Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

**BAKALÁŘSKÁ PRÁCE**

Milan Straka

# Faktorizace polynomů nad konečnými tělesy

Katedra algebry

Vedoucí bakalářské práce: Mgr. Jan Žemlička, Ph.D.

Studijní program: informatika, obecná informatika

2006

Rád bych poděkoval vedoucímu své práce Mgr. Janu Žemličkovi, Ph.D., za odborné vedení během mé práce a také za skvělé přednášky o algebře, které mě k tomuto tématu přivedly.

Také bych velmi rád poděkoval Janě Kravalové a Petrovi Sušilovi za korektury textu a Mgr. Martinu Marešovi jak za korektury tak za podnětné připomínky k implementaci programu.

V neposlední řadě bych chtěl poděkovat svým rodičům, bez jejichž ochoty a trpělivosti bych nemohl tuto práci dokončit.

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

V Praze dne 29. 5. 2006 Milan Straka

# Obsah

Název práce: Faktorizace polynomů nad konečnými tělesy

Autor: Milan Straka

Katedra (ústav): Katedra algebry

Vedoucí bakalářské práce: Mgr. Jan Žemlička, Ph.D.

E-mail vedoucího: *Jan.Zemlicka@mff.cuni.cz*

Abstrakt: Cílem práce je prozkoumat problém rozkladu polynomu nad konečným tělesem na součin ireducibilních polynomů. Popsáním několika algoritmů hledajících tento rozklad se ukáže, že tento problém je vždy řešitelný v polynomiálním čase vzhledem ke stupni polynomu a počtu prvků konečného tělesa.

U jednoho z algoritmů je popsána implementace s velmi dobrou asymptotickou časovou složitostí $\mathcal{O}(n^{1.815} \log q)$, kde $n$ je stupeň rozkládaného polynomu nad tělesem s $q$ prvky. Program používající jednodušší, ale prakticky rychlejší variantu tohoto algoritmu je součástí práce.

Klíčová slova: faktorizace, konečná tělesa, polynomy, algoritmus

Title: Factoring polynomials over finite fields

Author: Milan Straka

Department: Department of Algebra

Supervisor: Mgr. Jan Žemlička, Ph.D.

Supervisor's e-mail address: *Jan.Zemlicka@mff.cuni.cz*

Abstract: The goal of this work is to present the problem of the decomposition of a polynomial over a finite field into a product of irreducible polynomials. By describing algorithms solving this problem, we show that the decomposition can always be found in polynomial time in both the degree of the polynomial and the number of elements of the underlying finite field.

One algorithm is studied in detail and an implementation with good asymptotic time complexity $\mathcal{O}(n^{1.815} \log q)$ is described, where $n$ is the degree of the polynomial over the field with $q$ elements. A program using easier, but practically faster version of this algorithm is a part of this work.

Keywords: factorization, finite fields, polynomials, algorithm

# 1. Úvod

Prvočísla zaujímají mezi přirozenými čísly význačné postavení. Jednou z jejich nejdůležitější vlastností je to, že každé přirozené číslo můžeme jednoznačně zapsat jako součin mocnin prvočísel. Tento takzvaný prvočíselný rozklad má navíc tu zajímavou vlastnost, že ho zatím nikdo neumí najít v polynomiálním čase vzhledem k délce zápisu daného čísla, takže se stal základem pro asymetrickou šifru RSA. Přitom otestovat, zda je dané číslo prvočíslem, v polynomiálním čase lze (viz [1]).

Podobnou analogii s prvočísly můžeme nalézt také u polynomů. Některé polynomy jsou podobně jako prvočísla už dále nerozložitelné a ostatní můžeme jednoznačně zapsat jako součin takových nerozložitelných polynomů. Problémy testování nerozložitelnosti a hledání rozkladu polynomů mají ale trochu odlišné vlastnosti než jejich protějšky týkající se prvočísel. Přesto jsou velmi zajímavé a právě pro jejich podobnosti a nepodobnosti se známými prvočíselnými úlohami jsem se rozhodl jimi zabývat.

V této práci se věnuji hlavně hledání rozkladu polynomů nad konečnými tělesy na součin polynomů nerozložitelných. Vzhledem k tomu, že tento rozklad lze efektivně nalézt v polynomiálním čase, můžeme algoritmy na jeho nalezení použít i pro testování rozložitelnosti. Nezabýval jsem se ovšem tím, zda pro samotné testování rozložitelnosti neexistují efektivnější algoritmy.

V následující kapitole popíšu jeden deterministický a jeden pravděpodobnostní algoritmus pro hledání rozkladu polynomů nad konečnými tělesy a dokážu jejich funkčnost. Nebudu se ovšem moc snažit, aby byly tyto algoritmy rychlé, bude stačit jejich polynomialita.

V další kapitole se zaměřím jenom na popisovaný pravděpodobnostní algoritmus a navrhnu variantu, která funguje rychleji než v kvadratickém čase vzhledem ke stupni zadaného polynomu, přesněji v čase $\mathcal{O}(n^{1.815} \log q)$, kde $n$ je stupeň zadaného polynomu nad konečným tělesem o $q$ prvcích.

Součástí této práce je i implementace jednodušší, ale prakticky rychlejší varianty podrobně popisovaného algoritmu. Její tvorbu a vlastnosti popíšu v kapitole poslední.

Při studiu učebnic a článků pro tuto práci jsem byl překvapen zvláštními vlastnostmi vědeckých článků a odborných textů na Internetu. Jednak jsem nedokázal najít v elektronické podobě popisy složitějších algoritmů, které byly v textech pokládány za všeobecně známé. Nyní mám na mysli například násobení polynomů nad konečnými tělesy pomocí FFT, pseudoinverze polynomů, Euklidův algoritmus pro polynomy pracující rychleji než v kvadratickém čase a jiné. Jediný zdroj, který jsem nakonec nalezl, byly „papírové“ učebnice algebry (například [5] a [7]).

Také mě překvapilo, že v několika případech byly různé speciální případy či důkazy ponechány na čtenáři, přestože mi nepřišly nijak jednoduché. Jednalo se například o rychlou bezčtvercovou faktorizaci v případě nenulové charakteristiky, otázku toho, zda v Berlekampově algoritmu jedna báze úplně faktorizuje daný polynom, či důkaz správnosti rychlé varianty stejnostupňové faktorizace. S pomocí učebnic a univerzitních přednášek se mi však všechny tyto problémy povedlo vyřešit.

## Značení

*Konečné těleso* o $q$ prvcích budeme značit $\mathbb{F}_q$. *Polynomem nad konečným tělesem* myslíme posloupnost $(a_0, a_1, \ldots, a_n, \ldots)$ koeficientů z tělesa takovou, že jenom konečně jejích prvků má nenulovou hodnotu. Symbolicky zapisujeme polynom jako $\sum_{i\geq 0} a_i x^i$.

*Stupeň polynomu* $f$, $\deg f$, je největší index $i$ takový, že $a_i$ je nenulový. Pokud jsou všechny koeficienty polynomu rovny nule, jeho stupeň definujme jako nulu, ačkoliv se častěji používá stupeň minus jedna nebo minus nekonečno. My použijeme stupeň nula, aby polynomy stupně nula byly právě všechny prvky tělesa, které používáme.

Pokud definujeme nad polynomy operace plus a minus jako $\sum_{i\geq 0} a_i x^i \pm \sum_{i\geq 0} b_i x^i = \sum_{i\geq 0}(a_i \pm b_i)x^i$ a operaci krát tak, že $\sum_{i\geq 0} a_i x^i \cdot \sum_{i\geq 0} b_i x^i = \sum_{i\geq 0} \sum_{k=0}^{i}(a_k \cdot b_{i-k})x^i$, tvoří polynomy nad konečným tělesem $T$ okruh $T[x]$, přičemž nulový prvek je polynom $\sum_{i\geq 0} 0x^i$ a jednotkový prvek je polynom $1x^0 + \sum_{i\geq 1} 0x^i$.

Polynom $f$ dělí polynom $g$, $f \setminus g$, pokud $g = f \cdot h$ pro nějaký polynom $h$. Všimněme si, že pokud $f \setminus g$ a $t$ je nenulový prvek tělesa, také $t \cdot f \setminus g$, protože $t$ je určitě invertibilní. To nás přivádí k následující definici: Nenulový polynom $f$ nazveme *monický*, pokud je jeho koeficient $a_{\deg f}$ jedničkový prvek tělesa.

*Největším společným dělitelem polynomů $f$ a $g$*, $\mathrm{NSD}(f, g)$, nazveme každý polynom $c$, který dělí $f$ i $g$, a žádný polynom stupně většího než $c$ už nedělí $f$ a $g$ zároveň. Vzhledem k tomu, že jsme v definici použili dělitelnost, je největší společný dělitel přenásobený nenulovým prvkem tělesa také největším společným dělitelem. Pokud bychom chtěli, aby byl největší společný dělitel jednoznačný, můžeme vzít ze všech největších společných dělitelů ten monický. V následujícím textu budeme $\mathrm{NSD}(f, g)$ používat jako označení jediného monického největšího společného dělitele.

V některých místech textu je výhodné dívat se na polynom $\sum_{i=0}^{n-1} a_i x^i$ nad tělesem $\mathbb{F}_q$ jako na $n$-složkový vektor $(a_0, \ldots, a_{n-1})$ nad tělesem $\mathbb{F}_q$. Protože jsou si oba tyto pohledy velmi blízké, budeme je v textu bez zvláštního upozornění zaměňovat.

Pokud máme polynom $f$ z okruhu $\mathbb{F}_q[x]$, můžeme jím tento okruh faktorizovat. Tuto faktorizaci budeme značit jako $\mathbb{F}_q[x]/f$ a myslíme jí faktorizaci okruhu podle kongruence, která odpovídá hlavnímu ideálu určenému polynomem $f$. Pozor na to, že tento pojem faktorizace okruhu nemá nic společného s popisovanou faktorizací polynomů.

Ostatní použité značení je ve shodě s běžně používaným značením a je možné ho nalézt například v knize [5] či [7].

## Formulace problému

Polynom nazveme *ireducibilní*, pokud ho nemůžeme zapsat jako součin dvou netriviálních polynomů. Přesněji řečeno, pokud ho zapíšeme jako součin dvou libovolných polynomů, stupeň jednoho z nich musí být nula. Můžeme také říci, že polynom $f$ je ireducibilní, pokud ho nedělí žádný polynom, jehož stupeň by byl větší než nula a menší než $\deg f$.

Pokud polynom ireducibilní není, jedná se o polynom *složený* a každý polynom menšího stupně, který ho dělí, je jeho *dělitel* neboli *faktor*.

Faktorizací polynomu myslíme jeho rozklad na součin ireducibilních polynomů. Tento rozklad vždy existuje a pokud jsou všechny faktory monické, je až na pořadí jednoznačný.

V textu budeme faktorizací značit nejenom rozklad na součin ireducibilních polynomů, ale také obecný rozklad na součin polynomů daných vlastností. Je dobré si uvědomit, že i takový obecný rozklad vždy existuje (stačí sdružit ireducibilní faktory původního polynomu do skupin stejných vlastností) a pokud můžeme u všech faktorů chtěnou vlastnost jednoznačně určit, je tento obecný rozklad také jednoznačný až na pořadí.

Pokud budeme faktorizací mínit výslovně rozklad na součin ireducibilních polynomů, můžeme ji nazvat úplnou faktorizací.

Cílem algoritmů faktorizace je tedy pro zadaný polynom $f$ nalézt jeho (jednoznačný) rozklad na součin monických ireducibilních polynomů.

# 2. Základní algoritmy

V celém textu budeme předpokládat znalost toho, jak se provádí základní aritmetické operace nad $\mathbb{F}_q$ a $\mathbb{F}_q[x]$. Také použijeme několik základních vět týkajících se konečných těles, které nebudeme dokazovat. Zájemci mohou najít jejich důkazy v [5] nebo [7]. Jedná se především o tato tvrzení:

**Tvrzení 2.1:** Mějme těleso $T$, vzájemně nesoudělné polynomy $m_1, \ldots, m_k$ z okruhu $T[x]$ a položme $m = m_1 \cdots m_k$. Potom je zobrazení $\varphi : T[x]/m \to T[x]/m_1 \times \ldots \times T[x]/m_k$ definované jako

$$\varphi(a) = (a \bmod m_1, \ldots, a \bmod m_k)$$

isomorfismus okruhů.
Toto tvrzení je známé jako *Čínská věta o zbytcích.*

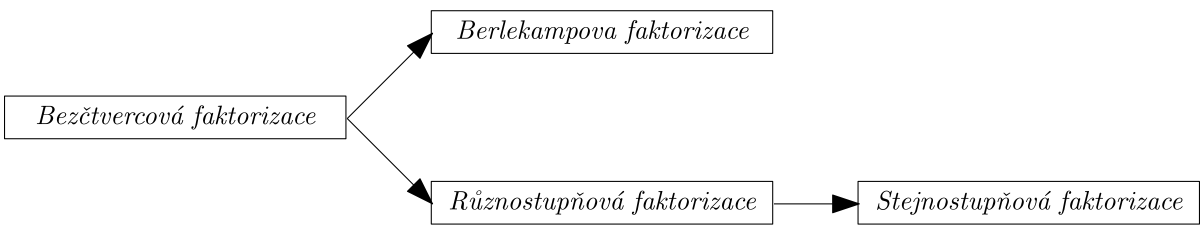
**Tvrzení 2.2:** Buď $m$ prvek $\mathbb{F}_q[x]$. Pak $\mathbb{F}_q[x]/m$ je těleso $\iff$ $m$ je ireducibilní.

**Tvrzení 2.3:** V okruhu polynomů nad $\mathbb{F}_q$ platí $x^q - x = \prod_{s \in \mathbb{F}_q}(x - s)$.

**Tvrzení 2.4:** Polynom $x^{q^d} - x$ nad $\mathbb{F}_q$ je součin všech monických ireducibilních polynomů, jejichž stupeň dělí $d$.

**Tvrzení 2.5:** Multiplikativní grupa nenulových prvků konečného tělesa je cyklická.

Cílem této kapitoly je popsat dva základní algoritmy faktorizace. Ve skutečnosti popíšeme čtyři částečné faktorizace, které se dají pospojovat do dvou odlišných algoritmů úplné faktorizace podle následujícího vzoru:




**Graf 2.6: Závislosti částečných faktorizací**

Nyní začneme popisovat jednotlivé částečné faktorizace.

## Bezčtvercová faktorizace

O polynomu $f$ řekneme, že je *bez čtverců*, pokud neexistuje žádný polynom $g$ stupně alespoň jedna takový, že $g^2$ dělí $f$. Cílem bezčtvercové faktorizace je zadaný polynom $f$ rozložit do tvaru

$$f = \prod_{i=1}^{k} f_i^i$$

tak, aby všechny polynomy $f_i$ byly bez čtverců a každé dva $f_i$ a $f_j$ byly nesoudělné.

Bezčtvercová faktorizace je samozřejmě slabší než faktorizace jako taková, jednotlivé polynomy $f_i$ mohou být ještě složené. Nicméně o nich víme, že nemají opakující se dělitele, což zjednoduší naše algoritmy faktorizace.

