále lze ukázat [27], že každé místo v rozšíření leží nad nějakým místem základního tělesa. To nás opravňuje k definici normy místa z rozšíření nad základním tělesa:

Definice. *Nechť E nad L je rozšířením F nad K, nechť $Q \in \mathbb{P}_E$, $P = Q \cap F$. Pak zobrazení $N : \mathbb{P}_E \to \mathrm{Div}_F$ definované předpisem*

$$N(Q) = f(Q|P)P$$

nazveme normou Q nad F. Toto zobrazení můžeme lineárně rozšířit na zobrazení $N : \mathrm{Div}_E \to \mathrm{Div}_F$.

Pro funkční síto klíčové tvrzení, dokázané na straně 75 v [30], dává do souvislosti klasickou normu prvku tělesa s normou divisoru:

Věta 12. *Buď $x \in E$, kde E nad L je rozšířením F nad K. Pak*

$$N(\operatorname*{Div}_E(x)) = \operatorname*{Div}_F(N(x)).$$

Z posledního tvrzení pak plyne následující pozorování: všechna místa, v nichž má prvek x nad E nulu nebo pól, leží nad místem, ve kterém má $N(x)$ nad F nulu nebo pól. V důkazu této věty, uvedeném v [30], je navíc dokázáno, že

$$v_P(N(x)) = \sum_{Q|P} f(Q|P) v_Q(x),$$

což bude dále využito.

Závěrečné tvrzení pokládá rovnost mezi pojmy algebraické funkční těleso a funkční těleso asociované křivce.

Věta 13. *Nechť F nad K je algebraické funkční těleso. Pak existuje algebraická křivka H taková, že F je funkčním tělesem asociovaným křivce H. Opačně pro každou křivku H je jí asociované funkční těleso algebraickým funkčním tělesem.*

V jistých situacích v popisu algoritmu afinní křivka nestačí a je nezbytné pracovat s projektivní křivkou, ve které je afinní křivka obsažena. V závěru této části proto zadefinujeme vnoření afinní křivky do projektivní křivky a zmíníme vztah projektivní křivky k původní afinní křivce.

Definice. *Nechť K je těleso. Projektivním prostorem dimenze n rozumíme množinu $n+1$-tic $(x_0 : \ldots : x_n) \in K^{n+1} \setminus (0 : \ldots : 0)$, v níž prvky $(x_0 : \ldots : x_n)$ a $(y_0 : \ldots : y_n)$ považujeme za shodné, pokud existuje prvek $0 \neq \lambda \in K$, pro který platí*

$$(x_0 : \ldots : x_n) = (\lambda y_0 : \ldots : \lambda y_n).$$

Definice. *Nechť K je těleso, C je algebraická křivka definovaná polynomem $H \in K[x, y]$. Položme $\overline{H}(x, y, z) := z^{\deg H} H(\frac{x}{z}, \frac{y}{z})$. Projektivní křivku odpovídající afinní křivce C značíme $\overline{C}$ a rozumíme jí podmnožinu bodů projektivního prostoru nad tělesem K definovanou pomocí polynomu $\overline{H}$ takto:*

$$\overline{C} = \{(x : y : z) | \overline{H}(x, y, z) = 0\}.$$

Řekneme, že bod $(x : y : z)$ na projektivní křivce $\overline{C}$ je bodem v nekonečnu afinní křivky C, jestliže $z = 0$.

Obecně platí, že je-li (x, y) bodem afinní křivky C, je $(x : y : 1)$ bodem projektivní křivky $\overline{C}$. Afinní křivku lze tedy přirozeně chápat jako část příslušné křivky projektivní. Podrobnější představu o afinních a projektivních křivkách a jejich vztahu lze získat například z [30].

Jedním z podpůrných algoritmů funkčního síta je metoda založená na Newtonově mnohoúhelníku. Newtonova metoda pracuje s nekonečnými součty prvků z algebraického funkčního tělesa. Ty nemusí mít v tělese samotném smysl, je zapotřebí definovat větší strukturu, takzvané P-adické zúplnění. Důkazy uvedených tvrzení lze nalézt v [20].

Definice. *Buď T těleso s diskrétní valuací v. $\{x_n\}_{n \geq 0}$ buď posloupnost prvků z T. Řekneme, že posloupnost $\{x_n\}_{n \geq 0}$ je konvergentní ve valuaci v, jestliže existuje $x \in T$, které nazýváme limita, splňující:*

$$\forall c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : v(x - x_n) \geq c.$$

Definice. *Buď T těleso s diskrétní valuací v. $\{x_n\}_{n\geq 0}$ buď posloupnost prvků z T. Řekneme, že posloupnost $\{x_n\}_{n\geq 0}$ je cauchyovská ve valuaci v, jestliže*

$$\forall c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall m, n \geq n_0 : v(x_m - x_n) \geq c.$$

Definice. *Těleso T s diskrétní valuací v je úplné, pokud je každá cauchyovská posloupnost prvků z T konvergentní ve valuaci v. Zúplněním tělesa T s valuací v rozumíme těleso $\hat{T}$ s valuací $\hat{v}$ splňující:*

- *$T \subset \hat{T}$, $\hat{v}|_T = v$*
- *$\hat{T}$ s diskrétní valuací $\hat{v}$ je úplné*
- *T je husté v $\hat{T}$, každý prvek $\hat{T}$ je tedy limitou nějaké posloupnosti prvků ležících v T.*

Věta 14. *Buď T těleso s diskrétní valuací v. Pak existuje jeho zúplnění $\hat{T}$ s diskrétní valuací $\hat{v}$, které je jednoznačné ve smyslu: je-li $\tilde{T}$ s diskrétní valuací $\tilde{v}$ jiné zúplnění, pak existuje právě jedno $f : \hat{T} \to \tilde{T}$, pro které platí*

$$\tilde{v} \circ f = \hat{v}.$$

Definice. *Nechť F nad K je algebraické funkční těleso, $P \in \mathbb{P}_F$ je jeho místo s odpovídající diskrétní valuací v_P. Zúplnění tělesa F s valuací v_P pak nazýváme P-adické zúplnění a značíme ho $\hat{F}_P$.*

Věta 15. *Nechť F nad K je algebraické funkční těleso, $P \in \mathbb{P}_F$ je jeho místo s odpovídající diskrétní valuací v_P a uniformizujícím prvkem t, $\deg P = 1$, $z \in \hat{F}_P$. Pak lze z jednoznačně psát jako*

$$z = \sum_{i=n}^{\infty} a_i t^i,$$

kde $a_i \in F$ pro $i = n, n+1, \ldots$, přičemž $n \in \mathbb{Z}$.

7.2 Popis algoritmu

7.2.1 Dvojnásobně hladké páry

Předpokládejme p, n, x a f jako výše. Polynom $m \in \mathbb{F}_p[x]$ definujme jako monický polynom stupně d_1, kde d_1 je pevně stanovená konstanta, jejíž přesnou hodnotu určíme později.

Dále mějme absolutně ireducibilní polynom dvou proměnných $H \in \mathbb{F}_p[x, y]$ stupně d v y splňující

$$H(x, m) = 0 \pmod{f}.$$

Z vlastnosti, kterou pro polynom H požadujeme, dostáváme, že zobrazení

$$\psi : \mathbb{F}_p[x, y]/(H) \to \mathbb{F}_p[x]/(f)$$

definované předpisem $\psi(y) = m + (f)$ je surjektivní homomorfismus. Této vlastnosti v budoucnu využijeme, její důkaz plyne přímo z definice ψ. Pro jednoduchost budeme dále prvky $\mathbb{F}_p[x]/(f)$ a $\mathbb{F}_p[x,y]/(H)$ zapisovat pouze pomocí reprezentantů určité třídy, například místo $w(x) + (f) \in \mathbb{F}_p[x]/(f)$ tak budeme psát pouze $w(x)$.

Nyní zafixujeme mez hladkosti b, její přesnou hodnotu určíme později při analýze časové složitosti. Naším cílem v první fázi algoritmu je spočítat diskrétní logaritmus všech ireducibilních polynomů v množině

$$B := \{g \in \mathbb{F}_p[x] | g \text{ ireducibilní}, \deg g < b\}.$$

Jelikož bude mít mez hladkosti b oproti algoritmu index kalkulus výrazně nižší hodnotu, bude zapotřebí komplikovanějších algebraických struktur, abychom získali potřebné lineární vztahy mezi diskrétními logaritmy „malých“ prvků z množiny B.

Jeden takový lineární vztah získáme z dvojice polynomů r, $s \in \mathbb{F}_p[x]$, $r \neq 0 \neq s$ jistých vlastností:

- $rm + s$ musí být b-hladký,
- norma $ry + s$ vzhledem k tělesu $quot(\mathbb{F}_p[x,y]/(H))$ nad tělesem $\mathbb{F}_p(x)$, tedy $r^d H(x, -\frac{s}{r})$, kde d je stupeň H v y (odůvodnění vztahu viz níže), musí být b-hladká.

Splňuje-li dvojice (r, s) obě výše uvedené podmínky, nazveme ji dvojnásobně hladkým párem. Každý dvojnásobně hladký pár uvádí na scénu dva objekty: $rm+s \in \mathbb{F}_p[x]$ a $ry+s \in \mathbb{F}_p[x,y]$. Ty jsou spjaty homomorfismem ψ, definovaným v textu výše:

$$\psi(ry + s) = rm + s.$$

7.2.2 Výpočet normy $ry + s$

V předchozím odstavci bylo pro ověření dvojnásobné hladkosti zapotřebí vypočítat normu pevně zvoleného prvku $ry + s$ v tělese $quot(\mathbb{F}_p[x,y]/(H))$ nad tělesem $\mathbb{F}_p(x)$.

Předpokládejme, že H je tvaru

$$H(x,y) = a_0(x) + a_1(x)y + a_2(x)y^2 + \cdots + y^d$$

a uvažme akce prvku $ry + s$ na bázových vektorech $1, y, y^2, \ldots, y^{d-1}$:

- $L_{ry+s}(1) = ry + s$
- $L_{ry+s}(y) = ry^2 + sy$
- $L_{ry+s}(y^2) = ry^3 + sy^2$
- $\cdots$
- $L_{ry+s}(y^{d-1}) = ry^d + sy^{d-1} = r(-a_0(x) - a_1(x)y - \cdots - a_{d-1}(x)y^{d-1}) + sy^{d-1}$

Hledaná matice přechodu vzhledem k bázi $\{1, y, \ldots, y^{d-1}\}$ je tedy tvaru:

$$L_{ry+s} = \begin{pmatrix} s & 0 & 0 & \cdots & -ra_0(x) \\ r & s & 0 & \cdots & -ra_1(x) \\ 0 & r & s & \cdots & -ra_2(x) \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & r & s - ra_{d-1}(x) \end{pmatrix}$$

Symbolem D_i budeme pro $0 \leq i < d$ v následujícím výpočtu rozumět determinant matice:

$$\begin{pmatrix} s & 0 & 0 & \cdots & -ra_i(x) \\ r & s & 0 & \cdots & -ra_{i+1}(x) \\ 0 & r & s & \cdots & -ra_{i+2}(x) \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & r & s - ra_{d-1}(x) \end{pmatrix}$$

Rozvojem podle prvního sloupce dostáváme pro $d - i > 1$ vztah:

$$D_i = s \cdot D_{i+1} + (-r)^d a_i(x).$$

Hledaný determinant tedy můžeme počítat pomocí uvedeného vztahu:

$$\begin{aligned} \det(L_{ry+s}) = D_0 &= s \cdot \begin{vmatrix} s & 0 & \cdots & -ra_1(x) \\ r & s & \cdots & -ra_2(x) \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & r & s - ra_{d-1}(x) \end{vmatrix} + (-1)^d r \cdot \begin{vmatrix} 0 & 0 & \cdots & -ra_0(x) \\ r & s & \cdots & -ra_2(x) \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & r & s - ra_{d-1}(x) \end{vmatrix} \\ &= s \cdot D_1 + (-r)^d a_0(x) \\ &= s(s \cdot D_2 + (-r)^d a_1(x)) + (-r)^d a_0(x) \\ &= \cdots \\ &= s^d + (-r)^d \textstyle\sum_{i=0}^{d-1} (\frac{s}{r})^i a_i(x) \\ &= (-r)^d H(x, -\tfrac{s}{r}) \end{aligned}$$

7.2.3 Konstrukce polynomu H

Doposud jsme předpokládali existenci polynomu H požadovaných vlastností. Nyní uvažme, jak takové H exaktně nalézt. Máme polynom f stupně n a monický polynom m stupně d_1, kde d_1 můžeme volit. Víme, že f je ireducibilní primitivní polynom, a můžeme požadovat, aby byl f tvaru

$$f(x) = x^n + w(x),$$

kde $w(x)$ je polynom co nejmenšího stupně volený tak, aby f byl primitivní, tedy aby prvek $x + (f)$ byl primitivním prvkem tělesa $\mathbb{F}_p[x]/(f)$.

Pokusíme se nalézt H stupně d takového, že $d_1 = \lceil n/d \rceil$. Pak

$$dd_1 = n + \delta$$

pro $\delta < d$. Jistě potom existují polynomy $a_0, a_1, \ldots, a_{d-1} \in \mathbb{F}_p[x]$ stupně menšího než d_1 splňující:

$$x^\delta f = m^d + a_{d-1} m^{d-1} + \cdots + a_0.$$

Tento rozklad polynomu $x^\delta f$ lze chápat jako jeho zápis v bázi mocnin m. Polynom H pak můžeme definovat jako jeden ze vzorů $x^\delta f$ v homomorfismu ψ:

$$H(x,y) := y^d + a_{d-1}y^{d-1} + \cdots + a_0.$$

Pro takto definovaný polynom jistě platí

$$H(x,m) = m^d + a_{d-1}m^{d-1} + \cdots + a_0 = x^\delta f = 0 \pmod{f}.$$

Adleman v [1] uvádí další podmínky pro volbu H:

1. H musí být absolutně ireducibilní (tedy ireducibilní nad algebraickým uzávěrem $\mathbb{F}_p$).
2. třídové číslo h podílového tělesa $\mathbb{F}_p[x,y]/(H)$ musí být nesoudělné s $\frac{p^n-1}{p-1}$.

Splnění první podmínky můžeme snadno dosáhnout vhodnou volbou d_1 a w [26]. Nejprve předpokládejme, že jsme zvolili w tak, aby v měl v algebraickém uzávěru $\mathbb{F}_p$ alespoň jeden kořen násobnosti 1. Tento kořen γ je zřejmě nenulový, jinak by f nebyl ireducibilní polynom. Zvolíme-li dále d_1 tak, aby

$$\deg(x^\delta w(x)) < d_1,$$

polynom m jako x^{d_1}, dostáváme H jistě tvaru

$$H(x,y) = y^d + x^\delta w(x),$$

polynom H má pak coby polynom nad $\mathbb{F}_p[x]$ pouze dva nenulové členy. Nyní můžeme uplatnit Eisensteinovo kritérium ireducibility polynomu nad obecným oborem integrity (dokázané například v [28]), v našem případě nad $\mathbb{F}_p[x]$:

Věta 16. *Buď D obor integrity, nechť*

$$Q = \sum_{i=0}^{n} a_i x^i \in D[x].$$

Dále nechť P je prvoideál oboru D takový, že platí:

- $a_i \in P$ *pro* $i = 0, \ldots, n-1$,
- $a_n \notin P$,
- $a_0 \notin P^2$.

Pak Q nelze rozložit na součin dvou nekonstantních polynomů v $D[x]$.

Jelikož platí

- $x^\delta w(x) = x^\delta (x-\gamma)\overline{w}(x) \in \langle (x-\gamma) \rangle$,
- $1 \notin \langle (x-\gamma) \rangle$,
- $x^\delta w(x) \notin \langle (x-\gamma)^2 \rangle$, protože γ je jednoduchý kořen $w(x)$,

můžeme Eisensteinovo kritérium uplatnit a takto zvolené H je jistě absolutně ireducibilní polynom.

V případě, že některá z podmínek není splněna, funkční síto neproběhne korektně a výsledkem soustavy lineárních rovnic nemusí být hledané logaritmy „malých“ prvků. Druhou podmínku navíc neumíme ověřit v polynomiálním čase, žádný efektivní algoritmus pro výpočet třídového čísla není znám. V případě, že druhá podmínka není zajištěna, zjistí algoritmus selhání až při ověření výsledků a musí začít od začátku s jinou volbou H.

7.2.4 Náhražková funkce

Předpokládejme, že H definuje nesingulární křivku C. Na ψ pak lze nahlížet jako na homomorfismus okruhu $\mathbb{F}_p[x,y]/(H)$ do $\mathbb{F}_p[x]/(f)$. Symbolem $K(C)$ budeme značit funkční těleso nad $\mathbb{F}_p$ asociované křivce C .