Než popíšeme algoritmus bezčtvercové faktorizace, připomeňme si, co je to derivace polynomu. Máme-li $f$ polynom stupně $n$, $f = \sum_{i=0}^{n} a_i x^i$, jeho derivace $f'$ je definována jako

$$f' = \sum_{i=0}^{n-1} (i+1) \cdot a_{i+1} x^i.$$

Navíc platí, že
(1) $(f+g)' = f' + g'$,
(2) $(f \cdot g)' = f' \cdot g + f \cdot g'$,
(3) $(f^n)' = n \cdot f^{n-1} \cdot f'$.

Nejdříve si předvedeme užitečnou charakterizaci bezčtvercových polynomů.

**Tvrzení 2.7:** Polynom $f$ je bez čtverců $\iff \mathrm{NSD}(f, f') = 1$.

**Důkaz:** Nejprve předpokládejme, že polynom $f$ není bez čtverců. V tom případě ho můžeme zapsat jako $f = g^2 h$, přičemž stupeň $g$ je alespoň jedna. Rozepišme si $f' = 2gg'h + g^2 h' = g(2g'h + gh')$. Pak ale $g \setminus f'$ a tedy $g \setminus \mathrm{NSD}(f, f')$. Stupeň $\mathrm{NSD}(f, f')$ je proto alespoň jedna, takže $\mathrm{NSD}(f, f') \neq 1$.

Naopak předpokládejme, že $\mathrm{NSD}(f, f') \neq 1$, a označme $g = \mathrm{NSD}(f, f')$. Stupeň $g$ musí být alespoň jedna. Vezměme ireducibilní rozklad (faktorizaci) polynomu $g$ a označme $h$ libovolný *ireducibilní* polynom stupně alespoň jedna, který dělí $g$. Pak platí, že $f = ha$ a $f' = hb$. Uvažujme nyní

$$hb = f' = (ha)' = h'a + ha',$$

z čehož plyne, že $h \setminus h'a$. Protože je $h$ ireducibilní a stupeň $h'$ je menší než stupeň $h$, musí platit, že $h \setminus a$. Tedy $a = h\hat{a}$, $f = hh\hat{a}$ a polynom $f$ tak není bez čtverců, protože ho dělí $h^2$.

$\boxtimes$

Nyní mějme polynom $f$ s bezčtvercovým rozkladem $f = \prod_{i=1}^{k} f_i^i$ a předpokládejme, že jsme nad tělesem charakteristiky nula. Derivace polynomu $f$ je

$$f' = \sum_{i=1}^{k} f_1 \cdots f_{i-1}^{i-1} \cdot i \cdot f_i^{i-1} \cdot f_i' \cdot f_{i+1}^{i+1} \cdots f_k^k.$$

Označme

$$c_1 = \mathrm{NSD}(f, f') = \prod_{i=2}^{k} f_i^{i-1}.$$

Pokud nyní $f$ a $c_1$ vydělíme, dostaneme

$$g_1 = f/c_1 = \prod_{i=1}^{k} f_i.$$

Nyní stačí položit

$$h_1 = \mathrm{NSD}(c_1, g_1) = \prod_{i=2}^{k} f_i$$

a vidíme, že platí

$$f_1 = g_1/h_1.$$

Pokud bychom chtěli dále získat $f_2$, můžeme aplikovat stejný postup na $c_1 = \prod_{i=2}^{k} f_i^{i-1}$ místo na $f$, a tak pokračovat ve výpočtu všech dalších $f_i$. Pokud si ovšem uvědomíme, že $c_1 = \prod_{i=2}^{k} f_i^{i-1}$, $h_1 = \prod_{i=2}^{k} f_i$ a že pro výpočet $f_2$ potřebujeme polynomy $c_2 = \prod_{i=3}^{k} f_i^{i-2}$ a $g_2 = \prod_{i=2}^{k} f_i$, vidíme, že dokážeme spočítat $c_2$ a $g_2$ jednodušeji:

$$c_2 = c_1/h_1$$
$$g_2 = h_1$$

Nyní už můžeme vytvořit jednoduchý iterativní algoritmus bezčtvercové faktorizace pro tělesa charakteristiky nula.

**procedure** BezčtvercováF($f$) : spočte bezčtvercový rozklad polynomu $f$ nad tělesem charakteristiky 0
**if** deg $f = 0$ **then return** $f$
$c \leftarrow \mathrm{NSD}(f, f')$;   $g \leftarrow f/c$;   $i \leftarrow 0$
**while** deg $g > 0$ **do**
  $i \leftarrow i + 1$
  $h \leftarrow \mathrm{NSD}(c, g)$;   $f_i \leftarrow g/h$
  $g \leftarrow h$;   $c \leftarrow c/h$
**return** $(f_1, \ldots, f_i)$

**Algoritmus 2.8: Bezčtvercová faktorizace pro tělesa charakteristiky 0**

Tělesa charakteristiky nula jsme sice vyřešili, ale všechna konečná tělesa mají charakteristiku nenulovou. Předpokládejme odteď, že máme těleso s prvočíselnou charakteristikou $p$. Právě popsaný algoritmus bezčtvercové faktorizace nebude nyní bohužel správně fungovat. Problém je v tom, že

$$\left(x^{tp}\right)' = tp \cdot x^{tp-1} = 0 \cdot t \cdot x^{tp-1} = 0.$$

Zkusme vysledovat, co přesně právě popsaný algoritmus vypočte, když ho použijeme na těleso charakteristiky $p$. Mějme polynom $f$ s bezčtvercovým rozkladem $\prod_{i=1}^{k} f_i^i$. Označme si ty faktory, které umocňujeme na nějaký násobek $p$, jako $F_j$. Polynom $f$ můžeme tedy zapsat jako $\prod_{i \neq tp} f_i^i \cdot \prod_{j=tp} F_j^j$. Nyní budeme postupovat podle algoritmu a dostaneme, že

$$f' = \left[\prod_{i \neq tp} f_i^i \cdot \prod_{j=tp} F_j^j\right]' = \left[\prod_{i \neq tp} f_i^i\right]' \cdot \prod_{j=tp} F_j^j + \prod_{i \neq tp} f_i^i \cdot \left[\prod_{j=tp} F_j^j\right]' =$$

$$= \left[\prod_{i \neq tp} f_i^i\right]' \cdot \prod_{j=tp} F_j^j + \prod_{i \neq tp} f_i^i \cdot 0 = \left(\sum_{i \neq tp} f_1 \cdots i \cdot f_i^{i-1} \cdot f_i' \cdots f_k^k\right) \cdot \prod_{j=tp} F_j^j.$$

Dále tedy

$$c_1 = \mathrm{NSD}(f, f') = \prod_{i \neq tp} f_i^{i-1} \cdot \prod_{i=tp} F_j^j,$$

$$
\begin{aligned}
g_1 &= f/c_1 = \prod_{i \neq tp} f_i, \\
h_1 &= \mathrm{NSD}(c_1, g_1) = \prod_{i \neq tp, i \geq 2} f_i, \\
f_1 &= g_1/h_1.
\end{aligned}
$$

Zatím všecho funguje. Když budeme pokračovat dál, dostaneme

$$
\begin{aligned}
c_2 &= c_1/h_1 = \prod_{i \neq tp, i \geq 2} f_i^{i-1} \cdot \prod_{i=tp} F_j^j, \\
g_2 &= h_1 = \prod_{i \neq tp, i \geq 2} f_i,
\end{aligned}
$$

a obecně

$$
\begin{aligned}
c_j &= c_{j-1}/h_{j-1} = \prod_{i \neq tp, i \geq j} f_i^{i-1} \cdot \prod_{i=tp} F_j^j, \\
g_j &= h_{j-1} = \prod_{i \neq tp, i \geq j} f_i.
\end{aligned}
$$

Algoritmus bude tím pádem fungovat jenom částečně – určí všechny faktory bezčtvercového rozkladu, jejichž mocnina není dělena charakteristikou tělesa.

Po skončení algoritmu se nám tedy může stát, že $g_k = 1$ a $c_k = \prod_{j=tp} F_j^j \neq 1$. Ukážeme, že v tomto případě musí být polynom $c_k$ $p$-tá mocnina nějakého polynomu.

Nejdříve si všimneme, že $c_k' = 0$, protože mocniny všech faktorů $F_j$ jsou děleny charakteristikou $p$. Polynom $c_k$ si zapišme jako $\sum_i a_i x^i$. Jeho derivace je nulová, takže nenulové koeficienty mohou být jenom ty $a_i$, pro které je $i$ násobek charakteristiky tělesa, čili jenom koeficienty $a_{jp}$. Polynom $c_k$ lze tedy zapsat jako $\sum_i a_{ip} x^{ip}$. Nyní už stačí jenom jednoduchý výpočet

$$
c_k = \sum_i a_{ip} x^{ip} = \left[ \sum_i a_{ip} x^i \right]^p .
$$

Polynom $c_k$ je tedy mocninou nějakého polynomu, dokonce víme, jak tento polynom zkonstruovat. Označme ho $c_k^{1/p}$.

Zbývá tedy vyřešit, jak získat bezčtvercové faktory polynomu $c_k$. Protože je to $p$-tá mocnina polynomu $c_k^{1/p}$, můžeme použít rekurzi – stačí algoritmus spustit na polynom $c_k^{1/p}$ a mocninu všech vrácených faktorů pak vynásobit charakteristikou $p$. Tím vzniká následující algoritmus.

**procedure** BezčtvercováF($q = p^m$,$f$) : spočte bezčtvercový rozklad
polynomu $f$ nad $\mathbb{F}_q$
**if** deg $f = 0$ **then return** $f$
$c \leftarrow \mathrm{NSD}(f, f')$; $g \leftarrow f/c$; $i \leftarrow 0$
**while** deg $g > 0$ **do**
    $i \leftarrow i + 1$
    $h \leftarrow \mathrm{NSD}(c, g)$; $f_i \leftarrow g/h$
    $g \leftarrow h$; $c \leftarrow c/h$
**if** deg $c > 0$ **then** $(f_p, f_{2p}, \ldots, f_{tp}) \leftarrow$ BezčtvercováF$(p^m, c^{1/p})$
**return** $(f_1, \ldots, f_{\max(i,tp)})$

**Algoritmus 2.9: Bezčtvercová faktorizace pro konečná tělesa**

**Tvrzení 2.9:** Algoritmus pracuje správně a v polynomiálním čase vzhledem k $n$.

**Důkaz:** Správnost algoritmu plyne z popisu. Co se týče polynomiality, jeden průchod while cyklem spočítá NSD a dvakrát vydělí polynomy stupně nejvýš $n$. Těchto průchodů může být nejvýš $n$, takže celý algoritmus kromě rekurzivního volání použije $n+1$ NSD a $2n+1$ dělení polynomů, vše pro polynomy stupně nejvýš $n$. To je jistě jen polynomiálně mnoho operací nad tělesem $\mathbb{F}_q$. Označme tento počet $C(n)$.

Co se týče rekurzivního volání, důležité je si všimnout, že rekurzivně voláme algoritmus na polynom, jehož stupeň je nejvýš $n/p$. Označme $T(n)$ počet operací nad $\mathbb{F}_q$ pro dokončení algoritmu pro polynom stupně $n$. Pak

$$T(n) \le C(n) + T(n/p) \le \sum_{i=0}^{\infty} C\left(\frac{n}{p^i}\right) \overset{C(n)\geq n}{\le} \sum_{i=0}^{\infty} \frac{1}{p^i} \cdot C\left(n\right) = \frac{p}{p-1} \cdot C(n) \overset{p\geq 2}{\le} 2C(n).$$

Takže rekurzivní volání na složitosti nepřidá a složitost algoritmu odpovídá složitosti $n$ výpočtů NSD a $n$ dělení polynomů.

$\boxtimes$

## Berlekampův algoritmus

Nyní popíšeme deterministický algoritmus pro faktorizaci bezčtvercového polynomu. Jedná se o jeden z prvních a nejznámějších algoritmů faktorizace, který byl poprvé popsán v [2]. Mějme bezčtvercový polynom $f$ stupně $n$ nad $\mathbb{F}_q$, který chceme rozložit. Označme si $V = \mathbb{F}_q[x]/f$, což je kromě okruhu také vektorový prostor nad $\mathbb{F}_q$ dimenze $n$.

Položme $W = \{v \in V : v^q = v\}$. Pokud ztotožníme prvky $V$ s jejich reprezentanty stupně menšího než $n$, můžeme se na $W$ dívat jako na množinu

$$W = \left\{v \in \mathbb{F}_q[x] : \deg v < n \wedge v^q = v \pmod f\right\}.$$

Tato množina, známá jako *Berlekampova podalgebra*, hraje hlavní roli v celém algoritmu.

**Tvrzení 2.10:** $W$ je podprostor vektorového prostoru $V$.

**Důkaz:** Potřebujeme dokázat uzavřenost na sčítání a skalární násobení. Ať jsou tedy $f, g \in W$. Pak $(f+g)^q = f^q + g^q$, protože kombinační čísla $\binom{q}{i}$ jsou pro všechna $i \in \{1, \ldots, q-1\}$ dělitelná charakteristikou tělesa. Máme tedy $(f+g)^q = f^q + g^q = f + g$, což dokazuje uzavřenost na sčítání.

Nyní mějme $f \in W$ a $c \in \mathbb{F}_q$ a počítejme: $(c \cdot f)^q = c^q \cdot f^q = c \cdot f$, což dokazuje uzavřenost na skalární násobení.

$\boxtimes$

Předpokládejme teď na chvíli, že zadaný polynom $f$ je ireducibilní. $V$ je potom dle (2.2) také těleso. Polynom $y^q - y \in V[y]$ tedy může mít nejvýš $q$ kořenů, přičemž každý prvek tělesa $\mathbb{F}_q$ dle tvrzení (2.3) kořenem je. Kořenů polynomu $y^p - y$ je tím pádem právě $q$ a množinu $W$ můžeme ztotožnit s $\mathbb{F}_q$, tj. s množinou všech polynomů z $V$ stupně nula. Proto je v tomto případě dimenze $W$ rovna jedné.

Už víme, jakou má $W$ dimenzi, když je zadané $f$ ireducibilní. Ale co když není?

**Tvrzení 2.11:** Buď bezčtvercový polynom $f$ součin ireducibilních polynomů $f_1 \cdots f_k$. Pak má $W$ dimenzi $k$.

**Důkaz:** Protože je $f$ bezčtvercový, jsou jeho faktory navzájem nesoudělné. Označíme si $V_i = \mathbb{F}_p[x]/f_i$ a $W_i = \{v \in V_i : v^q = v\}$. Podle Čínské věty o zbytcích (2.1) je zobrazení $\varphi : V \to V_1 \times \ldots \times V_k$ definované jako $\varphi(a) = (a \bmod f_1, \ldots, a \bmod f_k)$ isomorfismus okruhů.

Uvažujme zobrazení $\psi$ definované jako $\varphi$, které ale omezíme na $W$. Pak

1. $\psi : W \to W_1 \times \ldots \times W_k$, protože když $v^q = v \pmod f$, také $v^q = v \pmod{f_i}$.
2. $\psi$ je isomorfismus. Protože je $\psi$ definováno pomocí $\varphi$, je to prostý homomorfismus. Musíme ukázat, že je $\psi$ na.

   Buď $(w_1, \ldots, w_k) \in W_1 \times \ldots \times W_k$. Protože $\varphi$ je na, existuje $v$, že $\varphi(v) = (w_1, \ldots, w_k)$. My chceme ukázat, že $v$ je prvek $W$. Protože

   $$\varphi(v^q) = (w_1^q, \ldots, w_k^q) = (w_1, \ldots, w_k) = \varphi(v)$$

   a protože je $\varphi$ isomorfismus, $v^q = v$, a tedy $v \in W$.

$W$ je tím pádem isomorfní s $W_1 \times \ldots \times W_k$. Protože je ale každý polynom $f_i$ ireducibilní, je každé $V_i$ těleso, a tedy každé $W_i$ je ztotožnitelné s $\mathbb{F}_q$. Tedy $W$ je vektorový prostor nad $\mathbb{F}_q$ dimenze $k$.

$\boxtimes$

Teď tedy víme, jak pomocí $W$ poznat počet ireducibilních faktorů zadaného polynomu. Co kdybychom ale chtěli jednotlivé faktory také najít? K tomu nám bude

**Tvrzení 2.12:** Buď $f$ monický bezčtvercový polynom nad $\mathbb{F}_q$ a $w$ libovolný polynom z $W$ stupně alespoň 1. Pak platí

$$f = \prod_{s \in \mathbb{F}_q} \mathrm{NSD}(w - s, f).$$

**Důkaz:** Předpokládejme, že $f = f_1 \cdots f_k$. Z tvrzení (2.3) víme, že v $\mathbb{F}_q[x]$ platí $x^q - x = \prod_{s \in \mathbb{F}_q} (x - s)$. Proto platí

$$w^q - w = \prod_{s \in \mathbb{F}_q} (w - s).$$

Protože $w^q = w \pmod f$, pro každé $f_i$ platí, že $f_i \setminus w^q - w = \prod_{s \in \mathbb{F}_q} (w - s)$. Jelikož jsou všechny polynomy $(w - s)$ navzájem nesoudělné, existuje pro každý faktor $f_i$ právě jedno $s_i$, že $f_i \setminus w - s_i$. Proto

$$f_i \ \setminus \ \mathrm{NSD}(w - s_i, f),$$

a tím pádem

$$f \ \setminus \ \prod_{i=1}^{k} \mathrm{NSD}(w - s_i, f) \ \setminus \ \prod_{s \in \mathbb{F}_q} \mathrm{NSD}(w - s, f).$$

Naopak každý člen $\mathrm{NSD}(w - s, f)$ dělí $f$, takže máme, že

$$\prod_{s \in \mathbb{F}_q} \mathrm{NSD}(w - s, f) \ \setminus \ f.$$

Proto se polynomy $f$ a $\prod_{s \in \mathbb{F}_q} \mathrm{NSD}(w - s, f)$ navzájem dělí, z čehož plyne, že jeden je konstantní násobek druhého. Oba jsou ovšem monické, proto musí být shodné.

$\boxtimes$

Toto tvrzení nám dává návod, jak lze polynom $f$ faktorizovat. Vezmeme-li $w \in W$, každý výraz $(w - s)$ má stupeň menší než $f$ a faktorizace $f = \prod_{s \in \mathbb{F}_q} \mathrm{NSD}(w - s, f)$ proto obsahuje alespoň dva netriviální faktory polynomu $f$. Ty pak můžeme dále faktorizovat použitím jiných vektorů z $W$. Zkoušet všechny vektory ve $W$ ale není dobrý nápad, protože jich je $q^k$. Jenomže $W$ je vektorový prostor, a proto nám stačí použít $k$ lineárně nezávislých vektorů pro úplnou faktorizaci polynomu $f$.

Rádi bychom tedy získali bázi $W$, která nám poslouží jak pro stanovení počtu faktorů zadaného polynomu $f$, tak pro jejich získání.

Mějme polynom $f$ stupně $n$. Víme, že $W = \left\{ v \in \mathbb{F}_q[x] : \deg v < n \wedge v^q = v \pmod f \right\}$. To můžeme zapsat jako

$$\begin{aligned}
W &= \left\{ v = (v_0, \ldots, v_{n-1}) : \left[ \sum_{i=0}^{n-1} v_i x^i \right]^q = \sum_{i=0}^{n-1} v_i x^i \pmod f \right\} = \\
&= \left\{ v = (v_0, \ldots, v_{n-1}) : \sum_{i=0}^{n-1} v_i x^{iq} = \sum_{i=0}^{n-1} v_i x^i \pmod f \right\} = \\
&= \left\{ v = (v_0, \ldots, v_{n-1}) : \sum_{i=0}^{n-1} v_i (x^{iq} \bmod f) = \sum_{i=0}^{n-1} v_i x^i \right\} = \\
&= \left\{ v = (v_0, \ldots, v_{n-1}) : \sum_{i=0}^{n-1} v_i (x^{iq} \bmod f) - \sum_{i=0}^{n-1} v_i x^i = 0 \right\}.
\end{aligned}$$

Pokud označíme $I$ jako jednotkovou matici velikosti $n \times n$ a $Q$ jako matici $n \times n$, jejíž sloupce tvoří koeficienty vektorů $x^0 \bmod f, x^q \bmod f, \ldots, x^{(n-1)q} \bmod f$, můžeme přepsat definici $W$ na

$$W = \left\{ v = (v_0, \ldots, v_{n-1}) : 0 = Qv^\top - Iv^\top = (Q - I)v^\top \right\}.$$

Odtud vidíme, že $W$ je nulový prostor matice $Q - I$, jehož bázi můžeme jednoduše spočítat pomocí Gaussovy eliminace. Shrňme všechny získané informace do následujícího

**Tvrzení 2.13:** Buď $f$ bezčtvercový polynom stupně $n$ nad $\mathbb{F}_q$, $I_{n\times n}$ jednotková matice a $Q_{n\times n}$ matice, jejíž sloupce tvoří koeficienty polynomů $x^0 \bmod f, x^q \bmod f, x^{2q} \bmod f, \ldots$ $\ldots, x^{(n-1)q} \bmod f$. Označme bázi nulového prostoru matice $Q - I$ jako $\{w_1, \ldots, w_k\}$. Pak

1. počet prvků báze, $k$, odpovídá počtu ireducibilních faktorů v rozkladu $f$,
2. protože $1^q = 1$, je $(1, 0, \ldots, 0) \in W$, a tedy BÚNO $w_1 = (1, 0, \ldots, 0)$,
3. pochopíme-li vektory $w_2, \ldots, w_k$ jako polynomy, platí $f = \prod_{s\in\mathbb{F}_q} \mathrm{NSD}(w_i - s, f)$.

$\boxtimes$

**procedure** BerlekampovaF($q$,$f$) : faktorizuje bezčtvercový $f \in \mathbb{F}_q[x]$
$Q \leftarrow$ matice $n \times n$ se sloupci $x^0 \bmod f, x^q \bmod f, \ldots, x^{(n-1)q} \bmod f$
$((1, 0, \ldots, 0), w_2, \ldots, w_k) \leftarrow$ báze nulového prostoru matice $Q - I$
$i \leftarrow 2$
*faktory* $\leftarrow \{f\}$
**while** $|$*faktory*$| < k$ **do**
    **foreach** $g \in$ *faktory* **do**
        **foreach** $s \in \mathbb{F}_q$ **do**
            $c \leftarrow \mathrm{NSD}(w_i - s, g)$
            **if** $\deg c > 0 \wedge \deg c < \deg g$ **then**
                *faktory* $\leftarrow$ *faktory* $\setminus \{g\} \cup \{c, g/c\}$
                $g \leftarrow g/c$
    $i \leftarrow i + 1$
**return** *faktory*

**Algoritmus 2.14: Berlekampova faktorizace bezčtvercového polynomu**

**Tvrzení 2.14:** Algoritmus pracuje správně a v polynomiálním čase vzhledem k $n$ a $q$.

**Důkaz:** Časová složitost nejvnitřnějšího cyklu je stejná jako jeden NSD a jedno dělení. Počet opakování cyklů v pořadí od nejvnitřnějšího je $q$-krát pro $s \in \mathbb{F}_q$, dále $n$-krát pro $g \in$ *faktory* a $n$-krát pro vnější cyklus. Časová složitost všech cyklů algoritmu je tedy v nejhorším $qn^2$-krát složitost nalezení NSD a dělení.

Co se týče efektivního vytvoření matice $Q$, nejprve spočteme $x^q \bmod f$ a každý další řádek $x^{iq} \bmod f$ získáme jako $x^{(i-1)}q \cdot x^q \bmod f$, protože oba tyto polynomy už známe. To vše zvládneme pomocí $q + n$ násobení polynomů. Ještě chybí složitost nalezení báze nulového prostoru, což zvládneme Gaussovou eliminací v čase $n^3$.

$\boxtimes$

Nepříjemná je závislost na $q$, tj. na počtu prvků použitého tělesa. Popsaný algoritmus je sice možné modifikovat na pravděpodobnostní, který je v průměru polynomiální vzhledem k $n$ a $\log q$ (je popsán například v [3]), ale potřeba hledání báze nulového prostoru celý algoritmus velmi brzdí. Proto dále popíšeme jiný pravděpodobnostní algoritmus, jehož průměrný čas bude také polynomiální vzhledem k $n$ a $\log q$ a který Gaussovu eliminaci nebude muset provádět.

Pro úplnou korektnost bychom ještě měli dokázat, že každá báze $W$ úplně faktorizuje polynom $f$, tj. že oddělí každé dva jeho faktory. Mějme tedy BÚNO $f_1$ a $f_2$ faktory $f$. Nejprve nahlédneme, že vůbec existuje polynom $w \in W$, který je oddělí: podle Čínské věty o zbytcích (2.1) existuje právě jeden polynom $w$ takový, že $w \bmod f_1 = 1$ a pro ostatní $f_i$ je $w \bmod f_i = 0$. Protože $w^q = w \pmod{f_i}$ pro každé $f_i$, dle stejné věty je $w \in W$. Tento polynom oddělí $f_1$ a $f_2$, jelikož $f_1 \backslash w - 1$ a $f_2 \backslash w$, a můžeme ho zapsat jako $w = \sum_{i=1}^{k} a_i w_i$.

Nyní pro spor předpokládejme, že žádný prvek $w_1, \ldots, w_k$ faktory $f_1$ a $f_2$ neoddělil, takže ke každému $w_i$ existuje konstanta $s_i$, že $f_1 f_2 \backslash w_i - s_i$. Pak ale $f_1 f_2 \backslash \sum_{i=1}^{k} a_i(w_i - s_i) = w - \sum_{i=1}^{k} a_i s_i$, což je ve sporu s tím, že polynom $w$ oddělí $f_1$ a $f_2$.

## Různostupňová faktorizace

Nyní popíšeme další algoritmus faktorizace bezčtvercového polynomu. Tento algoritmus má dvě části. První je *různostupňová faktorizace,* jejímž cílem je rozložit bezčtvercový polynom $f$ na $\prod_i f_i$, kde každý polynom $f_i$ je součin všech ireducibilních faktorů polynomu $f$, jejichž stupeň je $i$.

Druhá fáze, *stejnostupňová faktorizace*, má za úkol rozkládat bezčtvercový polynom, jehož faktory mají stejný stupeň. Tuto faktorizaci použijeme na každé netriviální $f_i$, které vznikne při různostupňové faktorizaci.

Nejprve si popíšeme různostupňovou faktorizaci. Tato faktorizace, která je popsaná například v [9], je ještě deterministická. Náhodu budeme muset použít až při stejnostupňové faktorizaci.

Připomeňme si tvrzení (2.4). To tvrdí, že $x^{q^d} - x$ je součinem všech monických ireducibilních polynomů nad $\mathbb{F}_q$, jejichž stupeň dělí $d$. Toto tvrzení nám dává jasný návod, jak můžeme různostupňovou faktorizaci provést. Protože $x^q - x$ je součin všech ireducibilních polynomů stupně jedna, určitě je $f_1 = \mathrm{NSD}(x^q - x, f)$. Navíc v rozkladu polynomu $f/f_1$ už žádné polynomy stupně jedna nejsou, takže $f_2 = \mathrm{NSD}(x^{q^2} - x, f/f_1)$ a stejnou úvahou dostaneme, že

$$f_i = \mathrm{NSD}\left(x^{q^i} - x, \frac{f}{\prod_{j=1}^{i-1} f_j}\right).$$

Protože ale $\mathrm{NSD}(a, b) = \mathrm{NSD}(a - b, b)$, můžeme tuto rovnost přepsat na použitelnější

$$f_i = \mathrm{NSD}\left(x^{q^i} - x \bmod f, \frac{f}{\prod_{j=1}^{i-1} f_j}\right).$$

**procedure** RůznostupňováF($q$,$f$) : různostupňově faktorizuje bezčtvercový polynom $f$ nad tělesem $\mathbb{F}_q$
$i \leftarrow 0$; $w \leftarrow x$;
**while** deg $f > 0$ **do**
  $i \leftarrow i + 1$; $w \leftarrow w^q \bmod f$
  $f_i \leftarrow \mathrm{NSD}(w - x, f)$
  $f \leftarrow f/f_i$
**return** $(f_1, f_2, \ldots, f_i)$

**Algoritmus 2.15: Různostupňová faktorizace**

**Tvrzení 2.15:** Algoritmus pracuje správně a v polynomiálním čase vzhledem k $n$ a $\log q$.

**Důkaz:** Jediná změna algoritmu od popsaného postupu je postupné počítání polynomu $x^{q^i} - x$. Algoritmus využívá toho, že $x^{q^i} = (x^{q^{i-1}})^q$. V $i$-tém průchodu cyklem je tedy $w = x^{q^i}$, $w - x$ je požadovaný polynom a algoritmus proto pracuje správně.

Co se časové složitosti týče, jeden průchod cyklem vyžaduje spočítání jednoho NSD, dále $\log q$ násobení polynomů pro vypočítání $w^q$ a jedno dělení, které můžeme zanedbat, protože výpočet NSD je náročnější. Celý cyklus se může provádět až $n$-krát, takže celková složitost je omezena složitostí $n$ NSD a $n \log q$ násobení.

$\boxtimes$

## Stejnostupňová faktorizace

Úkolem stejnostupňové faktorizace je rozložit zadaný bezčtvercový polynom, jehož ireducibilní faktory mají všechny stupeň $d$, přičemž $d$ známe. Jedna z variant této faktorizace je popsána v [3].

Nejprve potřebujeme trochu teorie o odmocninách.

**Tvrzení 2.16:** Buď $T$ konečné těleso o lichém počtu prvků a $S$ cyklická multiplikativní grupa nenulových prvků tělesa $T$, přičemž $S$ je sudého řádu $k$ a má generátor $g$. Pak

(1) polovina prvků $S$ nemá druhou odmocninu a druhá polovina má přesně dvě,

(2) jednička druhé odmocniny má a jsou to jednička a minus jednička,

(3) má-li $a$ druhé odmocniny, je $a^{k/2} = 1$, pokud nemá, je $a^{k/2} = -1$.

**Důkaz:** Vezměme sudou mocninu generátoru $g^{2i}$. Určitě platí $(g^i)^2 = g^{2i}$ a také $(g^i \cdot g^{k/2})^2 = g^{2i} \cdot g^k = g^{2i}$. Takže každá sudá mocnina generátoru, čili polovina prvků, má alespoň dvě odmocniny a všechny tyto odmocniny jsou navzájem různé. Žádné další odmocniny ale nejsou, když už jsme jich našli $k$, takže bod (1) platí.

Co se týče odmocniny z jedničky, určitě platí, že $1^2 = 1$ a $(-1)^2 = 1$, takže jsme našli dvě odmocniny z jedničky. Žádné další už být nemohou.