Uvažme místo $P \in \mathbb{P}_{K(C)}$, O_P buď jeho valuační okruh, f_P stupeň P. Mějme Q místo splňující $f_Q = 1$. Symbolem h označme třídové číslo tělesa $K(C)$. Pro libovolné místo P pak platí, že $P - f_P Q$ je divisor stupně 0. Každému P tak můžeme přiřadit nějaký prvek $\alpha_P \in K(C)$ takový, že

$$\mathrm{Div}(\alpha_P) = h(P - f_P Q).$$

Prvek α_P je určen jednoznačně až na násobek konstantou z $\mathbb{F}_p$. Pro účely další definice prvek α_P fixujme jako jeden libovolný, pevně zvolený z těch, které jsou k dispozici. Divisor $h(P - f_P Q)$ je zcela jistě hlavní podle výše uvedené věty o třídovém číslu.

Předchozího faktu využijeme k definici náhražkové funkce α:

$$\alpha(P) := \alpha_P \text{ pro všechna } P \in \mathbb{P}_{K(C)}.$$

Díky fixaci prvků α_P je α korektně definovanou funkcí.

V implementaci samotné je třeba místo $\mathbb{F}_p$ uvážit jeho algebraický uzávěr $U = \overline{\mathbb{F}_p}$. Pro výpočet důležité jsou však pouze divisory a místa, která jsou rozšířením míst a divisorů z původního tělesa.

7.2.5 Morfismus ψ na hodnotách náhražkové funkce α

Hodnoty náhražkové funkce jsou obecné hodnoty z funkčního tělesa $K(C)$, do kterého můžeme přirozeně vnořit okruh $\mathbb{F}_p[x,y]/(H)$. Na $\mathbb{F}_p[x,y]/(H)$ je definován homomorfismus ψ. Pro další kroky algoritmu potřebujeme zajistit, aby byl homomorfismus ψ korektně definován pro všechny hodnoty náhražkové funkce. Ukážeme proto, že můžeme definici homomorfismu ψ korektně rozšířit na větší strukturu.

Jádrem homomorfismu ψ je prvoideál generovaný prvky f a $y - m$. Doplněk tohoto prvoideálu v $\mathbb{F}_p[x,y]/(H)$ označme R,

$$R := (\mathbb{F}_p[x,y]/(H)) \setminus \ker(\psi).$$

Mějme $G \in \mathbb{F}_p[x,y]$ obraz prvku $\overline{G} \in \mathbb{F}_p[x,y]/(H)$, kde

$$\overline{G} \notin \ker(\psi),$$

tedy prvku z R, který může zastávat roli jmenovatele v lokalizaci

$$A_\psi := R^{-1}\mathbb{F}_p[x,y]/(H).$$

Ideál $\ker(\psi)$ je prvoideál, vzniklá lokalizace A_ψ je tak lokálním okruhem. A_ψ proto obsahuje jediný maximální ideál, konkrétně maximální ideál vzniklý rozšířením $\ker(\psi)$, tedy

$$E_\psi := R^{-1}\ker(\psi).$$

Abychom rozšířili definici ψ na A_ψ, stačí ψ korektně definovat pro všechna $\frac{1}{\overline{G}}$.

Jelikož $\overline{G} \notin \ker(\psi)$, je G tvaru

$$G(x,y) = G_1(x,y)(y-m) + a(x),$$

kde $G_1(x,y) \in \mathbb{F}_p[x,y]$ a $a(x) \in \mathbb{F}_p[x]$ není dělitelné f. Prvek a je nenulovým prvkem tělesa $\mathbb{F}_p/(f)$, existuje tedy jeho inverze. Můžeme proto ψ korektně rozšířit na A_ψ definicí

$$\psi(\frac{1}{\overline{G}}) := a^{-1} \pmod f.$$

To, zda se budou všechny hodnoty náhražkové funkce vyskytovat v A_ψ, závisí na volbě H. Podaří-li se nám zvolit Q z definice náhražkové funkce a polynom H „šikovně“, bude ψ definováno pro všechny hodnoty náhražkové funkce. Ukážeme, že stačí zvolit Q vhodně a H k němu tak, aby byl Q jednoduchým bodem křivky C, tedy s koeficienty, které jsou kořenem polynomu H násobnosti 1. Za této volby bude A_ψ valuačním okruhem s diskrétní valuací, kterou budeme značit v_ψ. Na A_ψ je již homomorismus ψ definován. Stačí tedy dokázat, že pro všechna $P \in \mathbb{P}_{K(C)}$ bude platit $v_\psi(\alpha_P) \geq 0$. Pak jistě $\alpha_P \in A_\psi$.

Algebraický uzávěr tělesa $\mathbb{F}_p$ označme U. Nechť γ značí kořen f, $f(\gamma) = 0$, $\gamma \in U$. Pak můžeme volit místo $\overline{Q} = (\gamma, m(\gamma))$ tak, aby $Q = \overline{Q} \cap K(C)$ byl jednoduchý bod C nad K. Takový bod $\overline{Q}$ je jistě bodem na křivce C nad U, protože

$$H(\gamma, m(\gamma)) = k(\gamma)f(\gamma) = k(\gamma) \cdot 0 = 0.$$

$O_{\overline{Q}}$ buď lokální okruh bodu $\overline{Q}$ na křivce C. Z volby souřadnic bodu $\overline{Q}$ plyne, že $R \subset \overline{Q}$, takže $O_{\overline{Q}}$ pokrývá A_ψ. Protože je $\overline{Q}$ navíc volen jako jednoduchý bod, je $O_{\overline{Q}}$ valuační okruh [27]. A_ψ je potom nutně také valuační okruh, přičemž jeho místo je už známý maximální ideál E_ψ.

Dále víme, že hodnota náhražkové funkce α_P má z definice hlavní divisor tvaru $h(P - f_P Q)$. Jeho jediný pól se tedy nachází v místě Q, ve všech ostatních místech póly α_P nejsou. Bude stačit ukázat, že $E_\psi \neq Q$. Pak jistě $v_\psi(\alpha_P) \geq 0$, a tedy $\alpha_P \in A_\psi$, nezávisle na P.

Že se místa E_ψ a Q nerovnají, ukážeme porovnáním stupňů. Víme, že Q je jednoduchý bod, takže $f_Q = 1$. Naproti tomu ovšem $f_{E_\psi} = \deg(f) > 1$.

Díky vhodné volbě Q a H je tak $\psi(\alpha_u)$ definováno pro všechna místa P tělesa $K(C)$.

Na tomto místě je nutné poznamenat, že význam náhražkové funkce je v algoritmu funkčního síta důležitý pouze z teoretického existenčního hlediska, pro formální ospravedlnění dalších úvah. Náhražková funkce není v průběhu algoritmu nikdy explicitně počítána.

7.2.6 Částečný diskrétní logaritmus

Na prvcích $\alpha \in \mathrm{GF}(p^n)$ nyní definujeme pomocné zobrazení, kterému budeme říkat částečný diskrétní logaritmus. Částečný diskrétní logaritmus budeme značit $\log_R \alpha$ a definovat ze vztahu

$$x^{\log_R \alpha} = u\alpha$$

pro nějaké $u \in \mathbb{F}_p, u \neq 0$. Aby bylo zobrazení jednoznačné, vybereme z $p-1$ možností, které splňují výše uvedenou rovnost, tu nejmenší, tedy ležící v intervalu

$$[0, (p^n - 1)/(p - 1)).$$

Dále v textu ukážeme, že k výpočtu diskrétního logaritmu daného prvku stačí spočítat částečný diskrétní logaritmus.

Cílem dalšího kroku bude pomocí dvojnásobně hladkých párů nalézt lineární rovnice, jejichž neznámými budou částečné diskrétní logaritmy „malých“ prvků z množiny B. Nalezneme-li rovnic dostatečně mnoho, budeme schopni jednoznačně určit hodnoty jednotlivých částečných diskrétních logaritmů.

7.2.7 Hladké kongruence

Definice. *Nechť R je okruh, $c \in R$. Pak pro $d \in R$ definujeme*

$$O_c(d) := \max\{i \in \mathbb{N}_0 : c^i | d\}.$$

Lze ukázat, že všechny diskrétní valuace racionálního funkčního tělesa $\mathbb{F}_p(x)$ jsou pro $f, h \in \mathbb{F}_p[x]$ právě tvaru

$$v_g(\frac{f}{h}) := O_g(f) - O_g(h),$$

kde g je ireducibilní, nebo tvaru

$$v_\infty(\frac{f}{g}) := \deg(g) - \deg(f).$$

Tvrzení je detailně vysvětleno například v [19].

Uvažme prvek $g_i \in B \subset \mathbb{F}_p[x]$. g_i je ireducibilní polynom, zobrazení v_{g_i} definované v předchozím odstavci je tedy jednou z diskrétních valuací $\mathbb{F}_p(x)$ nad $\mathbb{F}_p$.

Racionální funkční těleso $\mathbb{F}_p(x)$ je podtělesem funkčního tělesa $quot(\mathbb{F}_p[x,y]/(H))$. Každá diskrétní valuace $\mathbb{F}_p(x)$ nad $\mathbb{F}_p$ se v nadtělese rozkládá do několika diskrétních valuací $quot(\mathbb{F}_p[x,y]/(H)) = K(C)$ nad $\mathbb{F}_p$. V průběhu algoritmu budeme pro různé vstupy počítat hodnoty diskrétních valuací $K(C)$ nad $\mathbb{F}_p$.

Nechť je dána množina

$$B_1 := \{v_1, v_2, \ldots, v_z\},$$

kde v_i jsou diskrétní valuace tělesa $K(C)$ nad $\mathbb{F}_p$, do kterých se rozkládají diskrétní valuace tělesa $\mathbb{F}_p(x)$ nad $\mathbb{F}_p$ odpovídající prvkům z B. α_i nechť jsou obrazy náhražkové funkce míst P_i příslušných diskrétních valuací v_i. Mějme prvek $w \in K(C)$ a dvojice r, s nechť tvoří dvojnásobně hladký pár a platí, že $\deg(r) \leq W$, $\deg(s) \leq W$, kde W je konstanta, kterou určíme později.

Jednou z podmínek kladených na dvojnásobně hladký pár (r, s) je b-hladkost normy prvku $ry + s \in K(C)$ nad tělesem $\mathbb{F}_p(x)$. Norma

$$N(ry + s) = (-r)^d H(x, -\frac{s}{r})$$

se tedy rozkládá do prvků bázové množiny B. Odtud potom vyplývá, že se všechny nuly i póly $ry + s$ nacházejí v místech $K(C)$, která jsou rozšířením míst prvků z množiny B v tělese $\mathbb{F}_p(x)$. Všechny ostatní diskrétní valuace tělesa $K(C)$, tedy ty, které nejsou rozšířením diskrétní valuace $\mathbb{F}_p(x)$ žádného bazového prvku g_i, jsou na prvku $ry + s$ nulové.

Můžeme proto psát:

$$\mathrm{Div}(ry + s) = \sum_{i=1}^{z} a_i P_i,$$

kde $a_i = v_i(ry + s)$.

Odtud počítejme:

$$\mathrm{Div}((ry + s)^h) = \sum_{i=1}^{z} h a_i P_i = \sum_{i=1}^{z} a_i h(P_i - \deg(P_i)Q) = \mathrm{Div}(\prod_{i=1}^{z} {\alpha_i}^{a_i}).$$

Dostáváme tedy rovnost dvou hlavních divisorů. Z věty o hlavních divisorech vyplývá, že se prvky liší maximálně přenásobením konstantou ze základního tělesa:

$$(ry + s)^h = c \prod_{i=1}^{z} {\alpha_i}^{a_i},$$

kde c je nenulový prvek $\mathbb{F}_p$.

Z dvojnásobné hladkosti páru (r, s) víme, že polynom $rm+s$ lze nad $\mathbb{F}_p$ rozložit na ireducibilní faktory z množiny B. Tedy

$$rm + s = \prod_{g \in B} g^{e_g}.$$

Nyní na obě strany rovnosti výše aplikujeme zobrazení ψ a dostáváme:

$$\psi((ry+s)^h) = \psi(c)\prod_{i=1}^{z}\psi(\alpha_i)^{a_i},$$

tedy

$$\prod_{g\in B} g^{he_g} \equiv c\prod_{i=1}^{z}\psi(\alpha_i)^{a_i}.$$

Použijeme-li na poslední rovnost částečný diskrétní logaritmus, dostáváme

$$\sum_{g\in B} he_g \log_R g \equiv \sum_{i=1}^{z} a_i \log_R(\psi(\alpha_i)) \pmod{\frac{p^n-1}{p-1}}.$$

Za předpokladu, že h je nesoudělné s $(p^n-1)/(p-1)$, můžeme nalézt jeho multiplikativní inverzi v $\mathbb{Z}_{(p^n-1)/(p-1)}$ a upravit rovnici do tvaru

$$\sum_{g\in B} e_g \log_R g \equiv \sum_{i=1}^{z} a_i \log_R(\psi(\alpha_i))h^{-1} \pmod{\frac{p^n-1}{p-1}}.$$

Označíme-li $\beta_i := h^{-1}\log_R(\psi(\alpha_i))$, dostaneme finální tvar hledané lineární rovnice:

$$\sum_{g\in S} e_g \log_R g \equiv \sum_{i=1}^{z} a_i\beta_i \pmod{\frac{p^n-1}{p-1}}.$$

V této rovnici známe e_g pro všechna $g \in B$ (můžeme je snadno zjistit rozkladem $rm+s$). Postupem, který bude ještě podrobněji vysvětlen níže, můžeme vypočítat hodnoty $a_i = v_i(ry+s)$. Neznámými jsou tedy hodnoty $\log_R g$ a β_i.

Získáme-li takových rovnic dostatečně vysoký počet, jsme schopni jednoznačně určit hodnoty $\log_R g$ a β_i. Víme totiž, že $\log_R g_1 = \log_R x = 1$, a můžeme tedy řešit nehomogenní soustavu rovnic

$$\begin{pmatrix} e_{1,g_2} & e_{1,g_3} & \cdots & e_{1,g_{|B|}} & -a_{1,1} & \cdots & -a_{1,z} \\ e_{2,g_2} & e_{2,g_3} & \cdots & e_{2,g_{|B|}} & -a_{2,1} & \cdots & -a_{2,z} \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ e_{k,g_2} & e_{k,g_3} & \cdots & e_{k,g_{|B|}} & -a_{k,1} & \cdots & -a_{k,z} \end{pmatrix} \begin{pmatrix} \log_R g_2 \\ \log_R g_3 \\ \vdots \\ \log_R g_{|B|} \\ \beta_1 \\ \vdots \\ \beta_z \end{pmatrix} = \begin{pmatrix} -e_{1,x} \\ -e_{2,x} \\ \vdots \\ -e_{k,x} \end{pmatrix} \pmod{\frac{p^n-1}{p-1}}$$

7.2.8 Diskrétní logaritmus

Předpokládejme, že známe hodnoty $\log_R g$ z minulého odstavce, kde g jsou ireducibilní polynomy stupně nižšího než zvolená mez hladkosti. Mějme h polynom, jehož diskrétní logaritmus chceme spočítat.

Volme nyní náhodné $r \in \mathbb{N}$, dokud neplatí, že se $x^r h$ rozkládá do ireducibilních polynomů stupně nejvýše b. Pak

$$\log_R(x^r h) = r + \sum_{g \in G} e_i \log_R g \pmod{\frac{p^n - 1}{p - 1}}.$$

Vypočítat diskrétní logaritmus z omezeného diskrétního logaritmu je již snadné: máme-li a a známe-li $\log_R a$, platí:

$$x^{\log_R a} a^{-1} = u,$$

kde prvek u leží v prvotělese. Stačí tedy spočítat $\log u$ a dostáváme:

$$a = x^{\log_R a} x^{-\log u} = x^{(\log_R a) - (\log u)}.$$

Prvek u je přitom nenulový a nachází se v prvotělese, výpočet diskrétního logaritmu takového prvku je proto výrazně jednodušší než výpočet diskrétního logaritmu prvku obecného. Prvek $x^{(p^n-1)/(p-1)}$ jistě generuje cyklickou podgrupu řádu $p - 1$ grupy $F_{p^n}^*$, což je z Lagrangeovy věty nutně cyklická grupa F_p^*, tedy grupa všech nenulových prvků prvotělesa. Spočítáme-li tedy $x' = x^{(p^n-1)/(p-1)}$ a následně diskrétní logaritmus $\log_{x'} u$ v grupě F_p^*, platí

$$u = x'^{\log_{x'} u} = \left(x^{(p^n-1)/(p-1)}\right)^{\log_{x'} u} = x^{(\log_{x'} u) \cdot ((p^n-1)/(p-1))},$$

a tedy

$$\log u = \log_{x'} u \cdot \frac{p^n - 1}{p - 1}.$$

7.2.9 Optimalizace - prosívání

Teoretické úvahy i experimenty na dostatečně velkých vstupech ukazují, že časově nejnáročnější částí funkčního síta je hledání dvojnásobně hladkých párů, tedy dvojic (r, s) takových, že jsou oba polynomy $rm + s$ a $N(ry + s)$ b-hladké.