Pro důkaz posledního bodu si nejprve uvědomíme, že jak $g^0 = 1$, tak $g^{k/2}$ jsou odmocniny z jedničky. Tedy $g^{k/2} = -1$. Nyní vezměme prvek $a$, který má druhé odmocniny. Pak je to sudá mocnina generátoru, čili $a = g^{2i}$. Proto $a^{k/2} = (g^{2i})^{k/2} = g^{ik} = (g^k)^i = (g^0)^i = 1$. Naopak je-li $a$ prvek, který nemá druhé odmocniny, je to lichá mocnina generátoru. Takže $a^{k/2} = (g^{2i+1})^{k/2} = g^{ik+k/2} = (g^k)^i \cdot g^{k/2} = 1 \cdot (-1) = -1$.

Ještě poznamenejme, že celý důkaz jsme mohli provést pro libovolnou cyklickou multiplikativní grupu sudého řádu, jenom bychom místo nedefinované $-1$ museli psát $g^{k/2}$.

$\boxtimes$

Mějme bezčtvercový polynom $f$ stupně $k \cdot d$, který se skládá z $k$ ireducibilních polynomů stupně $d$, $f = f_1 \cdots f_r$. Pokud je $k = 1$, je faktorizace $f$ triviální, takže předpokládejme, že $k \geq 2$. Podle Čínské věty o zbytcích (2.1) je zobrazení

$$\varphi : \mathbb{F}_q[x]/f \to \mathbb{F}_q[x]/f_1 \times \ldots \times \mathbb{F}_q[x]/f_k$$
$$\varphi(a) = (a \bmod f_1, \ldots, a \bmod f_k) = (\varphi_1(a), \ldots, \varphi_k(a))$$

izomorfismus okruhů.

Platí, že $f_i$ dělí $a$ právě tehdy, když $0 = a \bmod f_i = \varphi_i(a)$. Pokud tedy dokážeme najít polynom $a$, pro který bude nějaké $\varphi_i(a) = 0$ a nějaké $\varphi_j(a) \neq 0$, bude $\mathrm{NSD}(a, f)$ netriviální dělitel $f$. Pomocí něho můžeme $f$ rozdělit na dva faktory a pokud jsou tyto netriviální, použít na ně jiný vhodný polynom $a$.

Jak ale najdeme polynom $a$, který bude splňovat popsané podmínky? Zde přichází na řadu náhoda. Pokud zvolíme $a$ jako náhodný polynom stupně menšího než $f$, budou všechny $\varphi_i(a)$ podle Čínské věty o zbytcích také náhodné prvky těles $\mathbb{F}_q[x]/f_i$. Chtěli bychom, aby nějaké $\varphi_i(a) = 0$ a nějaké $\varphi_j(a) \neq 0$.

Na chvíli předpokládejme, že jsme nad konečným tělesem, jehož charakteristika je liché prvočíslo (není to tedy 2). Každé těleso $\mathbb{F}_q[x]/f_i$ má pak lichý počet prvků a multiplikativní grupa prvků tohoto tělesa bez nuly má sudý počet prvků a je dle (2.5) cyklická. Splňuje tedy předpoklady posledního tvrzení (2.16), takže $\varphi_i(a)^{(q^d-1)/2}$ je buď jedna nebo minus jedna a obě tyto možnosti jsou stejně pravděpodobné.

Zvolme tedy náhodný polynom $a$ stupně menšího než $f$. Polynomy $\varphi_i(a)$ jsou pak náhodné a $\varphi_i(a)^{(q^d-1)/2} - 1 = \varphi_i(a^{(q^d-1)/2} - 1)$ je buď 0 nebo $-2$, přičemž obě tyto možnosti jsou stejně pravděpodobné. Polynom $a$ nám nevyhovuje, když jsou všechny $\varphi_i = 0$ nebo když jsou všechny $\varphi_i = -2$. To lze zapsat tak, že nám nevyhovuje, když $\varphi_1(a^{(q^d-1)/2} - 1) = \ldots = \varphi_k(a^{(q^d-1)/2} - 1)$, což nastává s pravděpodobností $2 \cdot (1/2)^k = 2^{1-k} \leq 1/2$.

Algoritmus bude následovný: vygeneruj náhodný polynom $a$ stupně menšího než $f$ a spočítej $b = a^{(q^d-1)/2}$ a $c = \mathrm{NSD}(b-1, f)$. Pokud $c$ není netriviální dělitel $f$ (což nastane s pravděpodobností $2^{1-k} \leq 1/2$), zvol jiný polynom $a$. V opačném případě rekurzivně zpracuj polynomy $c$ a $f/c$.

Ještě než algoritmus zapíšeme formálně a rozebereme jeho složitost, musíme vyřešit poslední problém, a to tělesa s charakteristikou 2, na která se tvrzení (2.16) nevztahuje. Toto tvrzení nám pro tělesa s lichou charakteristikou dává do ruky zobrazení, které pro všechny prvky nabývá pouze dvou hodnot, a každá tato hodnota je stejně pravděpodobná. Rádi bychom něco podobného našli také pro tělesa charakteristiky 2. Poslouží nám

**Tvrzení 2.17:** Mějme konečné těleso $\mathbb{F}_{2^k}$ a definujme $T_k = \sum_{i=0}^{k-1} x^{2^i}$. Pak
(1) $x^{2^k} - x = T_k(T_k + 1)$,
(2) přesně pro polovinu prvků $a \in \mathbb{F}_{2^k}$ je $T_k(a) = 0$, pro druhou polovinu je $T_k(a) = 1$.

**Důkaz:** Začněme bodem (1):

$$T_k(T_k+1) = T_k^2 + T_k = \left[\sum_{i=0}^{k-1} x^{2^i}\right]^2 + \sum_{i=0}^{k-1} x^{2^i} = \sum_{i=0}^{k-1} \left(x^{2^i}\right)^2 + \sum_{i=0}^{k-1} x^{2^i} =$$
$$= \sum_{i=0}^{k-1} x^{2^{i+1}} + \sum_{i=0}^{k-1} x^{2^i} \overset{\text{charakteristika 2}}{=} \sum_{i=1}^{k} x^{2^i} - \sum_{i=0}^{k-1} x^{2^i} = x^{2^k} - x.$$

Polynom $x^{2^k} - x = T_k(T_k+1)$ se ale nad $\mathbb{F}_{2^k}[x]$ dle (2.3) rozkládá jako $\prod_{s \in \mathbb{F}_{2^k}} (x - s)$, což znamená, že hodnota $x^{2^k} - x$ je po dosazení libovolného prvku $s \in \mathbb{F}_{2^k}$ nulová. Takže součin $T_k(T_k + 1)$ musí být po dosazení prvku $s$ také nulový. To je ale možné jenom v případě, že $T_k(s) = 0$ nebo $T_k(s) = 1$.

Ještě potřebujeme ukázat, že obě možnosti jsou stejně pravděpodobné. Uvažujme znovu rozklad $x^{2^k} - x = T_k(T_k + 1)$. Pokud je $T_k(a) = 0$, je $a$ kořenem polynomu $T_k$. V opačném případě je $a$ kořenem polynomu $T_k - 1$. Oba tyto polynomy mají stupeň $2^{k-1}$, každý z nich může mít tedy nejvýš $2^{k-1}$ kořenů. Prvků $a$, se kterými pracujeme, je ale $2^k$, což znamená, že přesně polovina z nich musí být kořeny polynomu $T_k$ a druhá polovina kořeny polynomu $T_k - 1$.

$\boxtimes$

Teď už víme, jak se postarat i o tělesa s charakteristikou 2, takže se můžeme pustit do algoritmu.

**procedure** StejnostupňováF($q = p^m$,$f$,$d$) : stejnostupňově faktorizuje bezčtvercový polynom $f$ nad $\mathbb{F}_q$
**if** deg $f \leq d$ **then return** $\{f\}$
**do**
    $a \leftarrow$ náhodný polynom nad $\mathbb{F}_q$ stupně menšího než deg $f$
    **if** $p = 2$ **then** $b \leftarrow \sum_{i=0}^{md-1} a^{2^i}$ mod $f$ **else** $b \leftarrow a^{(q^d-1)/2} - 1$ mod $f$
    $c \leftarrow \mathrm{NSD}(b, f)$
**while** $c = 1 \vee c = f$
**return** StejnostupňováF($p^m, c, d$) $\cup$ StejnostupňováF($p^m, f/c, d$)

**Algoritmus 2.18: Stejnostupňová faktorizace**

**Tvrzení 2.18:** Algoritmus pracuje správně a v polynomiálním čase vzhledem k $n$ a $\log q$.

**Důkaz:** Tentokrát je algoritmus přímočarou implementací popsaného postupu, jehož správnost už máme dokázanou.

Nejprve si spočítejme, kolik polynomů $a$ musíme v průměru otestovat, než najdeme nějaký užitečný. Šance, že jeden polynom $a$ není užitečný, je nejhůř 1/2. Očekávaný počet polynomů, které musíme vyzkoušet, než najdeme jeden úspěšný, je $z = 1/2 \cdot 1 + 1/2 \cdot (1+z)$, z čehož dostaneme $z = 2$.

Zjištění, zda je polynom $a$ pro faktorizaci užitečný, zvládneme jedním NSD a $n \log q$ násobeními. V průměru to musíme udělat dvakrát, což náš odhad nijak nezkazí.

Známe složitost algoritmu bez rekurzivního volání. Jak se nám na složitosti podepíše rekurze? Nechť $T(n)$ je složitost algoritmu pro polynom stupně $n$ a $C(n)$ je složitost algoritmu bez rekurzivního volání, tedy složitost nalezení užitečného polynomu $a$. V nejhorším případě platí, že

$$T(n) = C(n) + T(n-1) + T(1) = C(n) + C(n-1) + \ldots + C(1) + n \cdot T(1) \leq n \cdot C(n) + n.$$

Složitost celého algoritmu tedy můžeme odhadnout jako spočtení $n$ NSD a $n^2 \log q$ násobení. Poznamenejme jenom, že tato analýza rekurze je velmi hrubá, dá se dokázat, že v průměru je $T(n) \leq 2 \log n \cdot C(n)$, což dokážeme při analýze algoritmu stejnostupňové faktorizace v další kapitole. Můžeme tím zpřesnit náš odhad složitosti na složitost provedení $2 \log n$ NSD a $2n \log n \log q$ násobení.

$\boxtimes$

# 3. Subkvadratický algoritmus

Cílem této kapitoly je popsat implementace bezčtvercové, různostupňové a stejnostupňové faktorizace, které mají co nejmenší časovou složitost. Abychom vůbec mohli o časové složitosti mluvit, musíme říct, co jí budeme mínit.

Pro naše potřeby použijeme zjednodušenou definici. *Časovou složitostí* algoritmu, který pracuje s polynomy nad tělesem $\mathbb{F}_q$, budeme rozumět počet elementárních operací s prvky tělesa $\mathbb{F}_q$, které musí algoritmus vykonat. Budeme ji vyjadřovat jako funkci velikosti vstupu, nejčastěji budeme používat $n$, stupeň zpracovávaného polynomu, a $q$, počet prvků tělesa, nad kterým pracujeme. Budeme používat složitost v *nejhorším případě*, takže časová složitost udává největší použitý počet operací, který je třeba pro zpracování libovolného korektního vstupu dané délky.

Určit přesnou časovou složitost je ale docela náročné, takže budeme pro jednoduchost pracovat pouze s *asymptotickou* časovou složitostí. Řekneme, že funkce $f : \mathbb{N} \to \mathbb{N}$ je $\mathcal{O}(g)$, pokud existují kladné konstanty $c$ a $n_0$ takové, že $\forall n \geq n_0 : f(n) \leq c \cdot g(n)$. Časovou složitost pak budeme vyjadřovat pomocí zavedeného symbolu $\mathcal{O}$.

Naše definice bere v úvahu jenom operace prováděné nad tělesem, ale nepočítá cykly, testování podmínek, volání procedur ani jinou režii programu. U všech popsaných algoritmů lze však jednoduše nahlédnout, že je tato režie vzhledem k počtu operací nad tělesem zanedbatelná.

Nyní si popíšeme úpravy, které budeme s asymptotickými složitostmi často provádět. Nejprve si uvědomíme, že pokud používáme v asymptotické složitosti logaritmus, nezáleží na jeho základu. Protože

$$\log_a b = \frac{\log_c b}{\log_c a},$$

všechny logaritmy se liší jenom o konstantu a ta se „do $\mathcal{O}$-čka schová“.

Dále si uvědomíme, že $\mathcal{O}(\log n)$ je $\mathcal{O}(n^\varepsilon)$ pro libovolné kladné $\epsilon$. To bude platit, pokud $\log n \leq n^\varepsilon$. To ale můžeme pro dostatečně velká $n$ přepsat na $\log \log n \leq \varepsilon \cdot \log n$. Pokud by vpravo nebyla konstanta $\varepsilon$, daná nerovnost pro dostatečně velká $n$ jistě platí. Konstantu $\varepsilon$ ale můžeme schovat do $\mathcal{O}$-čka, takže opravdu $\mathcal{O}(\log n)$ je $\mathcal{O}(n^\varepsilon)$. Tuto skutečnost budeme zapisovat jako $\mathcal{O}(\log n)$ je $\mathcal{O}(n^{\sigma(1)})$, kde $\sigma(1)$ značí libovolně malé kladné číslo.

Abychom mohli určovat časovou složitost co nejpřesněji, musíme si říci, jak rychle dokážeme provádět aritmetické operace s polynomy nad $\mathbb{F}_q$. Uvedené algoritmy lze najít například v knize [5]. Budou nás zajímat následující operace:

- *sčítání* a *odčítání* polynomů stupně nejvýše $n$ dokážeme pomocí $\mathcal{O}(n)$ operací nad $\mathbb{F}_q$. Stačí nám k tomu klasický školní algoritmus.
- *násobení* dvou polynomů stupně nejvýše $n$ dokážeme provést pomocí
  - $\mathcal{O}(n^2)$ operací nad $\mathbb{F}_q$ použitím klasického školního algoritmu,
  - $\mathcal{O}(n^{\log_2 3})$ operací nad $\mathbb{F}_q$ pomocí algoritmu *Karatsuba & Ofman*,
  - $\mathcal{O}(n \log n \log \log n)$ operací nad $\mathbb{F}_q$ užitím algoritmů založených na FFT. Jde kupříkladu o algoritmy *Schönhage & Strassen* nebo *Cantor & Kaltofen*.

  Protože budeme asymptotickou složitost násobení potřebovat často, položíme $M(n) = n \log n \log \log n$. Též mnohokrát využijeme faktu $\sum_i M(n_i) \leq M(\sum_i n_i)$, který plyne z $M(n) \geq n$.
- *dělení se zbytkem* dvou polynomů stupně nejvýš $n$ můžeme provést v čase odpovídajícím času čtyř násobení (použitím *Newtonovy iterace*). Dostáváme se tedy také na čas $\mathcal{O}(M(n))$.
- *umocnění* polynomu stupně $n$ na $d \in \mathbb{N}$ dokážeme provést binárním umocňováním pomocí $\mathcal{O}(\log d)$ násobení, tedy v čase $\mathcal{O}(M(n) \log d)$.