Podobně jako v případě index kalkulu, i v případě funkčního síta lze běh urychlit prosíváním - vyloučením některých kandidátů před samotným ověřením hladkosti, které obnáší polynomiální faktorizaci a je z hlediska času náročné. V implementaci, která je součástí této práce, byla použita technika prosívání navržená pro Coppersmithův algoritmus, popsaná v kapitole zabývající se optimalizacemi index kalkulu: volit pevné r a postupně prosívat možná s. Tím odfiltrujeme neperspektivní kandidáty, pro které $rm + s$ pravděpodobně není hladké, a u těch ostatních dvojnásobnou hladkost ověříme standardním rozkladem. Pokud je nalezen dostatek hladkých párů, hledání končí, jinak je zvoleno další r a pokračuje se v prosívání.

Polynomy r je vhodné volit nízkého stupně, protože $rm + s$ je hladký s vyšší pravděpodobností, pokud je nižšího stupně. Funkční síto v rychlosti těží z malé hladké báze a nízké hranice, kterou jsou omezeny stupně polynomů r a s. V některých případech, zejména pro menší vstupy, například pro $p = 2$, $n = 7$,

může dojít k situaci, kdy jsou vyzkoušeni všichni kandidáti na r a s a přesto není nalezeno dost dvojnásobně hladkých párů.

V takovém případě nezbývá než zvýšit mez pro stupně r a s a spustit funkční síto znovu. Zvýšit mez hladkosti nic neřeší; více párů bude dvojnásobně hladkých, ale řešená soustava bude také mít víc proměnných a bude zapotřebí více hladkých kongruencí.

V praxi se ukazuje, že se pro bazové polynomy malého stupně (nad $\mathbb{F}_2$ především x, $x+1$ a x^2+x+1) vyplácí prosívat i ve vyšších mocninách než v mocnině první. Nedojde tak k odfiltrování velkého počtu kandidátů, kteří dají vzniknout hladké kongruenci, ale neprojdou sítem.

V konkrétní implementaci funkčního síta je na první pohled zřejmý nepoměr v náročnosti výpočtu diskrétní valuace v tělese $\mathbb{F}_p(x)$ a v jeho rozšíření $quot(U[x,y]/(H))$. Zatímco výpočet diskrétní valuace v menším tělese znamená pouze polynomiální faktorizaci, ve větším tělese musí být použita náročná Newtonova mnohoúhelníková metoda.

Mohlo by proto být výhodné volit dvě různé meze hladkosti b_1 a b_2, kde $b_1 > b_2$, a polynomy $rm+s$ testovat na b_1-hladkost, kdežto polynomy $N(ry+s)$ na b_2-hladkost. Další možností by mohlo být nalezení dostatečného počtu hladkých relací a zjištění, pro které valuace má smysl Newtonovu metodu vůbec spouštět. Pokud je totiž daná valuace nulová pro dostatečný počet hladkých relací, vyplatí se ty relace, kde valuace nulová není, z výpočtu vyřadit a Newtonovu metodu pro tuto valuaci vůbec nespouštět.

7.3 Přehled použitých struktur

Funkční síto pracuje paralelně v několika strukturách. Následující přehled by měl čtenáři pomoci zorientovat se v používaných symbolech a ozřejmit vztahy mezi jednotlivými strukturami.

Definice	Struktura	Značení	Prvky a zobrazení
$\mathbb{F}_p$	Těleso		a
$\mathbb{F}_p[x]$	Okruh		$f, m, g_1, \ldots, g_k, r, s$
$\mathbb{F}_p[x]/(f)$	Těleso	$\mathrm{GF}(p^n)$	h, x
$\mathbb{F}_p(x)$	Rac. funkční těleso		$v_{g_1}, \ldots, v_{g_k}$
$\mathbb{F}_p[x,y]$	Okruh		H
$\mathbb{F}_p[x,y]/(H)$	Okruh		ψ
$quot(\mathbb{F}_p[x,y]/(H))$	Alg. funkční těleso	$K(C)$	$ry+s, v_1, \ldots, v_z$

Prvotěleso $\mathbb{F}_p$ lze chápat jako vnořené do většího tělesa $\mathrm{GF}(p^n)$. Mezi okruhem $\mathbb{F}_p[x,y]/(H)$ a $\mathrm{GF}(p^n)$ působí homomorfismus ψ:

$$\mathbb{F}_p \longrightarrow \mathbb{F}_p[x] \xleftarrow{\psi} \mathbb{F}_p[x,y]/(H)$$

Následující diagram znázorňuje vztah mezi dvěma funkčními tělesy, se kterými algoritmus pracuje: racionálním funkčním tělesem $\mathbb{F}_p(x)$ a jeho rozšířením $quot(\mathbb{F}_p[x,y]/(H))$. Diskrétní valuace $v_{g_1}, \ldots, v_{g_k}$ se v rozšíření rozkládají do diskrétních valuací $v_1, \ldots, v_z$.

$$\begin{tikzcd}
\mathbb{F}_p \arrow[d] & \\
\mathbb{F}_p(x) \arrow[r] \arrow[d, "v_{g_1}, \ldots, v_{g_k}"'] & quot(\mathbb{F}_p[x,y]/(H)) \arrow[dl, "v_1, \ldots, v_z"] \\
\mathbb{Z} \cup \{\infty\} &
\end{tikzcd}$$

7.4 Výpočet diskrétních valuací

V průběhu algoritmu je zapotřebí vypočítat hodnoty všech diskrétních valuací odpovídajících polynomům z množiny G pro polynom $ry + s$ pro každý nalezený hladký pár r a s.

V následujících úvahách budeme používat značení:

- C buď křivka zadaná polynomem H.
- $H_\infty(z, y) := z^m H(1/z, y)$, kde m je stupeň H v x

Každý bod křivky C ležící v nekonečnu odpovídá nějakému bodu $(0, b)$ na křivce zadané H_∞, tedy

$$H_\infty(0, b) = 0.$$

Bod v nekonečnu křivky C je tak zachycen jako vlastní bod křivky zadané H_∞ [2]. Teorie týkající se projektivních křivek a jejich vztahu k afinním křivkám je detailně popsána například v [30].

Mějme nyní libovolnou diskrétní valuaci v funkčního tělesa nad křivkou C. Řekneme, že v leží v bodě $P = (a, b)$, jestliže $v(x - a) > 0$ a $v(y - b) > 0$. Řekneme, že v leží v nekonečnu, jestliže $v(x) < 0$ a $v(y - b) > 0$.

Nechť g je ireducibilní polynom z $\mathbb{F}_p[x]$. Pak se diskrétní valuace $\mathbb{F}_p(x)$ nad $\mathbb{F}_p$ odpovídající g rozkládá na několik diskrétních valuací ve funkčním tělese nad C, které leží v bodech (a, b), kde a je kořen g nad rozkladovým nadtělesem U a pro b platí, že $H(a, b) = 0$. Tato a a b můžeme snadno vypočítat v polynomiálním čase pomocí polynomiální faktorizace. Pro daný polynom g nalezneme všechna taková a a b a tím všechny diskrétní valuace ležící v (a, b), na které se diskrétní valuace odpovídající g rozkládá. Abychom nalezli diskrétní valuaci v nekonečnu, budeme analogicky pracovat s H_∞ a body $(0, b)$, že $H_\infty(0, b) = 0$.

Mějme tedy diskrétní valuaci v ležící v bodě $P = (a, b)$ na křivce definované H. Bez újmy na obecnosti můžeme předpokládat, že $P = (0, 0)$, neboť (a, b) můžeme

na $(0,0)$ zobrazit vhodnou transformací souřadnic. F buď polynom udávající křivku C po transformaci souřadnic.

Jelikož v leží v bodě $(0,0)$, jistě existuje $e \in \mathbb{N}$, že

$$v(x) = e > 0.$$

Pro danou diskrétní valuaci v můžeme nalézt uniformizující prvek t, že $v(t) = 1$, a tedy:

- $x = t^e$
- $y = u_1 t^{m_1} + u_2 t^{m_2} + \ldots$, kde $\{m_i\}_{i \geq 1}$ je rostoucí posloupnost celých čísel a u_i jsou nenulové z U. Mocninnou řadu, do které je rozvinut prvek y, chápeme jako prvek P-adického zúplnění algebraického funkčního tělesa. Veškeré operace, které budeme v následujících odstavcích provádět s nekonečnými mocninnými řadami, chápeme jako operace v P-adickém zúplnění.

Z vyjádření y výše plyne, že $v(y) = m_1$. Uvažme prvek

$$\overline{y} = u_1 x^{\lambda_1} + u_2 x^{\lambda_2} + \ldots$$

kde $\lambda = m_i/e$. Pak platí, že $F(x, \overline{y}) = 0$. Známe-li navíc takové $\overline{y}$ je v některých případech možné odvodit e jako nejmenší společný jmenovatel prvních $2n^3$ mocnin s nenulovým koeficientem. Adleman a Huang se v [2] odkazují na [23] a tvrdí, že postup tam popsaný funguje pro všechna funkční tělesa.

Opomíjí však, že v [23] je metoda vykládána výhradně pro tělesa charakteristiky 0, zatímco v případě funkčního síta je charakteristika vždy kladná. V kladné charakteristice může být obecně posloupnost jmenovatelů hodnot λ_i shora neomezená. Pro funkční síto to ovšem neznamená zásadní překážku: v případě, že bude výsledkem $\overline{y}$ s neomezenou posloupností jmenovatelů, může být zvolena nová křivka H a výpočet může začít znovu.

Obráceně lze ukázat, že každá mocninná řada $\overline{y}$ v $x^{1/e}$ splňující $F(x,y) = 0$ odpovídá diskrétní valuaci splňující $v(x) = e$ ležící v bodě $(0,0)$.

Máme-li prvek z funkčního tělesa, vyjádřený polynomem $A(x,y)$, je $v(z)$ rovno řádu $A(x(t), y(t))$ v t, kde $x(t) = t^e$ a $y(t)$ je mocninná řada v $x^{1/e}$.

$y(t)$ může obecně mít nekonečný rozvoj, pro výpočet $v(ry + s)$, kde r a s tvoří dvojnásobně hladký pár, ovšem stačí určit konečný počet členů, jak bude vysvětleno v následujícím odstavci.

Norma polynomu $ry + s$ nad $\mathbb{F}_p(x)$ má díky vhodné volbě H a omezením kladeným na stupeň polynomů r a s stupeň nejvýše n^2 a je tvaru

$$(-s)^d + r^d x^\delta w(x).$$

Všechny diskrétní valuace $N(ry+s)$ v $\mathbb{F}_p(x)$ jsou proto omezeny rovněž hodnotou n^2. Z odhadu

$$n^2 \geq v_P(N(ry+s)) = \sum_{Q|P} f(Q|P) v_Q(ry+s),$$

uvedeného v teoretickém úvodu, tedy plyne odhad pro každou jednotlivou diskrétní valuaci prvku $ry+s$ v $quot(U[x,y]/(H))$.

Víme-li, že $v(ry+s)$ je omezená hodnotou $l \in \mathbb{N}$, stačí určit prvních l členů $y(t)$, protože ve všech dalších členech je již mocnina t ostře vyšší než l, a tedy také počítané diskrétní valuace těchto členů jsou jistě ostře vyšší než l. Z trojúhelníkové nerovnosti tak plyne, že se na výsledné diskrétní valuaci podílí nejvýše prvních l členů $y(t)$, v našem případě $l = n^2$.

Z nalezené mocninné řady

$$\overline{y} = u_1 x^{\lambda_1} + u_2 x^{\lambda_2} + \dots$$

můžeme zpravidla hodnotu e určit jako nejmenší přirozené číslo takové, že $\overline{y}$ je mocninnou řadou v $x^{\frac{1}{e}}$.

Nalezení všech diskrétních valuací nad daným polynomem g se tak redukuje na výpočet prvních $2n^3$ členů všech mocninných řad y v x splňujících $F(x,y) = 0$. To lze provést za pomoci Newtonovy mnohoúhelníkové metody.

Shrnutí výpočtu diskrétních valuací odpovídajících polynomu g prvku $ry+s$ pro r, s dvojnásobně hladký pár:

1. Nalezneme všechny dvojice (a,b), kde $a \in U$, $b \in U$, a je kořen g a b splňuje, že $H(a,b) = 0$. Každé takové dvojici odpovídá jedna diskrétní valuace $v_{a,b}$ tělesa $U[x,y]/(H)$nad algebraicky uzavřeným tělesem.

2. Pro každou takovou dvojici (a,b) provedeme transformaci souřadnic, která bod (a,b) zobrazí na $(0,0)$. Křivku definovanou polynomem H pro nové souřadnice označíme F.

3. Nalezneme prvních $2n^3$ členů všech mocninných řad $\overline{y}$ takových, že $F(x,\overline{y}) = 0$.

4. Pro každou takovou mocninnou řadu $\overline{y}$ odvodíme hodnotu e jako nejmenší přirozené číslo takové, že $\overline{y}$ je mocninnou řadou v $x^{\frac{1}{e}}$.

5. Do $ry+s$ dosadíme za y prvních n^2 členů $\overline{y}$. Dostaneme polynom $A(x)$.

6. Diskrétní valuace $ry+s$ odpovídající dvojici (a,b) je pak rovna

$$v_{a,b} = e \cdot O_x(A).$$

Výhodou funkčního síta je skutečnost, že časově náročný výpočet všech mocninných řad $\overline{y}$ pomocí Newtonovy metody může být proveden jednou jako předvýpočet a následně využíván při všech výpočtech konkrétních hodnot diskrétních valuací $ry+s$ všech dvojnásobně hladkých párů. Předvýpočet totiž nijak nezávisí na $ry+s$, vstupem je pouze H a polynom, kterým je realizováno rozšíření tělesa.

7.5 Newtonova mnohoúhelníková metoda

V této části popíšeme obecný algoritmus, který se využívá jako dílčí algoritmus ve funkčním sítu. Mějme těleso K. Nechť

$$F(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_{ij}x^{a_j}y^i = \sum_{i=0}^{n} f_i(x)y^i,$$

kde $f_{ij} \in K$, a_i jsou racionální čísla tvořící rostoucí posloupnost a

$$f_i(x) = \sum_{j=0}^{m} f_{ij}x^{a_j}.$$

Pro

$$\overline{y} = u_1x^{\lambda_1} + u_2x^{\lambda_2} + \dots$$

označme $O_x(\overline{y}) := \lambda_1$, $O_x(0) := \infty$.

Chceme nalézt $\overline{y}$ mocninnou řadu v $x^{1/e}$ splňující $F(x,\overline{y}) = 0$ a zároveň $O_x(\overline{y}) > 0$. Algoritmus, který umožňuje spočítat prvních k členů každé takové řady, bývá nazýván Newtonova metoda nebo také Newtonova-Puiseuxova metoda a je popsán například v [21], [2] nebo [22].

Pro $i = 0, \dots, n$ buď f_{ij_i} koeficient nejmenšího nenulového členu$f_i(x)$, tedy $a_{j_i} = O_x(f_i)$. Označme $b_i = a_{j_i}$. Newtonovým mnohoúhelníkem [22] pak budeme rozumět konvexní uzávěr množiny

$$N(F) = \{(i, b_i) : 0 \le i \le n, b_i < \infty\}.$$

Předpokládejme, že E je strana Newtonova mnohoúhelníku ležící na přímce zadané rovnicí $\gamma X + Y = \beta$. Polynomem zadaným stranou E budeme rozumět polynom

$$h(z) = \sum_i f_{ij_i}z^i,$$

kde součet probíhá přes všechna taková i, že bod (i, b_i) leží na straně E.

Pokud by pro všechna i byla hodnota b_i nenulová, mohli bychom z F vytknout vhodnou mocninu x. Můžeme proto předpokládat, že $b_i = 0$ pro nějaké i. Nechť d je nejmenší takové i.

Z definice Newtonova mnohoúhelníku lze vypozorovat, že klesající strany mnohoúhelníku leží nalevo od bodu $(d, b_d) = (d, 0)$. Jediným případem, kdy Newtonův mnohoúhelník žádné klesající strany nemá, je případ $f_0 = f_1 = \dots = f_{d-1} = 0$. V takovém případě $F(x, 0) = 0$.

Opačně máme-li nenulovou mocninnou řadu y, splňující $O_x(y) > 0$, pak nutně $F(x,y) \neq 0$, jelikož pro $i > d$ platí:

$$O_x(f_iy^i) \ge iO_x(y) > dO_x(y) = O_x(f_dy^d).$$

$F(x, y)$ nazveme konečným případem, jestliže

$$f_0 = \ldots = f_{d-1} = 0$$

pro nějaké $d > 0$ a $O_x(f_d) = 0$. V minulém odstavci jsme ukázali, že jediným řešením konečného případu je $y = 0$. Cílem metody založené na Newtonově mnohoúhelníku bude počítat mocninnou řadu rekurzivně člen po členu a s každým nalezeným členem modifikovat funkci F až do konečného případu, jehož jediným řešením je nulová řada. U těch řešení, která neskončí konečným případem, vypočítáme pouze požadovaný počet členů.

Mějme tedy obecný polynom $F(x, y)$, který není konečným případem. Nechť

$$\overline{y} = u_1 x^{\gamma_1} + u_2 x^{\gamma_1+\gamma_2} + \ldots$$

splňuje $F(x, \overline{y}) = 0$. Naším cílem je odvodit nutné podmínky pro $u = u_1$ a $\gamma = \gamma_1$. Můžeme psát:

$$\overline{y} = x^{\gamma}(u + y_1),$$

kde y_1 je nová mocninná řada

$$y_1 = \sum_{i=2}^{\infty} u_i x^{\gamma_2+\cdots+\gamma_i}.$$

Odtud

$$F(x, \overline{y}) = F(x, x^{\gamma}(u + y_1)) = \sum_{i,j} f_{ij} x^{i\gamma+a_j}(u + y_1)^i = x^{\beta} F_1(x, y_1),$$

kde $\beta = \min\{i\gamma + a_j : f_{ij} \neq 0\} = \min\{i\gamma + b_i : i = 0, \ldots, n\}$.

Uvažme

$$h(z) = \sum_{a_j+i\gamma=\beta} f_{ij} z^i.$$

Z předpokladu, že F není konečný případ, musí mít Newtonův mnohoúhelník F alespoň jednu klesající stranu. Každá taková klesající strana nám dává jednu možnou hodnotu γ a několik možných hodnot u.

Nechť klesající strana Newtonova mnohoúhelníku leží na přímce splňující $\gamma_1 X + Y = \beta_1$. Nechť H_1 je polynom určený touto stranou. Pak každý nenulový kořen H_1 je možnou hodnotou u_1.

S každou takto nalezenou dvojicí γ_1 a u_1 můžeme definovat mocninnou řadu $F_1(x, y)$, splňující

$$F(x, x^{\gamma_1}(u_1 + y)) = x^{\beta_1} F_1(x, y).$$

Zřejmě $F(x, x^{\gamma_1}(u_1 + y)) = 0$ právě tehdy, když $F_1(x, y) = 0$. Dostaneme-li tedy F_1 konečný případ, jsou všechny další členy pro dříve vypočítané členy řady $\overline{y}$ nulové. V opačném případě opakujeme uvedený postup pro F_1, a odhalíme tak

všechny možnosti pro γ_2 a u_2. S každou další iterací vypočítáme další člen hledané mocninné řady.

Shrnutí Newtonovy mnohoúhelníkové metody pro prvních k členů mocninné řady pro

$$F(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} f_{ij} x^{a_j} y^i$$

1. Pokud je $F(x,y)$ konečný případ, je jeho řešením pouze nulová řada.
2. Pokud $F(x,y)$ není konečný případ, nalezneme všechny jeho klesající strany.
3. Pro každou klesající stranu:

   Nechť klesající strana leží na přímce $\gamma_1 X + Y = \beta_1$ a H_1 je polynom určený touto stranou.

   Nalezneme všechny nenulové kořeny H_1.

   Pro každý u_1 nenulový kořen H_1:

   Prvním členem řešení bude člen $u_1 x^{\gamma_1}$. Jeho dalšími členy budou členy všech možných řešení pro

$$F_1(x,y) = \frac{F(x, x^{\gamma_1}(u_1 + y)}{x^{\beta_1}}.$$

Jak navíc uvádí [31], Newtonovu metodu lze provést v polynomiálním čase vzhledem k počtu hledaných členů a stupni polynomu F.

7.5.1 Implementace

Newtonova mnohoúhelníková metoda byla pro účely funkčního síta implementována v rámci této práce a zdrojový kód implementace je k nahlédnutí v přiložených souborech `newtonPolygonMethod.cpp` a `newtonPolygonMethodGF.cpp`. V prvním souboru implementace hledající řešení

$$F(x,y) = 0,$$

kde zadaný polynom F je nad tělesem $\mathbb{F}_2$ a jsou nalezena řešení pouze nad tímto dvouprvkovým tělesem. Druhý soubor obsahuje obecné řešení; koeficientem polynomu dvou proměnných F může být prvek libovolného konečného tělesa a jsou nalezena všechna řešení nad tímtéž konečným tělesem.

Pro funkční síto je důležité, aby byla nalezena všechna řešení v algebraickém uzávěru prvotělesa, tedy nekonečném tělese. Aby algoritmus všechna taková řešení odhalil, musí pracovat v tělese tak velkém, aby se v něm rozkládaly na kořenové činitele všechny polynomy klesajících stran, které se v průběhu vyskytnou. Nejvyšší možný stupeň polynomu klesající strany je možné odhadnout stupněm n polynomu F v proměnné y. V praxi je tedy možné spustit Newtonovu mnohoúhelníkovou metodu pro těleso velikosti p^k, kde k je dostatečně velké, a nalézt

tak všechna řešení. k je hledáno nejprve tak, aby se v tělese $\mathrm{GF}(p^k)$ rozkládal polynom g_i, jehož příslušné diskrétní valuace jsou právě předpočítávány, a polynom H po transformaci souřadnic a dosazení 0 za x.

Polynomy stran Newtonových mnohoúhelníků, které se vyskytnou v průběhu algoritmu, nelze zjistit s předstihem. V případě, že se některý z polynomů strany v tělese nerozloží, začíná dílčí výpočet znovu pro větší těleso.

V průběhu algoritmu se pracuje se zobecněnými polynomy tvaru

$$F(x,y) = \sum_i \sum_j a_{ij} x^{\lambda_j} y^i,$$

kde λ_j jsou racionální čísla. Není proto možné pracovat s obvyklými strukturami standardní knihovny `NTL`. Pro reprezentaci polynomů dvou proměnných výše uvedeného tvaru proto byla použita struktura `std::vector<std::vector<pair_ZZ_pE_RR>>`, kde `pair_ZZ_pE_RR` je dvojice obsahující prvek tělesa $\mathrm{GF}(p^n)$ a racionální číslo. Prvky vnějšího vektoru představují polynomy $f_i(x)$ splňující

$$F(x,y) = \sum_i f_i(x) y^i.$$

Vnitřní vektor na pozici i je potom seznamem nenulových členů ve tvaru (koeficient, mocnina) konkrétního členu v polynomu $f_i(x)$.

Klíčovým úsekem implementace je rekurzivní procedura `nextStep`, která pro zadané F nalezne všechny možné první členy hledané posloupnosti a pro každý z nich se znovu spustí s cílem najít členy další. De facto tak prohledáváním do hloubky projde všechna možná řešení, která průběžně ukládá do proměnné `std::vector<pair_ZZ_pE_RR> result`.

Jakmile dosáhne stanoveného počtu nalezených členů, nebo dorazí do konečného stavu, vypíše nalezené řešení na výstup a pokračuje dalšími dosud nezpracovanými možnostmi.

7.6 Časová složitost

Z teoretické části algoritmu zůstalo několik nezodpovězených otázek, které souvisejí se správným nastavením parametrů tak, abychom u funkčního síta dosáhli optimální časové složitosti. Jsou jimi otázky:

- Jak volit stupeň polynomu m, tedy konstantu d_1?
- Jak volit konstantu W, a omezit tak stupeň polynomů r a s, u kterých budeme testovat dvojnásobnou hladkost?
- Jak volit mez hladkosti b?

Je patrné, že u žádné ze tří konstant nelze volit extrémní možnosti: příliš malé W může kupříkladu znamenat příliš málo hladkých kongruencí na to, aby byla vzniklá matice jednoznačně řešitelná. Vysoké W naopak snižuje pravděpodobnost,

že bude náhodně zvolený pár dvojnásobně hladký. Následující úvahy jsou převzaty z [2].

Pro dané n a p budeme hledat W a d_1 následujícího tvaru:

$$W = c_2 n^{\frac{1}{3}} (\log n)^{\frac{2}{3}} (\log p)^{-\frac{2}{3}},$$

$$d_1 = c_1 n^{\frac{2}{3}} (\log n)^{\frac{1}{3}} (\log p)^{-\frac{1}{3}},$$

kde c_1 a c_2 jsou konstanty na n a p již nezávislé.

Nejprve uvažme pravděpodobnost, že pro náhodně zvolené r a s je pár (r, s) dvojnásobně hladký. Pár (r, s) je dvojnásobně hladký, jsou-li polynomy $rm + s$ a $N(ry + s)$ hladké, tedy je-li polynom $(rm + s)N(ry + s)$ hladký.

Z volby r, s a m vyplývá, že

$$\deg(rm + s) \leq W + d_1 = n^{\frac{2}{3}} (\log n)^{\frac{1}{3}} (\log p)^{-\frac{1}{3}} (c_1 + o(1)).$$

Podobně lze odhadnout stupeň $N(ry + s)$:

$$\deg(N(ry + s)) \leq n^{\frac{2}{3}} (\log n)^{\frac{1}{3}} (\log p)^{-\frac{1}{3}} (\frac{c_2}{c_1} + o(1)).$$

Můžeme tedy zvolit D tvaru

$$D = (c_1 + \frac{c_2}{c_1} + o(1)) n^{\frac{2}{3}} (\log n)^{\frac{1}{3}} (\log p)^{-\frac{1}{3}},$$

a odhadnout tak stupeň polynomu $(rm + s)N(ry + s)$.

Buď $Q = p^D$ a zvolme mez hladkosti b takto:

$$b = \log_p(L_Q[\frac{1}{2}; \frac{1}{\sqrt{2}}]).$$

Dále budeme vycházet z heuristického předpokladu, že se pro náhodně zvolené r a s chová polynom $(rm + s)N(ry + s)$ jako náhodný polynom stupně nejvýše D. Pak, jak je uvedeno v [2] s odkazem na [29], je pár (r, s) dvojnásobně hladký s pravděpodobností $L_Q[\frac{1}{2}; -\frac{1}{\sqrt{2}} + o(1)]$. Soustava lineárních rovnic, která v průběhu algoritmu vzniká, bude mít nejvýše $L_Q[\frac{1}{2}; \frac{1}{\sqrt{2}}]$ proměnných. Potřebujeme proto vyzkoušet alespoň $L_Q[\frac{1}{2}; \sqrt{2} + o(1)]$ párů. Takových párů je ovšem právě $p^{2c_2 n^{\frac{1}{3}} (\log n)^{\frac{2}{3}} (\log p)^{-\frac{2}{3}}}$, z čehož nutně plyne odhad

$$L_Q[\frac{1}{2}; \sqrt{2}] \leq p^{2c_2 n^{\frac{1}{3}} (\log n)^{\frac{2}{3}} (\log p)^{-\frac{2}{3}}}.$$

D bylo zvoleno v závislosti na konstantách c_1 a c_2, a odhad tak lze upravit [2] do tvaru

$$\frac{c_2 + c_1^2}{3c_1} \leq c_2^2.$$

Řešením jsou hodnoty $c_1 = (\frac{2}{3})^{\frac{1}{3}}$, $c_2 = (\frac{4}{9})^{\frac{1}{3}}$. Během hledání hladkých kongruencí je tedy vyzkoušeno $L_Q[\frac{1}{2}; \sqrt{2} + o(1)]$ párů a poté je vypočítána soustava

rovnic zadaná řídkou maticí. To všechno proběhne v čase, který lze shora odhadnout hodnotou

$$L_{p^n}[\frac{1}{3}; (\frac{32}{9})^{\frac{1}{3}} + o(1)].$$

Závěrečná fáze algoritmu, v níž je počítán částečný diskrétní logaritmus zadaného prvku počítán pomocí již známých částečných diskrétních logaritmů prvků z hladké báze, proběhne v čase $L_{p^n}[\frac{1}{3}; (\frac{3}{2})^{\frac{1}{3}} + o(1)]$. Odtud tedy plyne, že pro časovou složitost celého algoritmu je určující časová složitost generování hladkých kongruencí [2].

Pro závěrečnou fázi algoritmu je v [2] popsán trik, s jehož pomocí lze rychle spočítat diskrétní logaritmus prvku reprezentovaného polynomem stupně $\Delta \geq b$ pomocí diskrétního logaritmu jistých dvou polynomů, které jsou již $\sqrt{\Delta b}$-hladké.

V závěrečné fázi pak pro hledané $\log h$ nejprve zvolíme $l < n$ náhodné a spočítáme $x^l h \mod f$. Tento polynom bude $\sqrt{nb}$-hladký s pravděpodobností

$$\exp(-\sqrt{n/b}(\log(\sqrt{n/b}) + \log\log(\sqrt{n/b}) + o(1)).$$

Můžeme ho tedy rozložit na polynomy stupně nejvýše $\sqrt{n/b}$ a diskrétní logaritmus každého takového polynomu spočítat pomocí zmíněné redukce z diskrétních logaritmů nejvýše n polynomů stupně $\sqrt{b\sqrt{n/b}}$ Takto pokračujeme rekurzivně dál a výpočet diskrétního logaritmu prvku h redukujeme po t iteracích na výpočet diskrétního logaritmu nejvýše n^t polynomů stupně nejvýše $b(\frac{n}{b})^{2^{-t}}$. Po dostatečně mnoha krocích, tedy pro $t = \log_2 n$, jsou všechny polynomy v hladké bázi a jejich diskrétní logaritmy máme předpočítané z fáze hledání hladkých kongruencí.

7.7 Paralelizace

Podobně jako v případě index kalkulu, může být i v případě funkčního síta značná část algoritmu paralelizována. Situace, kdy je možné výpočet v průběhu algoritmu paralelizovat, jsou dvě: v předvýpočtu, kdy jsou pomocí Newtonovy metody hledány všechny diskrétní valuace ve větším funkčním tělese, a poté při samotném hledání hladkých párů. Přestože by předvýpočet z asymptotického hlediska měl být zanedbatelný, v praxi se ukazuje, že je výpočetně poměrně náročný a jeho paralelizace může výrazně přispět ke zrychlení celého algoritmu.

7.8 Algoritmus v pseudokódu

Celý algoritmus funkčního síta lze zachytit v následujícím pseudokódu. Některé složitější úlohy, jako například Newtonova mnohoúhelníková metoda, generování polynomu H nebo výpočet diskrétních valuací pro daný polynom $ry + s$, jsou v pseudokódu chápány jako atomické operace. Vstupem funkčního síta je prvočíslo p, přirozené číslo n a prvek h, jehož diskrétní logaritmus v $\mathrm{GF}(p^n)$ hledáme.

function FUNCTIONFIELDSIEVE(p, n, h)
  $f(x) \leftarrow x^n + q(x)$
  label start

$b, d_1 \leftarrow$ `computeConstants`(n)
$m(x) \leftarrow x^{d_1}$
$H(x, y) \leftarrow$ `generateH`(f, m)
$B \leftarrow$ `findIrredPolynomials`(b)
$Eq \leftarrow \{\}$
found $\leftarrow 0$
while found $\leq 4|B|$ **do**
  $r \leftarrow$ `randomPolynomial`()
  $s \leftarrow$ `randomPolynomial`()
  if notSmooth$(rm + s, b)$ **then**
    continue
  end if
  if notSmooth$(x^d H(x, -\frac{s}{r}))$ **then**
    continue
  end if
  found $\leftarrow$ found $+1$
  $a_1, \ldots, a_k \leftarrow$ `computeDiscValuations`$(ry + s, B)$
  $g_1, \ldots, g_l, e_1, \ldots, e_l \leftarrow$ `factor`$(rm + s)$
  $Eq \leftarrow Eq \cup \{\sum_{i=1}^{l} e_i \log_R g_i = \sum_{i=1}^{k} a_i \beta_i\}$
end while
$\log_R g_1, \ldots, \log_R g_k \leftarrow$ `solve`(Eq)
$q \leftarrow$ `random`()
while notSmooth$(x^q h)$ **do**
  $q \leftarrow$ `random`()
end while
$g_1, \ldots, g_l, e_1, \ldots, e_l \leftarrow$ `factor`$(x^q h)$
$\log_R(h) \leftarrow -r + \sum_{g \in G} e_i \log_R g.$
$u \leftarrow x^{\log_R h} h^{-1}$
$p = \log_R(h) - \log(u)$
if $x^p = h$ **then**
  return p
else
  $f \leftarrow$ `change`(f)
  goto start
end if
end function

Několik poznámek: polynom $H(x, y)$ na začátku funkce počítáme pomocí vyjádření f v bázi mocnin m, množinu B inicializujeme pomocí ireducibilních polynomů, které máme předpočítány. Dále máme předpočítány primitivní polynomy, čímž urychlíme počáteční vygenerování malé části $q(x)$ polynomu f.