- *největší společný dělitel* polynomů stupně nanejvýš $n$ můžeme hledat
  - modulárním Euklidovým algoritmem, z čehož vyjde složitost $\mathcal{O}(n^2)$,
  - složitým rekurzivním postupem, jehož složitost je $\mathcal{O}(M(n)\log n)$.
- *vyhodnocení* polynomu stupně $n$ v jednom bodě dokážeme pomocí tzv. *Hornerova schématu* pomocí $\mathcal{O}(n)$ operací. Stačí si všimnout, že hledaná hodnota je rovna

$$\sum_{i=0}^{n} a_i \cdot x^i = \overbrace{(\cdots(((}^{n} a_n)x + a_{n-1})x + a_{n-2})\cdots)x + a_0,$$

  což už dokážeme spočítat pomocí $n$ násobení a $n$ sčítání prvků tělesa $\mathbb{F}_q$.
- *vyhodnocení* polynomu stupně $n$ v $m$ bodech bychom dokázali vyřešit opakováním Hornerova schématu v čase $\mathcal{O}(nm)$, nicméně jde vymyslet efektivnější rekurzivní algoritmus se složitostí $\mathcal{O}(M(m)\log m + M(n))$.
- získat *zbytky po dělení* jednoho polynomu stupně $n$ několika polynomy dokážeme pomocí opakovaného dělení. Pokud je ale stupeň součinu polynomů, kterými chceme dělit, roven $m$, stejně jako v minulé operaci můžeme použít (dokonce velmi podobný) rekurzivní algoritmus se složitostí $\mathcal{O}(M(m)\log m + M(n))$.

Mimo polynomů budeme muset násobit také matice, proto následuje

**Tvrzení 3.1:** Buď $A$ a $B$ matice o velikostech $n \times n$. Označme $\omega$ nejmenší exponent takový, že dokážeme zadané matice vynásobit v čase $\mathcal{O}(n^\omega)$. Tvrdíme, že $\omega < 2.376$, a předpokládáme $\omega > 2$.
Důkaz tohoto tvrzení je velice komplikovaný a lze ho nalézt v *Coppersmith & Winograd* [4]. ⊠

Kromě již popsaných operací budeme používat ještě *skládání polynomů*, $g(h) \bmod f$, jehož výsledkem je $\sum_i g_i h^i \bmod f$. Pomocí Hornerova schématu ho dokážeme provést pomocí $\mathcal{O}(n \cdot M(n))$ operací nad $\mathbb{F}_q$. Nicméně pomocí rychlého násobení matic ho dokážeme provést ještě rychleji.

**procedure** SkládáníPolynomů($q$,$f$,$g$,$h$) : spočte $g(h) \bmod f$
$n \leftarrow \deg f$;     $m \leftarrow \lceil \sqrt{n}\, \rceil$
Najdi $g_0, \ldots, g_{m-1}$ stupně nejvýš $m$, aby $g = \sum_{i=0}^{m-1} g_i x^{mi}$
$h_0 \leftarrow 1$;     **for** $1 \le i \le m$ **do** $h_i \leftarrow h_{i-1} \cdot h \bmod f$
Vytvoř matici $G$ velikosti $m \times m$, jejíž řádky jsou $g_0, \ldots, g_{m-1}$
Vytvoř matici $H$ velikosti $m \times n$, jejíž řádky jsou $h_0, \ldots, h_{m-1}$
$B \leftarrow G \cdot H$, řádky této matice označ jako $b_0, \ldots, b_{m-1}$
**return** $\sum_{i=0}^{m-1} b_i \cdot (h^m)^i \bmod f$

**Algoritmus 3.2: Skládání polynomů pomocí maticového násobení**

**Tvrzení 3.2:** Jsou-li stupně $f, g, h$ nejvýš $n$, vydá algoritmus správný výsledek v čase $\mathcal{O}(n^{(\omega+1)/2})$. Použitím odhadu pro $\omega$ dostaneme $\mathcal{O}(n^{1.688})$.

**Důkaz:** Činnost algoritmu můžeme rozepsat jako následující maticové násobení.

$$\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{m-1} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ h \bmod f \\ \vdots \\ h^{m-1} \bmod f \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix}$$

Vzhledem k vlastnostem maticového násobení určitě platí, že $b_i = g_i(h) \bmod f$. Celý výsledek algoritmu je tedy $\sum_{i=0}^{m-1} b_i \cdot (h^m)^i \bmod f = \sum_{i=0}^{m-1} g_i(h) \cdot h^{mi} \bmod f = g(h) \bmod f$.

Nyní určíme složitost. Nalezení polynomů $g_i$ zvládneme v lineárním čase, $h_i$ dokážeme spočítat pomocí $m$ násobení, čili v čase $\mathcal{O}(\sqrt{n}M(n))$. Poté musíme vynásobit matice o rozměrech $m \times m$ a $m \times n$. To můžeme udělat tak, že provedeme $m$ násobení matic o rozměrech $m \times m$, čili v čase $\mathcal{O}(mm^\omega) = \mathcal{O}(n^{(\omega+1)/2})$. Závěrečnou sumu můžeme vyhodnotit Hornerovým schématem, což nás stojí opět $\mathcal{O}(\sqrt{n}M(n))$ operací nad $\mathbb{F}_q$. Pokud předpokládáme $\omega > 2$, nejnáročnější je právě maticové násobení, takže složitost celého algoritmu je slibovaných $\mathcal{O}(n^{(\omega+1)/2})$.

$\boxtimes$

Ještě popíšeme vylepšení právě popsaného skládání, které bude efektivnější, budeme-li skládat víc než jeden polynom.

**procedure** SkládáníMnoha$(q, f, g_1, \ldots, g_k, h)$ : spočte všechna $g_i(h)$ mod $f$
$n \leftarrow \deg f$; $\quad m \leftarrow \lceil\sqrt{nk}\rceil$
Pro každý $g_i$ najdi $g_{i,0}, \ldots, g_{i,n/m-1}$ stupně nejvýš $m$, aby $g_i = \sum_{j=0}^{n/m-1} g_{i,j}x^{mj}$
$h_0 \leftarrow 1$; $\quad$ **for** $1 \le i \le m$ **do** $h_i \leftarrow h_{i-1} \cdot h \bmod f$
Vytvoř matici $G$ velikosti $m \times m$ s řádky $g_{1,0}, \ldots, g_{1,n/m-1}, g_{2,0}, \ldots, g_{k,n/m-1}$
Vytvoř matici $H$ velikosti $m \times n$ s řádky $h_0, \ldots, h_{m-1}$
$B \leftarrow G \cdot H$, řádky této matice označ jako $b_{1,0}, \ldots, b_{1,n/m-1}, b_{2,0}, \ldots, b_{k,n/m-1}$
**return** $(\sum_{j=0}^{n/m-1} b_{1,j} \cdot (h^m)^j \bmod f, \ldots, \sum_{j=0}^{n/m-1} b_{k,j} \cdot (h^m)^j \bmod f)$

**Algoritmus 3.3: Skládání mnoha polynomů pomocí maticového násobení**

**Tvrzení 3.3:** Jsou-li všechny polynomy stupně nejvýš $n$ a je-li $k \le n$, funguje algoritmus v čase $\mathcal{O}(n^{(\omega+1)/2}k^{(\omega-1)/2})$. Po dosazení za $\omega$ dostaneme $\mathcal{O}(n^{1.688}k^{0.688})$.

**Důkaz:** Co se týče správnosti, algoritmus funguje stejně jako předchozí, jenom funguje pro víc polynomů najednou, takže správnost plyne ze správnosti jednoduché verze skládání. Časová složitost počítání $h_i$ je $\mathcal{O}(\sqrt{nk}M(n))$, násobení matic stojí $\mathcal{O}(\frac{n}{m}m^\omega) = \mathcal{O}(n^{(\omega+1)/2}k^{(\omega-1)/2})$ a závěrečné kombinování výsledků trvá opět $\mathcal{O}(\sqrt{nk}M(n))$. Stejně jako předtím je násobení matic časově nejnáročnější.

$\boxtimes$

## Bezčtvercová faktorizace

Naším cílem je zrychlit algoritmus bezčtvercové faktorizace. V minulé kapitole jsme popsali algoritmus, jehož složitost byla v nejhorším případě rovna složitosti $n$ NSD. Nyní algoritmus vylepšíme o celý řád, jeho složitost bude asymptoticky odpovídat složitosti NSD. Tuto vylepšenou verzi algoritmu pro tělesa charakteristiky nula jsme převzali z [11].

Opět tedy začněme s tělesem charakteristiky nula a mějme nad ním polynom $f$, jehož bezčtvercová faktorizace je $\prod_{i=1}^{k} f_i^i$, kde všechny $f_i$ jsou bezčtvercové. Hlavní roli ve výpočtu budou hrát polynomy

$$v_i = \prod_{j=i}^{k} f_j \quad \text{a} \quad w_i = \sum_{j=i}^{k} (j-i+1) \cdot \frac{f_j'}{f_j} \cdot v_i.$$

Výhoda těchto polynomů je v tom, že mají malé stupně, každé $f_j$ se v nich nachází pouze v první mocnině. K tomu, abychom jednotlivé $f_j$ dokázali odlišit, nepoužíváme tedy stupeň jejich výskytu, ale koeficient u příslušné části polynomu $w_i$.

Nejprve si ukažme, jak můžeme tyto polynomy induktivně počítat. Na začátku stačí položit $c = \mathrm{NSD}(f, f') = \prod_{i=2}^{k} f_i^{i-1}$, a pak dopočítat hledané polynomy jako

$$v_1 = f/c \quad \text{a} \quad w_1 = f'/c.$$

Nyní předpokládejme, že známe $v_i$, $w_i$ a $f_i$. Chtěli bychom pomocí nich spočítat hodnoty $v_{i+1}$ a $w_{i+1}$. Nejprve si označme

$$d_i = w_i - v_i' = \sum_{j=i}^{k} (j-i+1) \cdot \frac{f_j'}{f_j} \cdot v_i - \sum_{j=i}^{k} \frac{f_j'}{f_j} \cdot v_i = \sum_{j=i+1}^{k} (j-i) \cdot \frac{f_j'}{f_j} \cdot v_i,$$

z čehož už pak jednoduše dostaneme, že

$$v_{i+1} = v_i/f_i \quad \text{a} \quad w_{i+1} = d_i/f_i = (w_i - v_i')/f_i.$$

Zbývá vyřešit poslední a nejtěžší úkol, a to jak z hodnot $v_i$ a $w_i$ určit $f_i$. Uvažujme, jak vypadá $\mathrm{NSD}(v_i, d_i) = \mathrm{NSD}(v_i, w_i - v_i')$. Faktory (ne nutně ireducibilní) polynomu $v_i$ jsou $f_i, f_{i+1}, \ldots, f_k$.

- Polynom $f_i$ dělí $d_i$, protože $d_i = \sum_{j=i+1}^{k} (j-i) \cdot \frac{f_j'}{f_j} \cdot v_i$ a $f_i$ se vyskytuje v každém členu sumy.
- Pro každý polynom $f_j$, $j \neq i$, je $\mathrm{NSD}(f_j, d_i) = 1$. To proto, že $f_j$ je bezčtvercový, tím pádem je dle (2.7) $\mathrm{NSD}(f_j, f_j') = 1$, a tedy také $\mathrm{NSD}(f_j, (j-i) \cdot f_j'/f_j \cdot v_i) = 1$.

Hledané $f_i$ je tedy rovno $\mathrm{NSD}(v_i, d_i)$. Z popsaných rovností vznikne následující algoritmus.

**procedure** BezčtvercováF($f$) : $f$ polynom nad tělesem charakteristiky 0
**if** deg $f = 0$ **then return** $f$
$c \leftarrow \mathrm{NSD}(f, f')$; $\quad v \leftarrow f/c$; $\quad w \leftarrow f'/c$; $\quad i \leftarrow 0$
**while** deg $v > 0$ **do**
  $i \leftarrow i + 1$
  $d \leftarrow w - v'$; $\quad f_i \leftarrow \mathrm{NSD}(v, d)$
  $v \leftarrow v/f_i$; $\quad w \leftarrow d/f_i$
**return** $(f_1, \ldots, f_i)$

**Algoritmus 3.4: Rychlá bezčtvercová faktorizace pro tělesa charakteristiky 0**

**Tvrzení 3.4:** Algoritmus pracuje korektně v čase asymptoticky rovném času spočítání jednoho NSD polynomů stupně nejvýš $n$, tedy v čase $\mathcal{O}(n^{1+\sigma(1)})$.

**Důkaz:** Zbývá dokázat složitost algoritmu. V $i$-tém průchodu cyklem musíme spočítat nad polynomy stupně $\deg v_i$ a $\deg w_i$ dvě dělení, jedno odečítání, derivaci a jeden NSD. Nejsložitější z nich je právě NSD, takže budeme uvažovat jenom spočítání jednoho největšího společného dělitele polynomů $v_i$ a $d_i$. Označme $N(m)$ složitost nalezení NSD polynomů stupně nejvýš $m$ a uvědomme si, že $\deg v_i > \deg d_i$. Složitost celého algoritmu je pak $\sum_{i=1}^{k} N(\deg v_i)$. Víme, že $v_i = \prod_{j=i}^{k} f_j$, z toho dostaneme rovnost $\deg v_i = \sum_{j=i}^{k} \deg f_j$, a tedy složitost celého algoritmu je

$$\sum_{i=1}^{k} N(\deg v_i) \overset{N(n)\geq n}{\leq} N\left(\sum_{i=1}^{k} \deg v_i\right) = N\left(\sum_{i=1}^{k}\sum_{j=i}^{k} \deg f_j\right) =$$

$$= N\left(\sum_{i=1}^{k} i \cdot \deg f_i\right) \overset{f=\prod_i f_i^i}{=} N(\deg f) = N(n).$$

$\boxtimes$

Nyní zkoumejme, jak bude tento algoritmus fungovat na tělesech prvočíselné charakteristiky $p$. Nejprve si uvědomíme, že pro každé $f_j$ je $f_j' \neq 0$. Pokud by totiž byla derivace $f_j$ nulová, $f_j = \sum_i a_i x^{pi} = (\sum_i a_i x^i)^p$ a $f_j$ by nebyl beze čtverců.

Polynom $w_1 = \sum_{j=1}^{k} j \cdot f_j'/f_j \cdot v_1$ je nad $\mathbb{F}_{p^m}$ roven $\sum_{j=1}^{k} j \bmod p \cdot f_j'/f_j \cdot v_1$, takže algoritmus nám mocninu každého bezčtvercového faktoru $f_i$ určí jako $i \bmod p$. Je-li mocnina nějakého $f_i$ je větší nebo rovna $p$, musíme tuto situaci ošetřit. Až algoritmus doběhne a vydá $g_1, \ldots, g_i$, můžeme spočítat zbytek $z = f/(g_1 g_2^2 \cdots g_i^i)$. Pokud byly mocniny všech faktorů menší než $p$, je stupeň polynomu $z$ nula a hledané $f_j$ jsou rovny nalezeným $g_j$.

Když je ale stupeň polynomu $z$ alespoň jedna, musíme dopočítat správné mocniny bezčtvercových faktorů. Hodnota polynomu $z$ je rovna $\prod_i f_i^{p\lfloor i/p \rfloor}$, takže je to (stejně jako v kapitole jedna) $p$-tá mocnina polynomu $z^{1/p} = \prod_i f_i^{\lfloor i/p \rfloor}$. Tento polynom můžeme zpracovat rekurzivně a získáme tak faktory $(z_1, \ldots, z_t)$, přičemž $z_j$ je součin všech bezčtvercových faktorů $f_k$, pro které je $\lfloor k/p \rfloor = j$.

Máme tedy faktory $g_1, \ldots, g_i$, přičemž $g_j$ je součin bezčtvercových faktorů $f_k$ pro $k \bmod p = j$, a faktory $z_1, \ldots, z_t$, kde $z_j$ je součin bezčtvercových faktorů $f_k$ pro $\lfloor k/p \rfloor = j$. Dopočítat hledané $f_j$ je teď už jednoduché: $f_{jp+k}$ pro $1 \leq j$ a $1 \leq k < p$ je $\mathrm{NSD}(g_k, z_j)$, $f_{jp}$ je to, co zbylo ze $z_j$, a $f_j$ pro $1 \leq j < p$ je to, co zbylo z $g_j$. Výrazem „zbylo" myslíme takové faktory, které jsme zatím nepoužili v žádném polynomu $f_j$.

Tím vzniká algoritmus bezčtvercové faktorizace pro konečná tělesa.

> **procedure** BezčtvercováF$(q = p^m, f)$ : $f$ polynom nad tělesem $\mathbb{F}_q$
> **if** $\deg f = 0$ **then return** $f$
> $c \leftarrow \mathrm{NSD}(f, f')$; $\quad v \leftarrow f/c$; $\quad w \leftarrow f'/c$; $\quad i \leftarrow 0$
> **while** $\deg v > 0$ **do**
>     $i \leftarrow i + 1$
>     $d \leftarrow w - v'$; $\quad g_i \leftarrow \mathrm{NSD}(v, d)$
>     $v \leftarrow v/g_i$; $\quad w \leftarrow d/g_i$
> $z \leftarrow f/(g_1 g_2^2 \cdots g_i^i)$
> **if** $\deg z = 0$ **then return** $(g_1, \ldots, g_i)$
> $(z_1, \ldots, z_t) \leftarrow$ BezčtvercováF$(p^m, z^{1/p})$
> **for** $i + 1 \leq j \leq p - 1$ **do** $g_j \leftarrow 1$
> **for** $1 \leq j \leq p - 1$, $1 \leq k \leq t$ **do** $f_{kp+j} \leftarrow \mathrm{NSD}(g_j, z_k)$
> **for** $1 \leq k \leq t$ **do** $f_{kp} \leftarrow z_k/(f_{kp+1} f_{kp+2} \cdots f_{kp+p-1})$
> **for** $1 \leq j \leq p - 1$ **do** $f_j \leftarrow g_j/(f_{p+j} f_{2p+j} \cdots f_{tp+j})$
> **return** $(f_1, \ldots, f_k)$, kde $tp \leq k < (t+1)p$ je největší takové, že $\deg f_k > 0$

**Algoritmus 3.5: Rychlá bezčtvercová faktorizace pro konečná tělesa**

**Tvrzení 3.5:** Algoritmus pracuje správně a v čase $\mathcal{O}(N(n))$, tedy v čase $\mathcal{O}(n^{1+\sigma(1)})$.

**Důkaz:** Na správnosti není co dokazovat, algoritmus je přímou implementací popsaného postupu. Zato dokázat časovou složitost nám dá dost práce.

Nejdříve si uvědomíme, za jak dlouho dokážeme spočítat součin $\prod_{i=1}^{k} h_i$, pokud je $\sum_i \deg h_i = m$. Budeme to dělat po krocích. V každém kroku spočítáme součin dvou „sousedních" polynomů. V prvním kroku spočítáme součiny $h_1 \cdot h_2, h_3 \cdot h_4, \ldots, h_{k-1} \cdot h_k$, v druhém kroku spočítáme součiny $h_1 h_2 \cdot h_3 h_4, \ldots, h_{k-3} h_{k-2} \cdot h_{k-1} h_k$ atd. Těchto kroků bude $\log_2 k$, protože na začátku je polynomů $k$ a v každém kroku jejich počet klesne na polovinu. V jednom kroku se s každým polynomem $h_i$ pracuje právě jednou, takže složitost jednoho kroku můžeme omezit složitostí $\mathcal{O}(M(\sum_{i=1}^{k} \deg h_i)) = \mathcal{O}(M(m))$. Celková složitost je tedy $\mathcal{O}(M(m) \log k)$.

Nyní provedeme analýzu časové složitosti algoritmu bez rekurzivního volání. Algoritmus musí ve svém běhu kromě rekurzivního volání spočítat

- $(g_1, \ldots, g_i)$: z předchozího algoritmu již víme, že to zvládneme v čase $\mathcal{O}(N(n))$.
- $z = f/(g_1 g_2^2 \cdots g_i^i)$: součin polynomů $g_j^j$ dokážeme dle výše uvedeného pozorování spočítat v čase $\mathcal{O}(M(n) \log n) = \mathcal{O}(N(n))$. Pak už jen vydělíme v čase $\mathcal{O}(M(n))$.
- $\mathrm{NSD}(g_j, z_k)$ pro $1 \le j \le p-1$ a $1 \le k \le t$: abychom dosáhli dobré složitosti, budeme muset udělat předvýpočet – spočítat si $g_j \bmod z_k$ pro každé $j$ a $k$. Poté budeme místo $\mathrm{NSD}(g_j, z_k)$ počítat $\mathrm{NSD}(g_j \bmod z_k, z_k)$, což je jistě ekvivalentní. Nejdříve určíme složitost předvýpočtu. Víme, že pro pevné $j$ dokážeme spočítat všechny $g_j \bmod z_k$ v čase $\mathcal{O}(M(l) \log l)$, kde $l = \max(\deg g_j, \sum_k \deg z_k)$. Toto můžeme odhadnout jako $l \le \max(\deg g_j, n/p) \le \deg g_j + n/p$. Pokud sečteme složitost kroků pro všechna $j$, dostaneme
$$\sum_j \mathcal{O}(M(\deg g_j + n/p) \log(\deg g_j + n/p)) \le \sum_j \mathcal{O}(M(\deg g_j + n/p) \log n) \le$$
$$\le \mathcal{O}(M(\sum_j \deg g_j + p \cdot n/p) \log n) \le \mathcal{O}(M(n+n) \log n) = O(M(n) \log n).$$
Nyní odhadneme složitost výpočtu všech $\mathrm{NSD}(g_j \bmod z_k, z_k)$. Spočítání jednoho takového největšího společného dělitele zabere čas $\mathcal{O}(N(\deg z_k))$. Pokud vezmeme pevné $g_j$, na spočtení všech $\mathrm{NSD}(g_j \bmod z_k, z_k)$ potřebujeme $\sum_k \mathcal{O}(N(\deg z_k)) \le \mathcal{O}(N(\sum_k \deg z_k) \le \mathcal{O}(N(n/p))$ operací. Možných $g_j$ je nejvýš $p$, čímž se dostaneme na $\mathcal{O}(p \cdot N(n/p)) \le \mathcal{O}(N(n))$.
- $z_k/(f_{kp+1} f_{kp+2} \cdots f_{kp+p-1})$ pro $1 \le k \le t$: buď $k$ pevné. Součin polynomů $f_j$ dělí $z_k$. Proto je stupeň tohoto součinu nejvýš roven stupni $z_k$, takže spočítat ho dokážeme v čase $\mathcal{O}(M(\deg z_k) \log n/p)$. Pak jenom $z_k$ vydělíme v čase $\mathcal{O}(M(\deg z_k))$. Sečteme-li tyto složitosti pro všechna $z_k$, dostaneme $\mathcal{O}(M(\sum_k \deg z_k) \log n/p)$, což je určitě omezené složitostí $\mathcal{O}(M(n/p) \log n/p) = \mathcal{O}(N(n/p))$.
- $g_j/(f_{p+j} f_{2p+j} \cdots f_{tp+j})$ pro $1 \le j \le p-1$: stejnou úvahou jako v minulém případě zjistíme, že pro pevné $j$ potřebujeme $\mathcal{O}(M(\deg g_j) \log n)$ operací. Sečtením přes všechna $g_j$ dostaneme $\mathcal{O}(M(\sum_j \deg g_j) \log n)$, což je určitě omezené složitostí $\mathcal{O}(M(n) \log n) = \mathcal{O}(N(n))$.

Tím dostáváme složitost $\mathcal{O}(N(n))$ bez rekurzivního volání.

Vyřešit rekurzivní volání je jednoduché. Pokud označíme $T(n)$ složitost algoritmu spuštěného na polynom stupně $n$, dostaneme

$$T(n) \le N(n) + T\left(\frac{n}{p}\right) \le \sum_{i=0}^{\infty} N\left(\frac{n}{p^i}\right) \overset{N(n) \ge n}{\le} \sum_{i=0}^{\infty} \frac{1}{p^i} \cdot N(n) = \frac{p}{p-1} \cdot N(n) \overset{p \ge 2}{\le} 2N(n).$$

Složitost algoritmu je tím pádem opravdu rovna slíbené složitosti $\mathcal{O}(N(n)) = \mathcal{O}(n^{1+\sigma(1)})$.

⊠

## Různostupňová faktorizace

Zopakujme, že cílem různostupňové faktorizace je rozložit daný polynom $f$ na součin $f_1 \cdots f_k$ tak, aby každý polynom $f_i$ byl součinem všech ireducibilních faktorů $f$ stupně $i$. Již popsaný algoritmus pracuje takto:

> **procedure** RůznostupňováF($q$,$f$) : $f$ bezčtvercový polynom nad $\mathbb{F}_q$
> $i \leftarrow 0$;
> **while** $\deg f > 0$ **do** $i \leftarrow i+1$; $f_i \leftarrow \mathrm{NSD}(x^{q^i} - x, f)$; $f \leftarrow f/f_i$
> **return** $(f_1, f_2, \ldots, f_i)$

Začneme tvrzením, které nás zbaví nutnosti počítat všechny $x^{q^i} - x$.

**Tvrzení 3.6:** Polynom $x^{q^i} - x^{q^j}$ nad $\mathbb{F}_q$ je dělitelný všemi monickými ireducibilními polynomy, jejichž stupeň dělí $i - j$.

**Důkaz:** Toto tvrzení je jednoduchý důsledek (2.4). BÚNO předpokládejme, že $i \geq j$. Pak $x^{q^i} - x^{q^j} = (x^{q^{i-j}} - x)^{q^j}$ a důkaz plyne z použití (2.4) pro polynom $x^{q^{i-j}} - x$.

$\boxtimes$

Nyní se vrhneme na algoritmus, který byl poprvé popsán v [8] a dále zkoumán v [10]. Ten využívá k počítání vysokých mocnin polynomu $x^{q^i} - x$ jednak násobení a jednak skládání polynomů. To, kolik práce vykonává jaká operace, určuje parametr algoritmu $\beta$, přičemž $0 \leq \beta \leq 1$. Nakonec, po určení složitosti algoritmu, zvolíme takovou hodnotu parametru, aby byla časová složitost co nejmenší.

> **procedure** RůznostupňováF($q$,$f$,$\beta$) : $f$ bezčtvercový nad $\mathbb{F}_q$, $0 \leq \beta \leq 1$
> $n \leftarrow \deg f$
> (R1) $l \leftarrow \lceil n^\beta \rceil$; **for** $0 \leq i \leq l$ **do** $h_i \leftarrow x^{q^i} \bmod f$
> (R2) $m \leftarrow \lceil n/2l \rceil$; **for** $1 \leq j \leq m$ **do** $H_j \leftarrow x^{q^{lj}} \bmod f$
> (R3) **for** $1 \leq j \leq m$ **do** $I_j \leftarrow \prod_{i=0}^{l-1}(H_j - h_i) \bmod f$
> *Každý $I_j$ je dle (3.6) dělitelný všemi ireducibilními polynomy, jejichž stupeň dělí $k$, přičemž $(j-1)l + 1 \leq k \leq jl$.*
> (R4) **for** $1 \leq j \leq m$ **do** $F_j \leftarrow \mathrm{NSD}(f, I_j)$; $f \leftarrow f/F_j$
> *Každý $F_j$ je teď součin $f_{(j-1)l+1}, \ldots, f_{jl}$, kde $f_i$ jsou hledané faktory.*
> (R5) **for** $1 \leq j \leq m$ **do**
> **for** $l-1 \geq i \geq 0$ **do** $f_{lj-i} \leftarrow \mathrm{NSD}(F_j, H_j - h_i)$; $F_j \leftarrow F_j/f_{lj-i}$
> **if** $\deg f > 0$ **then** $f_{\deg f} \leftarrow f$
> **return** $(f_1, \ldots, f_k)$, kde $1 \leq k \leq n$ je největší takové, že $\deg f_k > 0$

**Algoritmus 3.7: Rychlá různostupňová faktorizace**

**Tvrzení 3.7:** Algoritmus funguje správně, v čase $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+\sigma(1)} \log q)$.

**Důkaz:** Správnost algoritmu plyne z komentářů a tvrzení (3.6). Časovou složitost algoritmu budeme určovat postupně po krocích:

(R1) Použijeme $l$ po sobě jdoucích binárních umocňování, čímž se dostaneme na časovou složitost $\mathcal{O}(n^{1+\beta+\sigma(1)} \log q)$.

(R2) Zde naopak použijeme skládání polynomů, konkrétně algoritmus (3.3). $H_1$ už na začátku známe, protože $H_1 = h_l$. Nyní předpokládejme, že máme spočítáno $H_1, \ldots, H_{2^i}$. Použitím (3.3) spočítáme $H_1(H_{2^i}) \bmod f, \ldots, H_{2^i}(H_{2^i}) \bmod f$. Protože $H_j(H_{2^i}) \bmod f = H_{2^i+j}$, po jednom takovém kroku známe $H_1, \ldots, H_{2^{(i+1)}}$. Protože se v jednom kroce zdvojnásobí počet spočtených $H_i$, musíme tento krok provést $\log\lceil n/2l \rceil$-krát. Všechny $H_i$ tedy spočteme v čase $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2})$.

(R3) Tento krok provedeme ve dvou fázích. Označme $V$ jako okruh $\mathbb{F}_q[x]/f$. V první fázi spočítáme koeficienty polynomu $H \in V[y]$ stupně $l$:

$$H = \prod_{i=0}^{l-1} (y - h_i).$$

To zvládneme pomocí $l$ násobení, tedy v čase $\mathcal{O}(n^{1+\beta+\sigma(1)})$.
Druhá fáze pak vyčíslí polynom $H$ v bodech $H_1, \ldots, H_m$. Tuto fázi zvládneme pomocí $\mathcal{O}(M(m)\log m + M(l))$ operací nad $V$, přičemž jedna taková operace trvá nejvýš $\mathcal{O}(M(n))$. Celkem dostaneme $\mathcal{O}(n^{2-\beta+\sigma(1)} + n^{1+\beta+\sigma(1)})$.
Za předpokladu $2 < \omega < 3$ je $\mathcal{O}(n^{2-\beta+\sigma(1)})$ omezena $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2})$, takže celý krok (R3) zvládneme v čase $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+\sigma(1)})$.

(R4) Přímočará implementace použije $m$ NSD a $m$ dělení, čímž jsme na $\mathcal{O}(n^{2-\beta+\sigma(1)})$.