V úvodu algoritmu je třeba sestavit hladkou bázi z ireducibilních polynomů do určitého stupně. Ireducibilní polynomy jsou předvyhledávány programem se zdrojovým kódem v přiloženém souboru `irredPolynomialGen.cpp`. Podobně je vhodné f zvoleno z předpočítaných primitivních polynomů, které jsou generovány programem v přiloženém souboru `primitivePolynomialsGen.cpp`. Součástí této práce jsou programy pro generování těchto polynomů k dispozici pouze pro těleso $\mathbb{F}_2$, pro větší prvotělesa musí uživatel předložit soubor s předgenerovanými iredu-

cibilními a primitivními polynomy. Případná úprava přiložených programů pro větší prvotělesa by nebyla obtížná.

Faktorizace polynomů probíhá v polynomiálním čase díky Cantor–Zassenhausovu algoritmu. Pro řešení soustavy lineárních rovnic je nutno použít rychlou Wiedemannovu metodu pro řídké matice.

V samotném závěru provádíme kontrolu výsledku. Pokud skončí neúspěchem, začínáme od začátku s jinou volbou polynomu f. Jak k neúspěchu může dojít? V průběhu algoritmu totiž předpokládáme, že je třídové vzniklého funkčního tělesa nesoudělné s $\frac{p^n-1}{p-1}$ a počítáme h^{-1}, které nemusí vždy existovat.

V současné době však není znám žádný algoritmus, jak třídové číslo s předstihem efektivně spočítat a tuto podmínku ověřit. Vhodný polynom však lze zvolit s dostatečně vysokou pravděpodobností a tato nepříjemnost v samotném závěru algoritmu se naštěstí nepromítne do očekávané časové složitosti.

V proceduře `solve` můžeme soustavu rovnic `Eq` počítat modulo $\frac{p^n-1}{p-1}$, výsledné hodnoty jsou hodnotami částečného diskrétního logaritmu. Číslo $\frac{p^n-1}{p-1}$ bohužel nemusí být prvočíslo, soustavu rovnic tak neřešíme v tělese, ale v okruhu, výpočet je tak výrazně obtížnější, protože inverzní prvky nemusí existovat. Tento problém lze částečně řešit pomocí čínské věty o zbytcích: soustavu řešíme postupně modulo prvočinitele $\frac{p^n-1}{p-1}$ a řešení pomocí čínské věty o zbytcích zrekonstruujeme.

8. Srovnání algoritmů

V rámci práce byly implementovány konkrétní algoritmy řešící problém diskrétního logaritmu. U všech algoritmů byl měřen čas v závislosti na délce vstupu a jednotlivá měření následně numericky aproximována hladkou křivkou pravděpodobné závislosti naměřeného času na délce vstupu.

Algoritmy „baby steps - giant steps“ a Pollardův ρ algoritmus byly spouštěny v Java Runtime Environment 1.6 od společnosti Sun v prostředí operačního systému OpenSuse 11.4 s výkonem 2,4 GHz a 2GB paměti. Numerická analýza byla následně provedena pomocí software Mathematica 9.0, konkrétně pomocí funkce `FindFit[]`. Všechny grafy byly rovněž generovány pomocí programu Mathematica.

Z uvedených parametrů hardware je patrné, že experimenty neměly ambice dosáhnout současných rekordů ani se k nim přiblížit. Cílem srovnání bylo pouze experimentálně doložit rozdíly ve složitosti mezi jednotlivými algoritmy spouštěním na konkrétních vstupech. Implementace žádného z algoritmů není optimální a jistě nabízí prostor pro případnou optimalizaci.

Pro měření času byla v jazyce Java použita systémová funkce `currentTimeMillis()`. Za výstup měření byl považován přírůstek hodnoty této funkce před zavoláním metody `discreteLogarithm` a po něm.

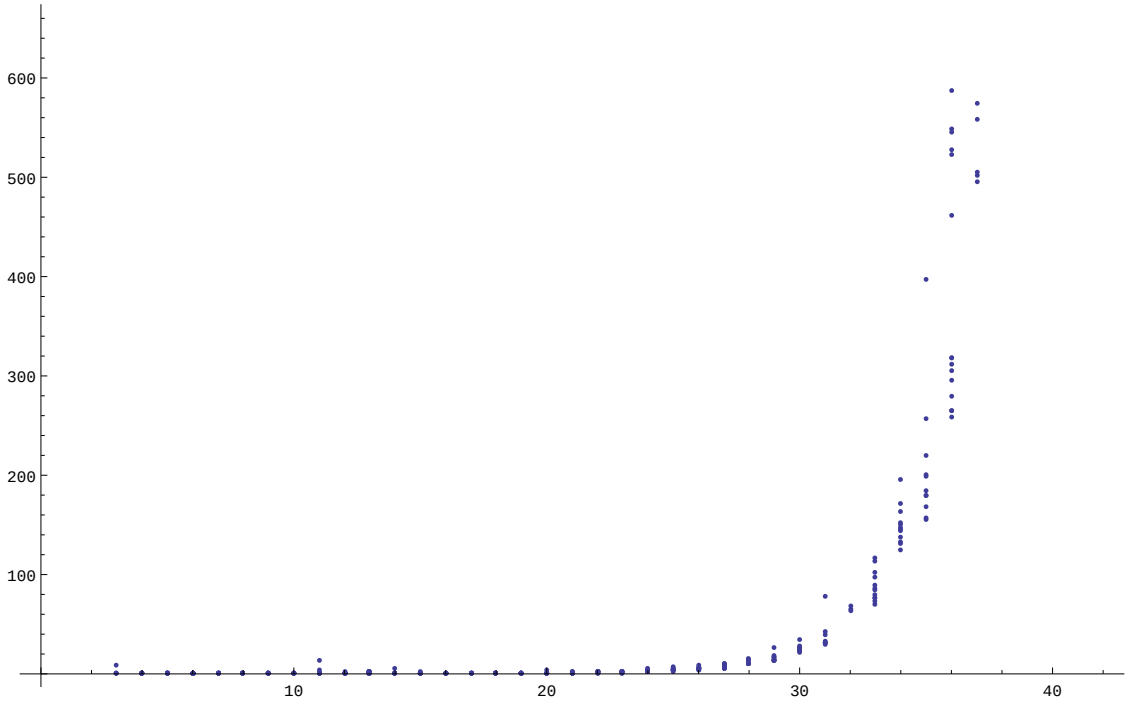
V jazyce C bylo využito přímo rozdílu v systémovém čase; měření probíhalo pomocí shell skriptu, který spouštěl index kalkulus a Coppersmithův algoritmus na vstupy různé velikosti a v okamžiku spuštění a ukončení běhu zaznamenal aktuální systémový čas do logovacího souboru.

8.1 Baby steps - giant steps

Měření časové náročnosti algoritmu „baby steps - giant steps“ proběhlo v grupách $\mathbb{Z}_p^*$, kde prvočíslo p bylo voleno od délky 4 do 40 v binárním zápisu. V rámci měření byla pro každou délku náhodně zvolena 4 různá prvočísla, pro každé prvočíslo 5 potenciálních generátorů $\mathbb{Z}_p^*$ a byl spočítán diskrétní logaritmus čísla 3 v daném generátoru. Pokud vstupní prvek neležel v grupě generované potenciálním generátorem, nebyl čas zahrnut do celkového měření.

Úplný přehled přesných naměřených hodnot je k vidění v přiloženém souboru. Experiment zároveň potvrdil hlavní slabinu „baby steps - giant steps“: od délky vstupu přibližně 45 začala docházet paměť. Pro větší vstupy by bylo nutné implementovat vytváření baby - steps seznamu do několika různých souborů, které by byly s každým giant step postupně prohledávány. Tím by se ovšem dramaticky zvýšil čas nutný pro vyhledávání, hešování by nebylo možné.

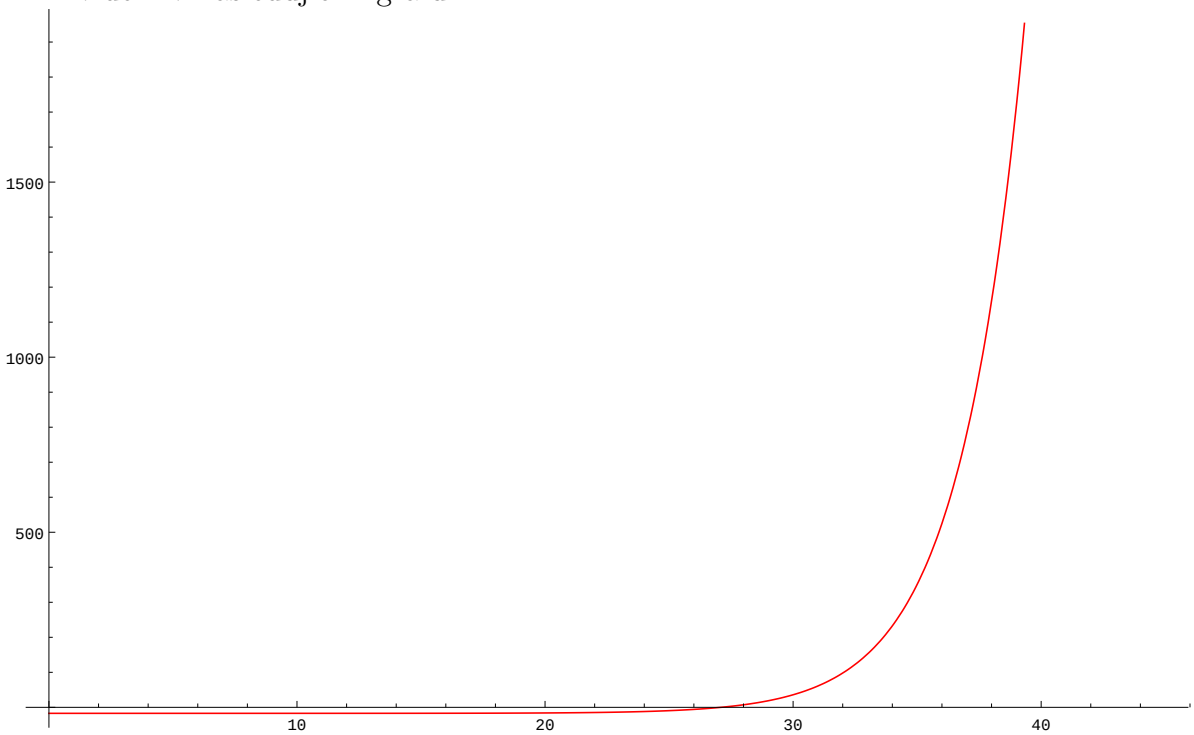
Následující graf zachycuje výstupy jednotlivých měření v milisekundách v závislosti na délce velikosti grupy.



Numericky se podařilo aproximovat naměřené hodnoty pomocí hladké křivky tvaru $a \cdot 2^{b \cdot x} + c$ pro hodnoty:

- $a = 0.0004731251713110929$
- $b = 0.559114906710799$
- $c = -17.137100553430237$

Tyto hodnoty odpovídají očekávané složitosti ($\sqrt{p}$), průběh hladké křivky je k vidění v následujícím grafu:



8.2 Pollard ρ

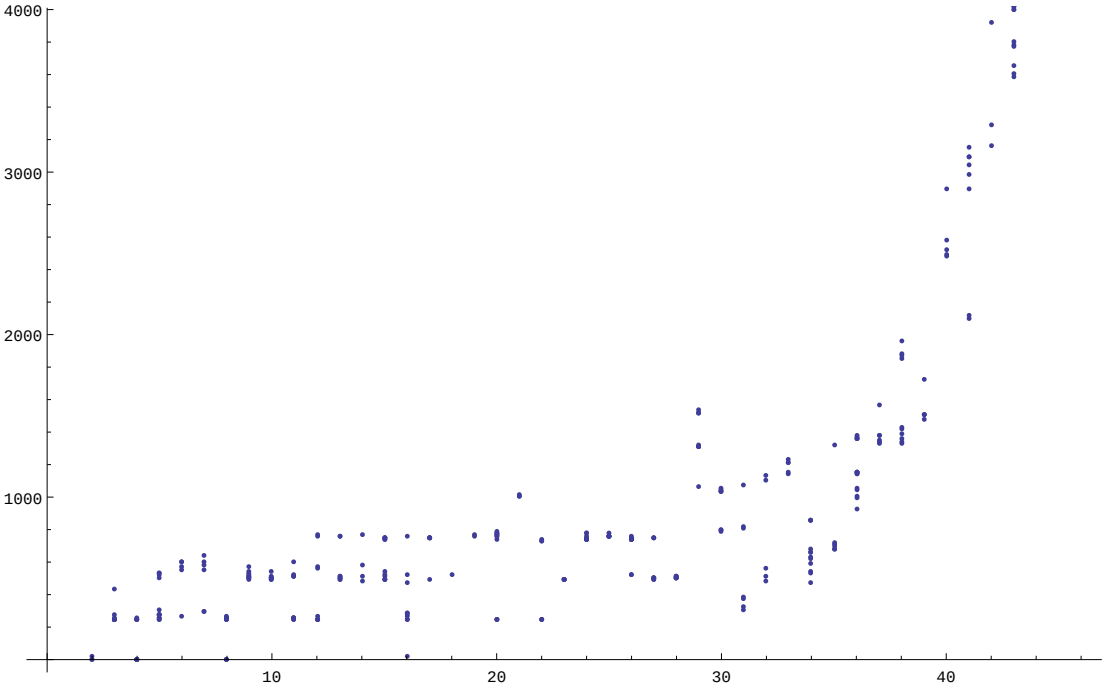
Podobně jako u „baby steps - giant steps“ proběhlo měření časové náročnosti algoritmu Pollard ρ v grupách $\mathbb{Z}_p^*$, kde prvočíslo p bylo voleno od délky 4 do 40 v binárním zápisu. V rámci měření byla pro každou délku náhodně zvolena 4 různá prvočísla, pro každé prvočíslo 5 potenciálních generátorů $\mathbb{Z}_p^*$ a byl spočítán diskrétní logaritmus čísla 3 v daném generátoru. Pokud vstupní prvek neležel v grupě generované potenciálním generátorem, nebyl čas zahrnut do celkového měření.

Pro vyšší efektivitu Pollardova algoritmu byla použita třída `DLReductor`, která převádí problém diskrétního logaritmu z obecné cyklické grupy do cyklické grupy prvočíselného řádu a z jednotlivých dílčích výsledků skládá celkový výstup. Do naměřeného času se tak promítla i složitost faktorizace řádu grupy.

Skutečný naměřený čas také do značné míry ovlivnil rozklad čísla $p-1$ pro zvolené prvočíslo p. V některých grupách (například pro prvočísla tvaru $p = 2^k + 1$) tak výpočet proběhl velmi rychle; faktorizace čísla 2^k je triviální, vyřešení diskrétního logaritmu v $\mathbb{F}_2$ rovněž. Ve srovnání s obecným „baby steps - giant steps“, který strukturu $p-1$ nijak nevyužívá, tak proběhl výpočet v některých grupách výrazně rychleji.

Z grafů pro „baby steps - giant steps“ a Pollardův ρ algoritmus je patrný rozdíl mezi deterministickým přístupem „baby steps - giant steps“ a pravděpodobnostním přístupem Pollard ρ. Zatímco měření „baby steps - giant steps“ vykazovala jen velmi malý rozptyl, naměřené hodnoty Pollard ρ vykazují rozptyl o poznání vyšší.