(R5) Pro dosažení pěkné časové složitosti budeme muset tentokrát provést dvě předpočítání. Problém je v tom, že ačkoliv je součet stupňů všech polynomů $F_j$ jenom $n$, pro polynomy $H_j$ ani $h_i$ to neplatí. Pokud by byly stupně všech těchto polynomů $n-1$, což je klidně možné, krok (R5) by trval $\mathcal{O}(n^{2+\sigma(1)})$, což nechceme.
Všichni ale víme, že $\mathrm{NSD}(a, b) = \mathrm{NSD}(a, b \bmod a)$. Pokud si předpočítáme
- $\forall j$ výraz $H_j \bmod F_j$,
- $\forall i, j$ výraz $h_i \bmod F_j$,

můžeme pro výpočet NSD použít místo skutečných $H_j$ a $h_i$ spočítané zbytky po dělení.
Co se týče časové složitosti, předvýpočet všech $H_j \bmod F_j$ trvá $\mathcal{O}(m \cdot M(n)) = \mathcal{O}(n^{2-\beta+\sigma(1)})$. Pokud si zafixujeme jedno $h_i$, získat zbytky po dělení všemi $F_j$ trvá jenom $\mathcal{O}(M(n)\log n) = \mathcal{O}(n^{1+\sigma(1)})$ operací, takže na všechna $h_i \bmod F_j$ potřebujeme čas $\mathcal{O}(n^{1+\beta+\sigma(1)})$. Zbývá složitost všech NSD. Pokud bychom počítali právě jeden NSD pro každý polynom $F_j$, časová složitost by byla $\mathcal{O}(n^{1+\sigma(1)})$, protože součet stupňů všech $F_j$ je roven $n$. Algoritmus ale potřebuje pro každé $F_j$ spočítat $l$ NSD, z čehož získáme složitost $\mathcal{O}(n^{1+\beta+\sigma(1)})$.

Celkem se dostáváme na slibovanou složitost $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+\sigma(1)} \log q)$. ⊠

Nyní stačí zvolit nejvhodnější $\beta$, což je takové, které splňuje rovnost

$$(\omega + 1)/2 + (1 - \beta)(\omega - 1)/2 = 1 + \beta.$$

Po jednoduchém výpočtu dostaneme $\beta = 2\frac{\omega-1}{\omega+1}$, z čehož po dosazení vyjde $\beta = 0.815$, a složitost našeho algoritmu je tedy $\mathcal{O}(n^{1.815} \log q)$. Z této složitosti opravdu zmizelo $\sigma(1)$, protože jestli je $\omega < 2.376$, tak také platí, že $\omega + \sigma(1) < 2.376$.

## Stejnostupňová faktorizace

Nyní popíšeme rychlou variantu algoritmu stejnostupňové faktorizace, která pochází z [6]. Pro zopakování nejdřív uvedeme základní verzi algoritmu z minulé kapitoly.

**procedure** StejnostupňováF($q = p^m$,$f$,$d$) : $f$ bezčtvercový polynom nad $\mathbb{F}_q$ s faktory stupně $d$
**if** deg $f \leq d$ **then return** $\{f\}$
**do**
    $a \leftarrow$ náhodný polynom nad $\mathbb{F}_q$ stupně menšího než deg $f$
♡   **if** $p = 2$ **then** $b \leftarrow \sum_{i=0}^{d-1} a^{2^i}$ mod $f$ **else** $b \leftarrow a^{(q^d-1)/2} - 1$ mod $f$
♡   $c \leftarrow \text{NSD}(b, f)$
**while** $c = 1 \vee c = f$
**return** StejnostupňováF($p^m, c, d$) $\cup$ StejnostupňováF($p^m, f/c, d$)

Výpočet si trochu upravíme. K tomu se nám bude hodit zobecnění tvrzení (2.17), které uvedeme jako

**Tvrzení 3.8:** Nad $\mathbb{F}_{q=p^k}$ s prvočíselnou charakteristikou $p$ definujme $T_k = \sum_{i=0}^{k-1} x^{p^i}$. Pak
(1) $x^q - x = \prod_{s \in \mathbb{F}_p} (T_k - s)$,
(2) pro každé $a \in \mathbb{F}_q$ je $T_k(a) \in \mathbb{F}_p$,
(3) pro každé $b \in \mathbb{F}_p$ existuje právě $p^{k-1}$ hodnot $a \in \mathbb{F}_q$, že $T_k(a) = b$.

**Důkaz:** Dle (2.3) nad $\mathbb{F}_p$ platí, že $x^p - x = \prod_{s \in \mathbb{F}_p} (x - s)$. Z toho získáme, že

$$\prod_{s \in \mathbb{F}_p} (T_k - s) = T_k^p - T_k = \left[ \sum_{i=0}^{k-1} x^{p^i} \right]^p - \sum_{i=0}^{k-1} x^{p^i} = \sum_{i=0}^{k-1} x^{p^i p} - \sum_{i=0}^{k-1} x^{p^i} = \sum_{i=1}^{k} x^{p^i} - \sum_{i=0}^{k-1} x^{p^i} = x^{p^k} - x.$$

Dále pokud máme $a \in \mathbb{F}_q$, je to kořen polynomu $x^q - x$, takže je i kořenem jednoho z polynomů $T_k - s$ pro $s \in \mathbb{F}_p$, čili $T_k(a) \in \mathbb{F}_p$. K důkazu posledního bodu si stačí uvědomit, že každé z $q$ různých $a \in \mathbb{F}_q$ je kořenem jednoho z $p$ polynomů $T_k - s$ stupně $p^{k-1}$. To je možné, jenom má-li každý polynom $T_k - s$ právě $p^{k-1}$ kořenů.

⊠

Na zbytku této stránky budeme používat značení, které jsme zavedli u popisu algoritmu stejnostupňové faktorizace v předchozí kapitole.

Mějme náhodný polynom $a$ nad $\mathbb{F}_{p^m}$ stupně menšího než $f$. Hledáme zobrazení $Z$, aby šance, že nějaké $\varphi_i(Z(a))$ je nula a nějaké $\varphi_j(Z(a))$ je nenulové, byla co největší. Když označíme $b = \sum_{i=0}^{md-1} a^{p^i}$, předchozí tvrzení nám říká, že každé $\varphi_i(b) \in \mathbb{F}_p$, kde všechny prvky $\mathbb{F}_p$ jsou stejně pravděpodobné. Je-li charakteristika tělesa dva, jsme spokojeni.

Pokud je ovšem charakteristika lichá, $\varphi_i(b)$ stále nabývá mnoha hodnot. Vzpomeneme-li si ale na tvrzení (2.16) o odmocninách, zjistíme, že každé $\varphi_i(b^{(p-1)/2})$ nabývá jenom hodnot $-1, 0, 1$ s pravděpodobnostmi po řadě $\frac{1}{2} - \frac{1}{2p}, \frac{1}{p}, \frac{1}{2} - \frac{1}{2p}$. Pokud tedy jako dělící polynom použijeme $b^{(p-1)/2} - 1$ a v případě jeho neúspěchu zkusíme ještě $b^{(p-1)/2} + 1$, žádný netriviální faktor nenalezneme právě tehdy, když jsou všechny $\varphi_i(b^{(p-1)/2})$ stejné. K tomu dojde s pravděpodobností $2(\frac{1}{2} - \frac{1}{2p})^k + (\frac{1}{p})^k$, což je pro $k \geq 2$ a $p \geq 3$ menší než $1/2$. Tím jsme vyřešili problém i pro lichou charakteristiku.

Označené řádky algoritmu můžeme tedy přepsat na:

$b \leftarrow \sum_{i=0}^{md-1} a^{p^i}$ mod $f$
**if** $p \neq 2$ **then** $b \leftarrow b^{\frac{p-1}{2}} - 1$ mod $f$
$c \leftarrow \text{NSD}(b, f)$
**if** $p \neq 2 \wedge (c = 1 \vee c = f)$ **then** $c \leftarrow \text{NSD}(b + 2, f)$

Naším cílem je teď co nejrychleji spočítat sumu $\sum_{i=0}^{k-1} a^{p^i}$. Předpokládejme na chvíli, že $k = 2^r$ je mocnina dvojky. Pro skládání polynomů platí následující rovnosti

$$a\left(x^{p^i}\right) \bmod f = a^{p^i} \bmod f$$
$$x^{p^i}\left(x^{p^i}\right) \bmod f = x^{p^{2i}} \bmod f$$
$$\sum_{j=0}^{i-1} a^{p^j} \bmod f + \left[\sum_{j=0}^{i-1} a^{p^j}\right]\left(x^{p^i}\right) \bmod f = \sum_{j=0}^{2i-1} a^{p^j} \bmod f,$$

ze kterých můžeme vytvořit následující algoritmus:

**procedure** SumaMocnin($f$,$x^p$,$a$,$k = 2^r$) : spočte $\sum_{i=0}^{k-1} a^{p^i} \bmod f$
**if** $d = 1$ **then return** a
$w \leftarrow x^p$; $\quad a \leftarrow a + a(w) \bmod f$
**for** $2 \leq i \leq r$ **do** $w \leftarrow w(w) \bmod f$; $\quad a \leftarrow a + a(w) \bmod f$
**return** $a$

Správnost algoritmu plyne z popsaných identit. Časová náročnost algoritmu se nám také velmi líbí, protože potřebujeme jenom $2\log k$ modulárních skládání. Jediné, co se nám nelíbí, je nutnost toho, aby $k$ byla mocnina dvojky. Tento problém jde samozřejme vyřešit.

Nyní je tedy $k$ libovolné a nechť $k_i$ je $i$-tý bit jeho dvojkového zápisu, takže $k$ můžeme zapsat jako $k = \sum_{i=0}^{\lfloor \log_2 k \rfloor} 2^i k_i$. Hledanou sumu budeme počítat postupně a v každém kroku budeme znát jednak její začátek $A = \sum_{i=0}^{m-1} a^{p^i}$ a jednak $W = x^{p^m}$ pro nějaké $m$. Algoritmus bude pracovat v $\lfloor \log_2 k \rfloor + 1$ krocích a na začátku $i$-tého kroku si již popsaným způsobem spočte $X_i = x^{p^{2^i}}$ a $S_i = \sum_{j=0}^{2^i-1} a^{p^j}$. Navíc, pokud je to vhodné, prodlouží vytvářenou sumu o několik dalších členů. Přesněji, je-li v $i$-tém kroku $k_i = 1$, prodlouží sumu o $2^i$ dalších členů tak, že udělá $A = A + S_i(W) \bmod f$. Navíc je v tomto případě potřeba udržet $W$ synchronizované, takže se ještě provede $W = W(X_i) \bmod f$.

Sepišme tento postup do algoritmu.

**procedure** SumaMocnin($f$,$x^p$,$a$,$k$) : spočte $\sum_{i=0}^{k-1} a^{p^i} \bmod f$
$A \leftarrow 0$; $\quad W \leftarrow x$
$X \leftarrow x^p$; $\quad S \leftarrow a \bmod f$
**for** $0 \leq i \leq \lfloor \log_2 k \rfloor$ **do**
    **if** $k_i = 1$ **then** $A \leftarrow A + S(W) \bmod f$; $W \leftarrow W(X) \bmod f$
    $S \leftarrow S + S(X) \bmod f$; $\quad X \leftarrow X(X) \bmod f$
**return** $A$

**Algoritmus 3.9: Suma mocnin pro stejnostupňovou faktorizaci**

**Tvrzení 3.9:** Algoritmus funguje správně a v čase $\mathcal{O}(n^{(\omega+1)/2} \log k)$.

**Důkaz:** Označme $K_i$ číslo, které vznikne z $k$ použitím posledních $i+1$ bitů. Tedy $K_i = \sum_{j=0}^{i} 2^j k_j = k \bmod 2^{i+1}$. Pomocí popsaných identit není těžké indukcí dokázat, že na konci $i$-tého kroku algoritmu platí

$$X_{i-1} = x^{p^{2^i}}, \qquad S_{i-1} = \sum_{j=0}^{2^i-1} a^{p^j},$$
$$W_i = x^{p^{K_i}}, \qquad A_i = \sum_{j=0}^{K_i-1} a^{p^j}.$$

Na konci algoritmu je tedy v $A$ požadovaná suma.

Co se týče časové složitosti, kromě $\mathcal{O}(\log k)$ skládání polynomů potřebujeme jenom sčítání. Složitost skládání tedy určuje složitost algoritmu, z čehož vzniká $\mathcal{O}(n^{(\omega+1)/2}\log k)$.

$\boxtimes$

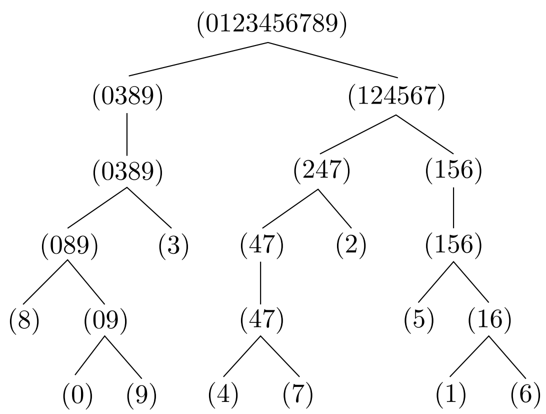
**procedure** StejnostupňováF$(q = p^m, f, d)$ : $f$ bezčtvercový polynom nad $\mathbb{F}_q$ s faktory stupně $d$
**if** $\deg f \le d$ **then return** $\{f\}$
**do**
  $a \leftarrow$ náhodný polynom nad $\mathbb{F}_q$ stupně menšího než $\deg f$
  $b \leftarrow \sum_{i=0}^{md-1} a^{p^i} \bmod f$ spočti pomocí algoritmu (3.9)
  **if** $p \neq 2$ **then** $b \leftarrow b^{\frac{p-1}{2}} - 1 \bmod f$
  $c \leftarrow \mathrm{NSD}(b, f)$
  **if** $p \neq 2 \wedge (c = 1 \vee c = f)$ **then** $c \leftarrow \mathrm{NSD}(b + 2, f)$
**while** $c = 1 \vee c = f$
**return** StejnostupňováF$(p^m, c, d) \cup$ StejnostupňováF$(p^m, f/c, d)$

**Algoritmus 3.10: Rychlá stejnostupňová faktorizace**

**Tvrzení 3.10:** Algoritmus funguje správně a v čase $\mathcal{O}(n^{(\omega+1)/2+\sigma(1)}\log q)$.

**Důkaz:** Správnost algoritmu plyne ze správnosti algoritmu (2.18) a z předchozích dvou tvrzení. Co se týče složitosti, nejprve určíme složitost algoritmu bez rekurzivního volání. Už víme, že očekávaný počet polynomů $a$ pro nalezení užitečného štěpícího polynomu je 2. Složitost algoritmu bez rekurze se tedy složitost otestování jednoho polynomu $a$, což v nejhorším dokážeme pomocí jednoho volání algoritmu (3.9), dvou umocnění na $p$ a dvou NSD. To zvládneme v čase $\mathcal{O}(n^{(\omega+1)/2+\sigma(1)}\log\log q + n^{1+\sigma(1)}\log p)$, což je určitě omezeno čitelnějším časem $\mathcal{O}(n^{(\omega+1)/2+\sigma(1)}\log q)$, který si označíme jako $C(n)$.

Ještě musíme vypočíst složitost rekurze. K tomu si zavedeme *strom výpočtu*. Strom výpočtu zachycuje postup rekurze. V kořeni stromu je polynom $f$, který chceme faktorizovat, a každý další vrchol odpovídá jednomu pokusu o rozštěpení nějakého faktoru $f$. Každý vrchol, který se pokouší faktorizovat ireducibilní faktor $f$, je tedy list, a vnitřní vrcholy mají jednoho nebo dva syny podle toho, zda se jim povedlo nebo nepovedlo faktor na první pokus rozštěpit. Následuje příklad jednoho fiktivního stromu výpočtu, který faktorizuje součin deseti ireducibilních polynomů označených jako 0 až 9.