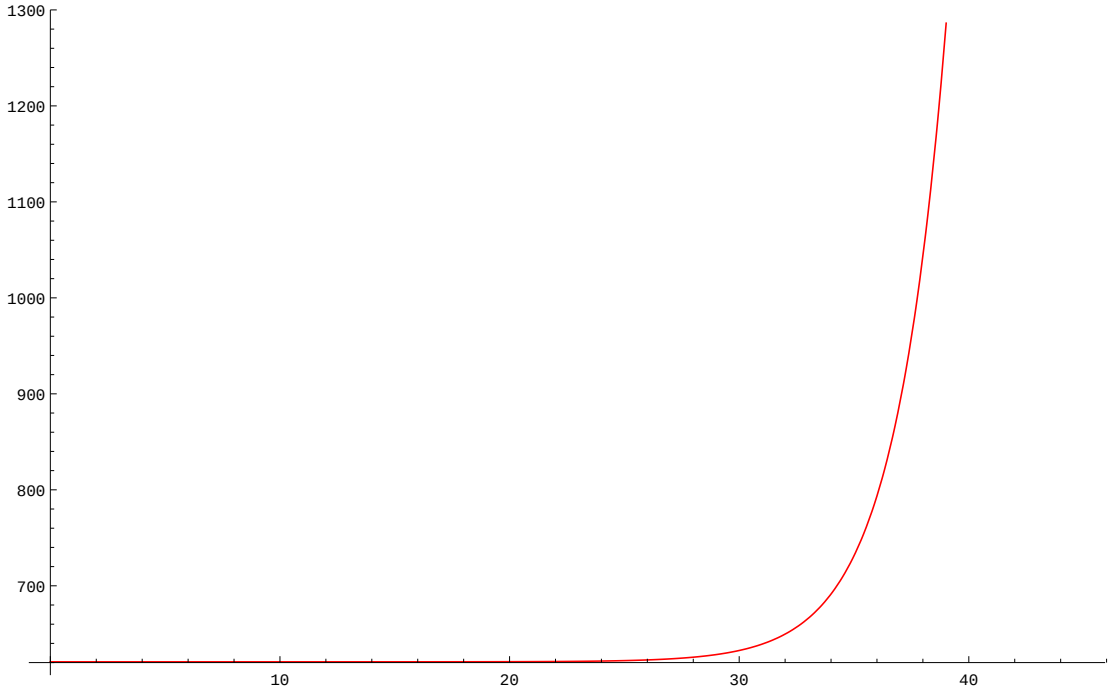
Následující graf zachycuje výstupy jednotlivých měření v milisekundách v závislosti na délce velikosti grupy.



Numericky se podařilo aproximovat naměřené hodnoty pomocí hladké křivky tvaru $a \cdot 2^{b \cdot x} + c$ pro hodnoty:

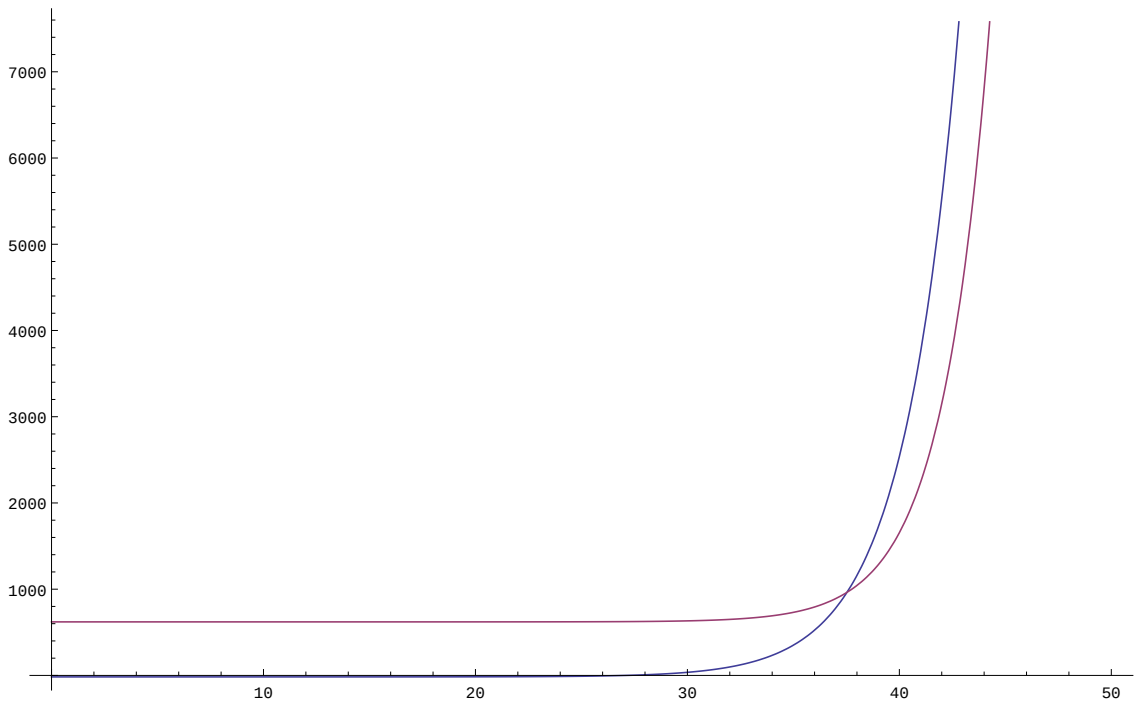
- $a = 0.0000175738216744665 27$
- $b = 0.0.6452576772694218$
- $c = 620.7928677614678$

Tyto hodnoty odpovídají očekávané složitosti ($\sqrt{p}$), graf hladké křivky je k vidění na následujícím obrázku:



8.3 Baby steps - giant steps vs. Pollard ρ

Podle naměřených časů vychází „baby steps - giant steps“ lépe pro menší grupy, zatímco Pollard ρ pro větší. Složitost Pollard ρ je v grafu zaznačena fialovou barvou, „baby steps - giant steps“ modrou:

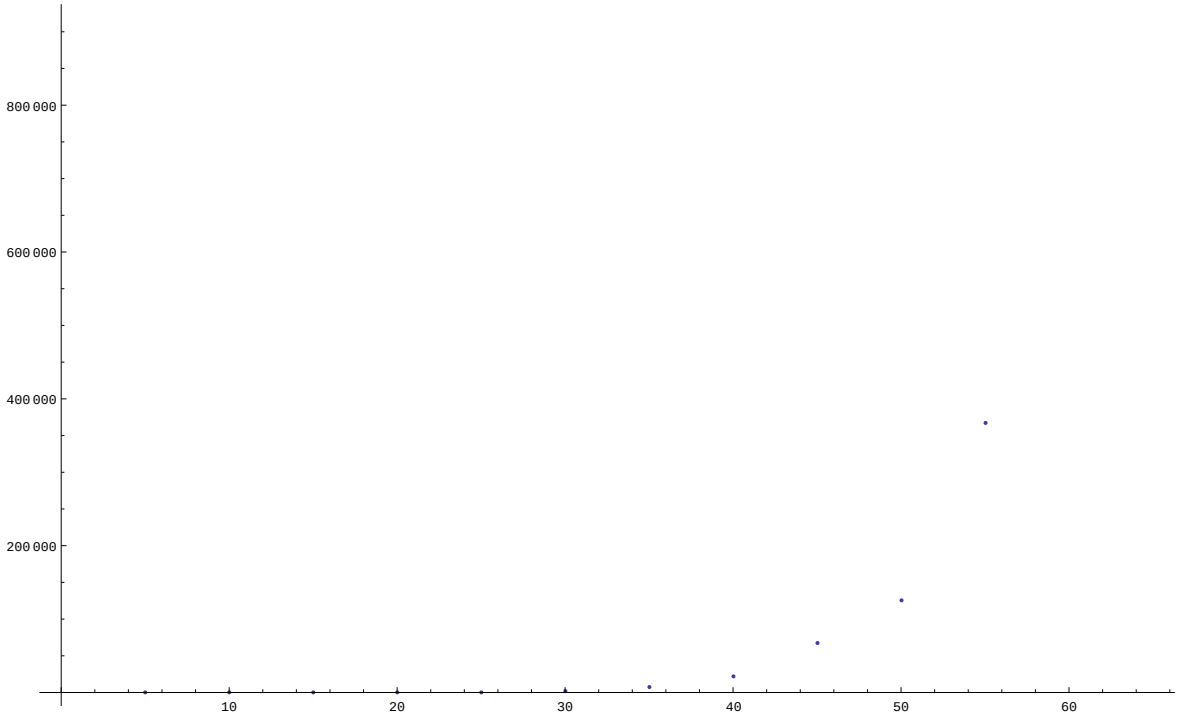


Výsledek do značné míry ovlivnila faktorizace v Pollard ρ, nezbytná i pro poměrně malé grupy. Z experimentu vyplývá, že od délky 38 je výhodnější Pollard ρ, nejen pro svou nižší prostorovou složitost.

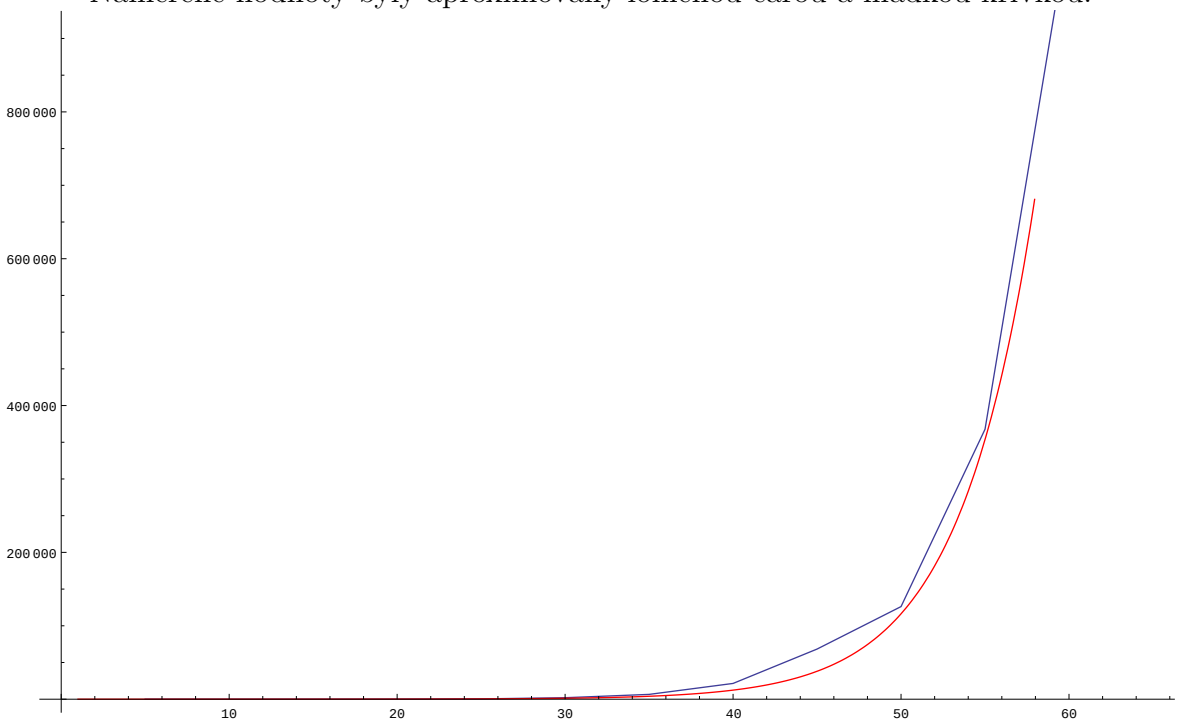
8.4 Index kalkulus

Kromě standardní verze index kalkulu byl v rámci této práce implementován také pokročilejší Coppersmithův algoritmus ve verzi bez prosívání a s prosíváním. Oba algoritmy byly postupně spouštěny v grupách $\mathrm{GF}(2^n)^*$, kde se n pohybovalo v intervalu od 5 do 65, na každém vstupu byly oba algoritmy spuštěny pětkrát a z jednotlivých výsledků byl spočítán průměr.

Na následujícím grafu jsou zachyceny výstupy jednotlivých měření index kalkulu v milisekundách v závislosti na n:

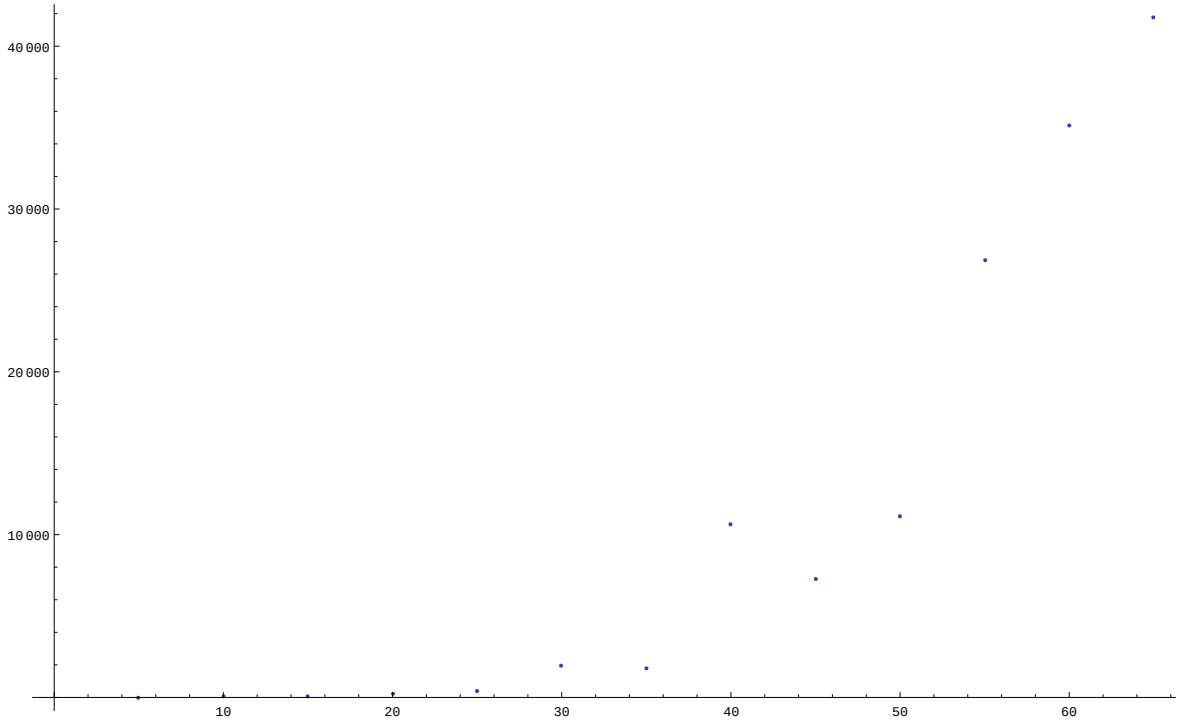


Naměřené hodnoty byly aproximovány lomenou čárou a hladkou křivkou:

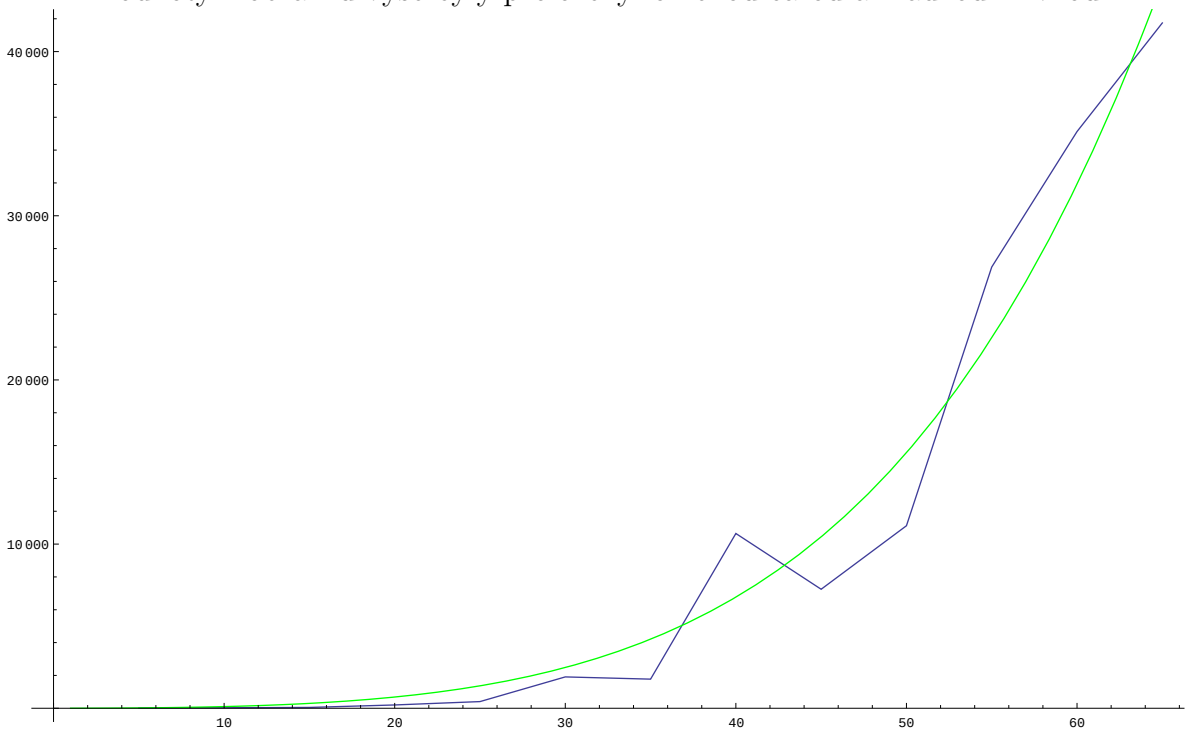


Podobně byla měřena také výkonnost Coppersmithova algoritmu. Základní algoritmus bez prosívání byl spouštěn pro stejné vstupy jako index kalkulus, tedy v grupách $\mathrm{GF}(2^n)^*$, kde se n pohybovalo v intervalu od 5 do 65, pětkrát pro každý vstup s následným zprůměrováním naměřených hodnot.

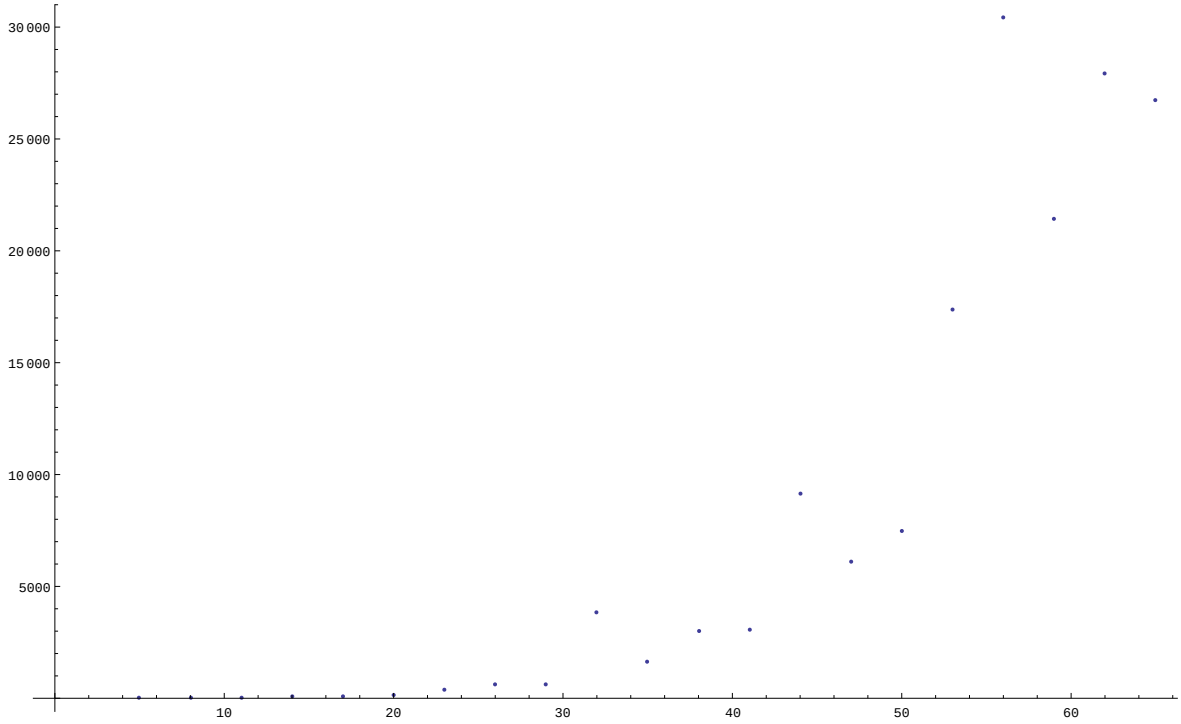
Na následujícím grafu jsou zachyceny výstupy jednotlivých měření Coppersmithova algoritmu v milisekundách v závislosti na n:



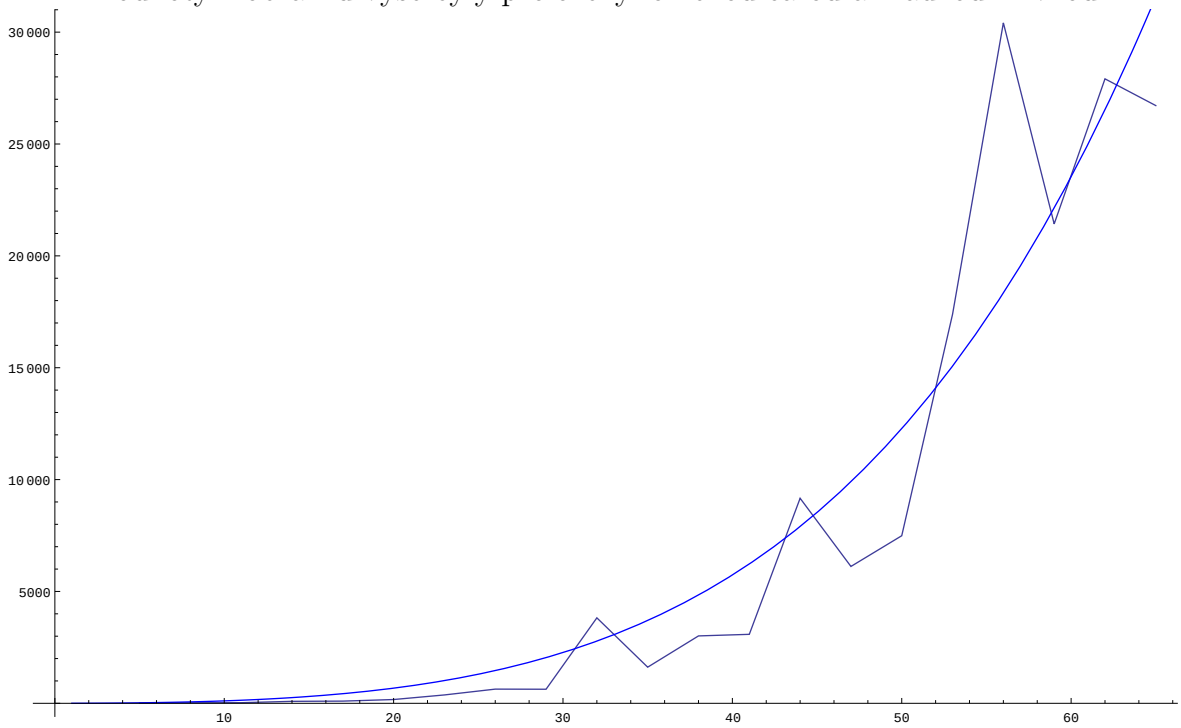
Hodnoty z obrázku výše byly proloženy lomenou čárou a hladkou křivkou:



Prosívání Coppersmithův algoritmus ještě vylepšilo; v grupách $\mathrm{GF}(2^n)^*$, kde se n pohybovalo v intervalu od 5 do 65, byly naměřeny následující hodnoty:

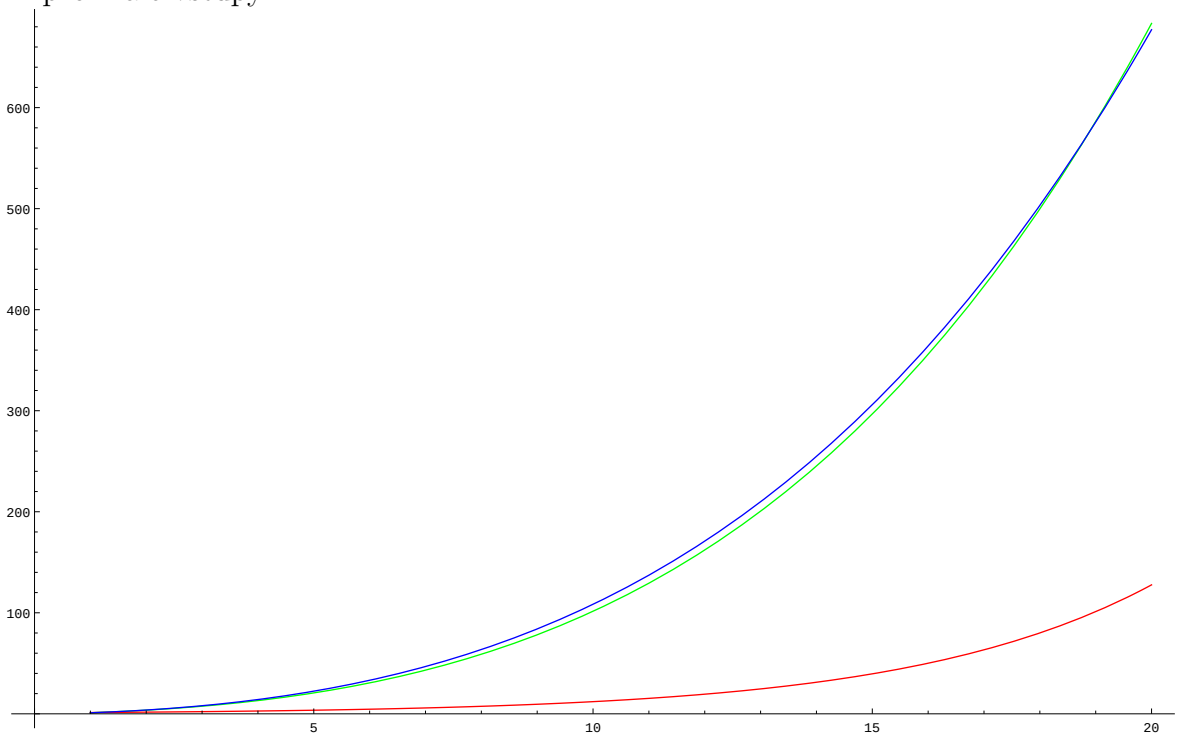


Hodnoty z obrázku výše byly proloženy lomenou čárou a hladkou křivkou:

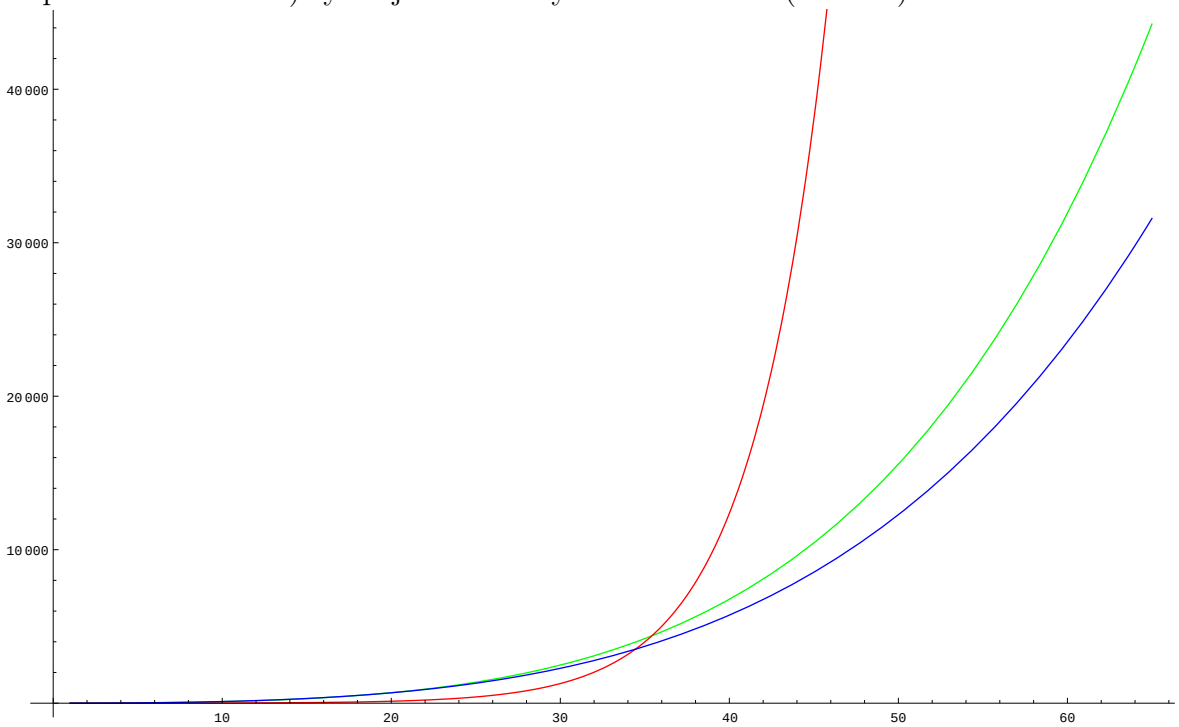


Z hlediska praktického využití nejzajímavějším výstupem měření je srovnání obou algoritmů na stejných vstupech. Naměřené hodnoty potvrdily teoretické odhady: index kalkulus byl rychlejší pro malé grupy pro svou jednoduchost a rychlé generování kandidátů na hladké polynomy, zatímco Coppersmithův algoritmus se ukázal být efektivnější pro grupy větší, přibližně pro $n > 35$, kde začala hrát roli menší báze hladkých prvků a v důsledku toho také nižší počet hledaných hladkých kongruencí. Pro dostatečně velké vstupy se projevila výhoda prosívání: pro malé vstupy nehrálo prosívání zásadní roli, přibližně od řádu 30 však začalo být prosívání výrazně rychlejší.

Následující graf je srovnáním index kalkulu (červeně), Coppersmithova algoritmu bez prosívání (zeleně) a Coppersmithova algoritmu s prosíváním (modře) pro malé vstupy:



Ve větších grupách je už Coppersmithův algoritmus (bez prosívání zeleně, s prosíváním modře) rychlejší než běžný index kalkulus (červeně):



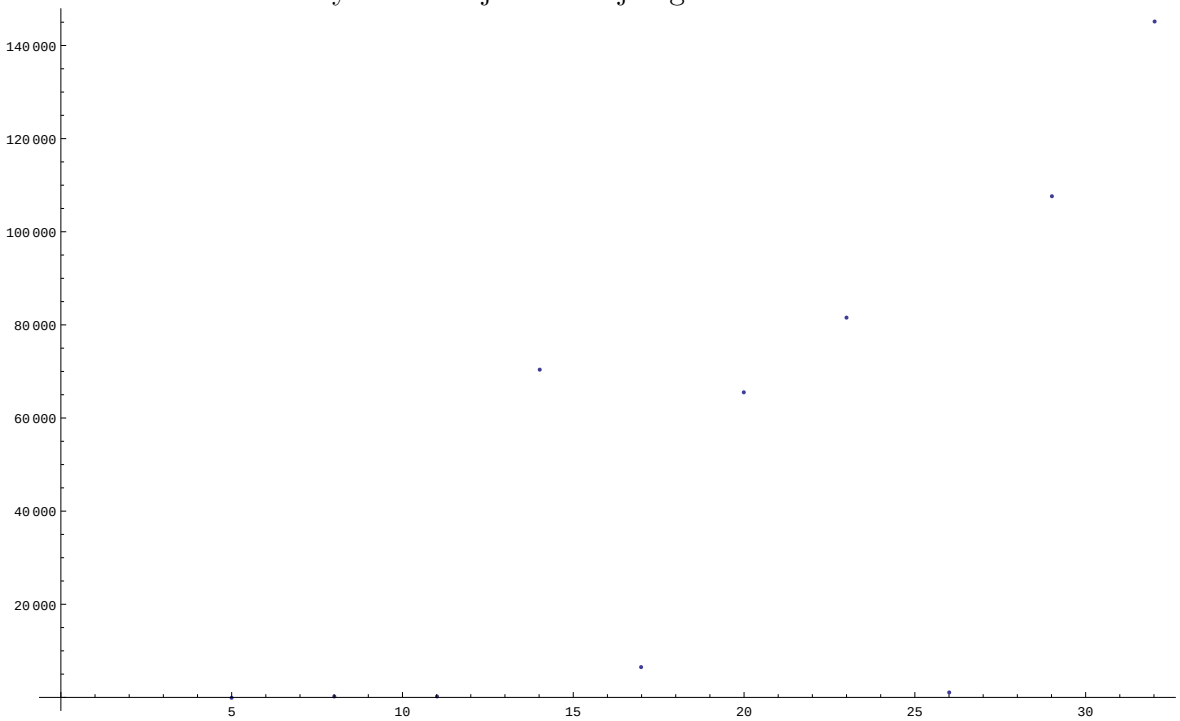
8.5 Funkční síto

Měření výkonu funkčního síta s prosíváním v $\mathrm{GF}(2^n)^*$ probíhalo podobně jako měření jednotlivých implementací index kalkulu; algoritmus byl spouštěn postupně pro zvětšující se vstupy od 10 do 37, na každém vstupu byl spuštěn pětkrát a výsledné hodnoty byly zprůměrovány.

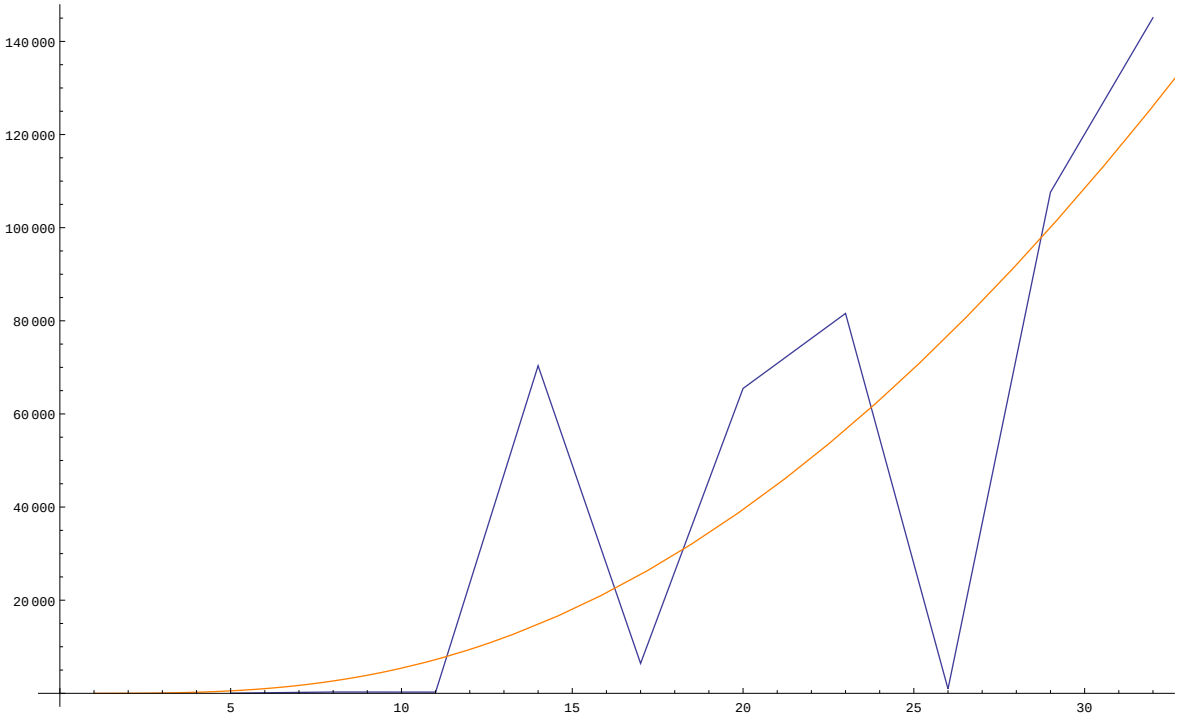
Na tomto místě je nutné zmínit, že na rozdíl od výše uvedených algoritmů výkon funkčního síta výrazně kolísal v závislosti na zvolené velikosti grupy a nezřídka funkční síto dosahovalo paradoxně lepších výsledků na větší grupě. V praxi tak vyšlo najevo, že se skutečný výkon funkčního síta odvíjí především od křivky H, která je použita pro konstrukci funkčního nadtělesa. Konkrétní volba křivky totiž udává klíčové parametry běhu:

- počet diskrétních valuací, do kterých se rozkládají diskrétní valuace odpovídající bázovým prvkům, a tedy nutný počet hledaných hladkých kongruencí, protože každé diskrétní valuaci odpovídá jedna proměnná v soustavě,
- stupeň algebraického rozšíření, ve kterém probíhá předvýpočet diskrétních valuací,
- počet iterací Newtonovy mnohoúhelníkové metody.

Naměřené hodnoty znázorňuje následující graf:

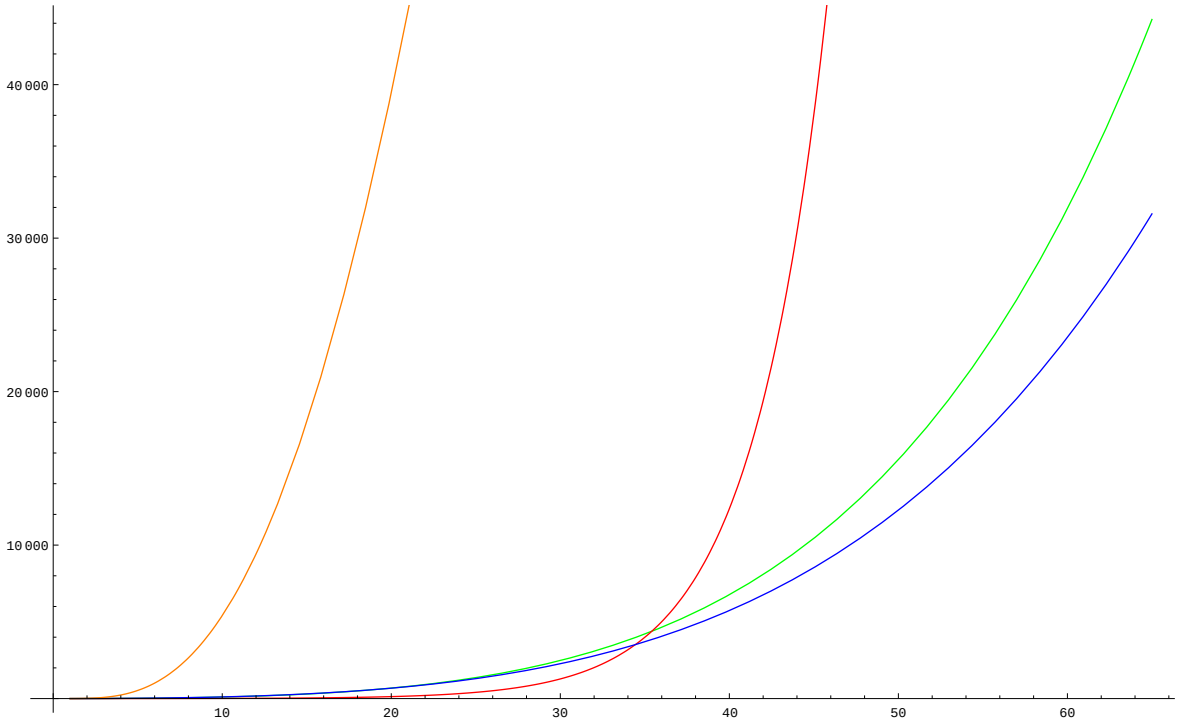


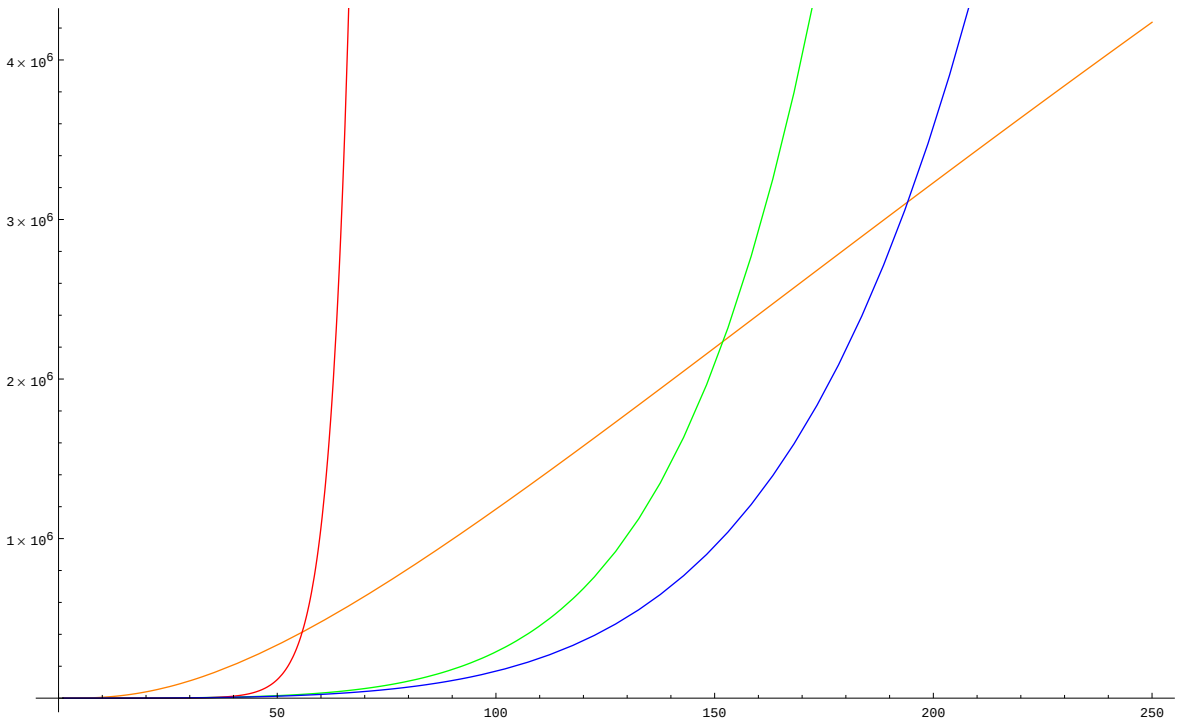
Body lze proložit lomenou čarou a hladkou křivkou pomocí procedury `FindFit` takto:



Vzhledem k výkyvům v časové náročnosti bohužel nelze aproximaci hladkou křivkou připisovat větší váhu. Pro přesnější aproximaci by bylo nutné spustit algoritmus na výkonnějším hardware a pro větší počet vstupů.

Následující srovnání funkčního síta (oranžově) a jednotlivých implementací index kalkulu (standardní červeně, Coppersmithův bez prosívání zeleně, Coppersmithův s prosíváním modře) je proto pouze orientační a skutečné hodnoty se od odhadovaných budou pravděpodobně dost lišit. Křivka odpovídající časové složitosti funkčního síta roste zpočátku velmi rychle, ostatní algoritmy funkční síto překoná pravděpodobně až pro velké grupy řádu přibližně 200.




4 × 10^6
3 × 10^6
2 × 10^6
1 × 10^6
50
100
150
200
250

9. Závěr

9.1 Shrnutí

V předchozích kapitolách jsme se zabývali problémem diskrétního logaritmu v konečné cyklické grupě a věnovali jsme se čtyřem algoritmům, které diskrétní logaritmus řeší, studovali souvislost diskrétního logaritmu s celočíselnou faktorizací a zvláštní pozornost jsme věnovali grupě $\mathrm{GF}(p^n)^*$ a metodám založeným na hladkých prvcích - index kalkulu a funkčnímu sítu.

Přes všechno úsilí, které bylo diskrétnímu logaritmu v posledních desítkách let věnováno, není problém uspokojivě vyřešen a v posledních letech dochází především k vylepšování a pilování algoritmů již známých; poslední převratný pohled na věc nabídl index kalkulus před více než třiceti lety.

9.2 Otevřené otázky

V souvislosti se zmíněnými algoritmy a obecně s diskrétním logaritmem zůstávají některé otevřené otázky, které mohou být (a pro svou důležitost pravděpodobně budou) předmětem dalšího zkoumání:

- Existuje polynomiální algoritmus pro výpočet diskrétního logaritmu?
- Existuje subexponenciální algoritmus pro výpočet diskrétního logaritmu v grupě bodů obecné eliptické křivky?
- Existuje způsob, jak v grupě bodů obecné eliptické křivky zavést hladké prvky?
- Jak rigorózně dokázat heuristické odhady složitosti Pollardova ρ algoritmu a náhodnost volených polynomů v index kalkulu?
- Jak efektivně počítat třídové číslo v obecném algebraickém funkčním tělese?
- Lze vylepšit funkční síto na očekávanou časovou složitost $L[x]$ pro $x < \frac{1}{3}$?

9.3 Přínosy práce

Práce pokrývá čtyři důležité algoritmy, které se používají pro výpočet diskrétního logaritmu v obecných a speciálních cyklických grupách a jejichž vylepšení by znamenalo hrozbu pro některé dnes používané kryptografické transformace a protokoly. Důležitým přínosem práce je tak teoretické srovnání funkčního síta coby dnes nejrychlejšího známého algoritmu a původního základního index kalkulu, který funkční síto svými principy inspiroval a dnes je jím ve velkých grupách již překonán.

Ve své práci jsme se dále věnovali možným vylepšením index kalkulu: polynomiálním sítem, Semaevovým a především Coppersmithovým algoritmem, který

index kalkulus dále rozvíjí a dokonce ve speciálních grupách vylepšuje jeho očekávanou asymptotickou složitost. Myšlenky, na kterých jsou Coppersmithův a další vylepšení založena, by jednoho dne mohly vést i ke zdokonalení funkčního síta.

Výstupem práce je srovnání časové náročnosti jednotlivých algoritmů na konkrétních vstupech a ověření vzhledem k očekávané, teoreticky odvozené složitosti.

A nakonec hlavním výstupem práce jsou samotné implementace uvedených algoritmů: algoritmu „baby steps - giant steps“, Pollardova ρ algoritmu, základního index kalkulu, Coppersmithova algoritmu ve variantě bez prosívání a s prosíváním a konečně funkčního síta. Při implementaci byla vyřešena celá řada dílčích problémů, které jsou studovány v předchozích kapitolách, a výstupem práce jsou kromě samotného programu také implementace podpůrných algoritmů: generování ireducibilních polynomů, generování řídkých primitivních polynomů a Newtonova mnohoúhelníková metoda aplikovaná na výpočet diskrétních valuací v tělese $quot(U[x, y]/(H))$.

Seznam použité literatury

[1] Adleman, Leonard. *The Function Field Sieve.*

[2] Adleman, Leonard; Huang, Ming-Deh. *Function Field Sieve Method for Discrete Logarithms over Finite Field.* Information and Computation 151: 5 - 16

[3] Sedgewick, Robert; Szymanski, Thomas G.. *The complexity of finding cycles in periodic functions*, SIAM Journal on Computing 11(2): 376 - 390.

[4] Pomerance, Carl. *Elementary Thoughts On Discrete Logarithms.*

[5] Bach, Eric. *Discrete Logarithms and factoring.*

[6] *Java Language Specification.* `http://docs.oracle.com/javase/1.5.0/docs/`

[7] *Java Algebra System (JAS) Project.* `http://krum.rz.uni-mannheim.de/jas/`

[8] Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin. *PRIMES is in P.*

[9] Rabin, Michael. *Probabilistic algorithm for testing primality*, Journal of Number Theory, 1980 Vol. 1

[10] *Angelus AKS Primality Test.* `http://www.angelusworld.com/`

[11] Leslie, Martin. *Elliptic Curve Cryptography*, Advanced Combinatorics 2006.

[12] National Institute of Standards and Technology. *Recommended elliptic curves for federal government use*, `http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf`, 1999

[13] Adleman, Leonard. *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proc. 20th IEEE Found. Comp. Sci. Symp., 1979, 55-60.

[14] Odlyzko, Andrew. *Discrete logarithms in finite fields and their cryptographic significance.*

[15] Wagner, Urs. *Wiedeman's algorithm*, 2008.

[16] *C programming.* `http://www.cprogramming.com/` .

[17] *The GNU Multiple Precision Arithmetic Library.* `gmplib.org/` .

[18] *NTL: A Library for doing Number Theory.* `http://www.shoup.net/ntl/` .

[19] Somberg, Petr. *Algebraic Geometry in Positive Characteristic.* `http://www.karlin.mff.cuni.cz/~somberg/lectures.htm`

[20] Somberg, Petr. *Elliptic Curves.* `http://www.karlin.mff.cuni.cz/~somberg/lectures`

[21] Verschelde, Jan. *The Newton-Puiseux method*, Analytic Symbolic Computation, Lecture 15, 2011.

[22] Garret, Paul. *Newton Polygons*, 2005.

[23] Dicrescenzo, Claire; Duval, Dominique. *Computations on Curves*, Lecture Notes in Computer Science, Vol. 174, str. 100 - 107, Springer-Verlag, 1984.

[24] *Wikipedia, the Free Encyklopedia.* `http://en.wikipedia.org/wiki/Puiseux_series`

[25] Joux, Antoine. *Faster index calculus for the medium prime case - Application to 1175-bit and 1425-bit finite fields.*

[26] Schirokauer, Oliver. *The impact of the number field sieve on the discrete logarithm problem in finite fields*, Algorithmic Number Theory, MSRI Publications, Vol. 44, 2008.

[27] Stichtenoth, Henning. *Algebraic Function Fields and Codes*, Springer Verlag 2009.

[28] H. L., Dorwart. *Irreducibility of polynomials*, American Mathematical Monthly 42, Vl. 6, 1935.

[29] R., Lovorn. *Rigorous Subexponential Algorithms for Discrete Logarithms over Finite Fields*, 1992.

[30] David, Goldschmidt. *Algebraic Functions and Projective Curves*, Springer Verlag 2003.

[31] Adrien, Poteaux; Marc, Rybowicz. *Complexity bounds for the rational Newton-Puiseux algorithm over finite fields*, Applicable Algebra in Engineering, Communication and Computing, Volume 22, Issue 3, pp 187-217, 2011.