Zaměřme se na to, kolik času strávíme při zpracování všech polynomů na jedné úrovni stromu. Víme, že na každé úrovni je každý faktor nejvýš jednou, takže součet stupňů polynomů jedné úrovně je $n$. Zpracování jedné úrovně stojí tedy $\sum_i C(n_i)$, přičemž $\sum_i n_i = n$. Protože je ale $C(n) \ge n$, platí, že $\sum_i C(n_i) \le C(\sum_i n_i) = C(n)$. Časová složitost celého algoritmu včetně rekurze je tedy $O(\textit{výška stromu výpočtu} \cdot C(n))$.

Odhadneme výšku stromu výpočtu. Nechť je rozklad polynomu $f = f_1 \cdots f_k$. Zvolme si pevná $1 \le i < j \le k$. Předpokládejme, že $f_i$ a $f_j$ zatím nebyly algoritmem rozděleny, čili jsou v jednom vrcholu. Jaká je šance, že teď budou odděleny? Tato šance odpovídá tomu, že $\varphi_i(a) \neq \varphi_j(a)$, což je alespoň $1/2$. Tím pádem je pravděpodobnost, že polynomy $f_i$ a $f_j$ *nejsou* po $h$-té úrovni odděleny, nanejvýš $2^{-h}$.

Označme $p_h$ pravděpodobnost, že po $h$-té úrovni nejsou ještě všechny dvojice ireducibilních faktorů $f$ rozděleny. Protože je těchto dvojic méně než $k^2$, je $p_h \le \min(k^2 \cdot 2^{-h}, 1)$.

Pokud je $P_h$ pravděpodobnost toho, že strom výpočtu má výšku $h$, střední hodnota výšky stromu je $\sum_h h \cdot P_h$. Pomocí $p_h$ to můžeme zapsat jako $\sum_{h=1}^{\infty} h \cdot (p_h - p_{h+1})$. Označme $s = \lceil 2 \log_2 k \rceil$, což je první hodnota, kdy je $k^2 \cdot 2^{-s} \le 1$, a upravujme:

$$\sum_{h=1}^{\infty} h \cdot (p_h - p_{h+1}) = \sum_{h=1}^{\infty} p_h \le \sum_{i=1}^{s-1} 1 + \sum_{i=s}^{\infty} k^2 \cdot 2^{-i} = s + 2 \cdot k^2 \cdot 2^{-s} \le s + 2 = \mathcal{O}(\log k).$$

Očekávaná výška stromu výpočtu je proto $\mathcal{O}(\log n)$, takže složitost celého algoritmu včetně rekurze je $\mathcal{O}(\log n \cdot C(n)) = \mathcal{O}(n^{\sigma(1)} \cdot C(n)) = \mathcal{O}(n^{(\omega+1)/2+\sigma(1)} \log q)$.

$\boxtimes$

Po dosazení za $\omega$ dostáváme složitost $\mathcal{O}(n^{1.688} \log q)$, což je méně, než složitost různostupňové faktorizace. Na druhou stranu to není složitost v nejhorším případě, ale pouze průměrná.

## Kompletní algoritmus faktorizace

Nyní můžeme poskládat všechny výsledky této kapitoly a sestavit celý algoritmus.

**procedure** Faktorizace($q$,$f$) : spočte rozklad libovolného $f \in \mathbb{F}_q[x]$
$(f_1, \ldots, f_k) \leftarrow$ BezčtvercováF$(q, f)$
*faktory* $\leftarrow \{\}$
**for** $1 \leq i \leq k$ **do**
  $(g_1, \ldots, g_l) \leftarrow$ RůznostupňováF$(q, f_i)$
  **for** $1 \leq j \leq l$ **do**
    $(h_1, \ldots, h_m) \leftarrow$ StejnostupňováF$(q, g_j, j)$
    *faktory* $\leftarrow$ *faktory* $\cup \{(h_1)^i, \ldots, (h_m)^i\}$
**return** *faktory*

**Algoritmus 3.11: Rychlá faktorizace polynomu nad konečným tělesem**

**Tvrzení 3.11:** Algoritmus funguje správně, v čase $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2}+n^{1+\beta+\sigma(1)}\log q)$. Použitím odhadu pro $\omega$ se dostáváme na $\mathcal{O}(n^{1.815}\log q)$.

**Důkaz:** Správnost algoritmu plyne ze správnosti použitých algoritmů částečných faktorizací.

V průběhu celého běhu algoritmu se jednou volá bezčtvercová faktorizace na polynom stupně $n$. Dále se volá několik různostupňových faktorizací na polynomy, jejichž součet stupňů je nejvýše $n$. Stejnostupňovou faktorizaci voláme také potenciálně víckrát, ale opět je součet stupňů polynomů, které v celém běhu algoritmu stejnostupňově faktorizujeme, nanejvýš $n$. Protože je čas různostupňové i stejnostupňové faktorizace větší než $n$, můžeme říci, že je čas celého algoritmus omezen voláním jedné bezčtvercové faktorizace polynomu stupně $n$, jedné různostupňové faktorizace polynomu stupně $n$ a jedné stejnostupňové faktorizace polynomu stupně $n$. Tyto tři faktorizace vyžadují po řadě čas $\mathcal{O}(n^{1+\sigma(1)})$, $\mathcal{O}(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2}+n^{1+\beta+\sigma(1)}\log q)$ a $\mathcal{O}(n^{(\omega+1)/2+\sigma(1)}\log q)$. Po použití odhadu $\omega$ je můžeme zapsat jako $\mathcal{O}(n^{1+\sigma(1)})$, $\mathcal{O}(n^{1.815}\log q)$ a $\mathcal{O}(n^{1.688}\log q)$. Složitost celého algoritmu je asymptoticky rovna největší z těchto složitostí, což je složitost různostupňové faktorizace.

⊠

# 4. Vlastní implementace

Implementaci algoritmu faktorizace jsem se rozhodl vytvořit v jazyce `C++`. Jednak je to jazyk známý a široce používaný a jednak umožňuje přetěžovat operátory, čímž se algoritmy faktorizace stanou na pohled srozumitelné. Navíc jsem pro implementaci polynomů s operacemi nad nimi a implementaci faktorizace použil šablony. To má tu výhodu, že polynomy efektivně pracují nad libovolnou implementací konečných těles a stejně tak faktorizace pracuje s každou dodanou implementací polynomů, takže je možné využívat již existujících knihoven.

K implementaci jsem použil v minulé kapitole popsaný subkvadratický algoritmus faktorizace, ovšem s drobnými změnami:

- celý subkvadratický algoritmus závisí na rychlém násobení matic. Rychlé násobení matic je rychlé bohužel pouze asymptoticky a pro skutečně používané matice je neefektivní. Proto jsem k násobení matic použil základní násobení řídkých matic podle definice, které pracuje při nejhorším v kubickém čase, ale na použitých maticích (které jsou velké řádově $\sqrt{n}$) je rychlejší než asymptoticky lepší násobení pracující v čase $\mathcal{O}(n^{2.375})$.
  Také by bylo možné použít Strassenův algoritmus pro násobení matic, který pracuje v čase $\mathcal{O}(n^{\log_2 7} \approx n^{2.8})$ a pro matice o řádově tisíci prvcích a více je v praxi opravdu rychlejší než přímočarý algoritmus.
- rychlou variantu různostupňové faktorizace upravíme tak, že zvolíme $l = m = \lceil\sqrt{n/2}\rceil$, $h_i$ i $H_j$ spočteme pouze pomocí skládání polynomů a ostatní kroky naprogramujeme přímočaře podle popisu algoritmu (3.7). Tím se spolu s přímočarým násobením matic dostaneme sice na asymptotickou složitost $\mathcal{O}(n^{2.5} + n^2 \log q)$, případně na $\mathcal{O}(n^{2.4} + n^2 \log q)$ pro Strassenovo násobení matic, což je asymptoticky horší, ale pro skutečné velikosti vstupů rychlejší algoritmus. Podrobnosti o takto upraveném algoritmu různostupňové faktorizace můžeme nalézt například v [10].

Problém faktorizace je ten, že ke svému efektivnímu běhu potřebuje pokud možno rychle provádět operace nad konečnými tělesy a polynomy. Těchto operací je bohužel nemalý počet a efektivní implementace mnoha z nich je pracná. Proto jsem zvažoval následující možnosti:

1) použít pro operace nad konečným tělesy a nad polynomy již existující knihovny. Tento postup by měl výhodu v tom, že by v nich byly tyto operace implementovány robustně a efektivně a já bych se mohl soustředit pouze na algoritmus faktorizace. Na druhou stranu není jisté, zda by takové knihovny neobsahovaly pouze obecné operace a zda by některé požadované speciální operace dokázaly provést opravdu rychle.
2) naprogramovat si operace nad konečnými tělesy a polynomy sám na míru. Zde je nevýhodou to, kolik času bych strávil nad jejich programováním.

Nakonec jsem se rozhodl pro to, si tyto operace naprogramovat sám, chtěl jsem se naučit, jak se to dělá efektivně. Jak operace nad polynomy tak samotnou faktorizaci jsem ovšem navrhl velmi modulárně, takže není žádný problém použít libovolnou vyhovující knihovnu pro práci s konečnými tělesy či polynomy.

Celá implementace se dá rozdělit do čtyř hlavních oblastí:

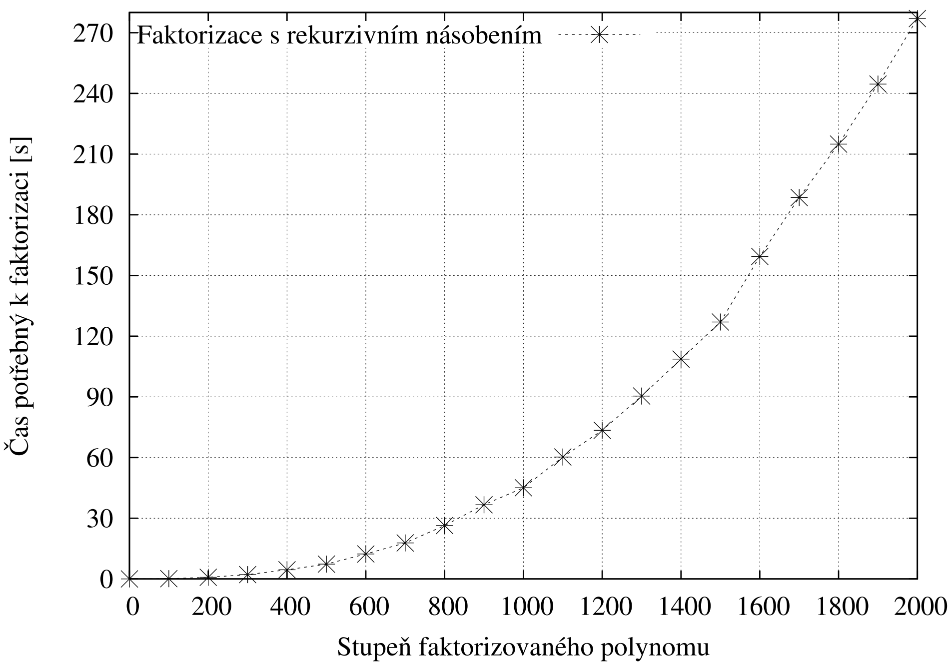
- operace nad konečnými tělesy: rozhodl jsem se vytvořit vlastní implementaci pouze pro tělesa prvočíselné velikosti a jenom taková, která se vejdou přímo do registrů dnešních počítačů. To v nejběžnějším případě znamená, že použité těleso musí mít nanejvýš $2^{31} - 1$ prvků. Za malou velikost tělesa ovšem získám velmi rychlé operace sčítání a násobení. Navíc v případě potřeby můžeme použít jinou implementaci povolující větší tělesa.

- operace nad polynomy: pro zachování jednoduchosti operací jsem nepoužil řídké polynomy, takže když má polynom stupeň $n$, vždy se skládá z $n+1$ prvků tělesa. Za zmínku stojí následující operace:
  - ⋆ násobení: používám rekurzivní algoritmus Karatsuba & Ofman, který převádí jedno násobení polynomů stupně $n$ na tři násobení polynomů stupně $n/2$ (a nějaká sčítání a odčítání). Tím dosahuje složitosti $\mathcal{O}(n^{\log_2 3} \approx n^{1.6})$. Jeho výhoda je v tom, že je jednoduchý a dokáže pracovat přímo nad tím tělesem, ze kterého jsou koeficienty násobených polynomů.
  - ⋆ dělení: pokud chceme vydělit polynomy $f$ a $g$, můžeme si předpočítat jakousi „pseudoinverzi" k polynomu $g$ a vynásobit jí s polynomem $f$. Takovou inverzi můžeme spočítat například metodou Newtonovy iterace v čase nejvýš tří násobení polynomů stupně $\deg g$. Celé dělení pak funguje v čase maximálně čtyř násobení, přičemž další dělení stejným polynomem už dokážeme provést v čase jednoho násobení.
    Popsaný postup má tu výhodu, že zlepšením algoritmu násobení se odpovídající měrou zlepší i složitost dělení.
  - ⋆ NSD: přestože existuje algoritmus pracující v čase $\mathcal{O}(M(n)\log n)$, je velmi komplikovaný a jeho složitost je ve skutečnosti omezena funkcí $24M(n)\log n$. Vzhledem k použité metodě násobení bychom se dostali na $24n^{1.6}\log n$, což by ani pro polynomy o řádově desetitisícových stupních nebylo lepší než triviální algoritmus se složitostí $\mathcal{O}(n^2)$.
    Proto jsem použil přímočarou metodu založenou na tom, že $\mathrm{NSD}(a,b) = \mathrm{NSD}(a - c\cdot b, b)$. Při výpočtu $\mathrm{NSD}(f,g)$ pro $\deg f \geq \deg g$ v jednom kroku spočtu $c = f_{\deg f}/g_{\deg g}$, $f_{\heartsuit} = f - c\cdot g\cdot x^{\deg f - \deg g}$ a pokračuji počítáním $\mathrm{NSD}(f_{\heartsuit}, g)$. Jeden takový krok zvládnu určitě v lineárním čase a snížím tak součet stupňů obou polynomů alespoň o jedna, takže je třeba ho zopakovat nanejvýš $(\deg f + \deg g)$-krát.
    Tento algoritmus popisuji tak podrobně proto, že použití klasické varianty $\mathrm{NSD}(f,g) = \mathrm{NSD}(f \bmod g, g)$ by nepřineslo tak dobrý výsledek, poněvadž přímočaré dělení se zbytkem trvá více než lineární čas a přitom by se muselo v nejhorším případě opakovat také $(\deg f + \deg g)$-krát.
- faktorizace polynomů: kromě upraveného algoritmu různostupňové faktorizace, který jsem už popsal, jsem použil přesně ty implementace popsané v předchozí kapitole o subkvadratickém algoritmu. Jejich asymptotická složitost je samozřejmě horší než subkvadratická, a to kvůli pomalejšímu algoritmu násobení polynomů, NSD dvou polynomů a pomalejšímu násobení matic.
- uživatelské rozhraní: pro komunikaci s uživatelem jsem zvolil příkazovou řádku a standardní vstup. Program je tak jednoduše použitelný a přenositelný mezi různými operačními systémy. Výpočet programu může být ale dlouhý, proto je možné zapnout zobrazování průběhu výpočtu.

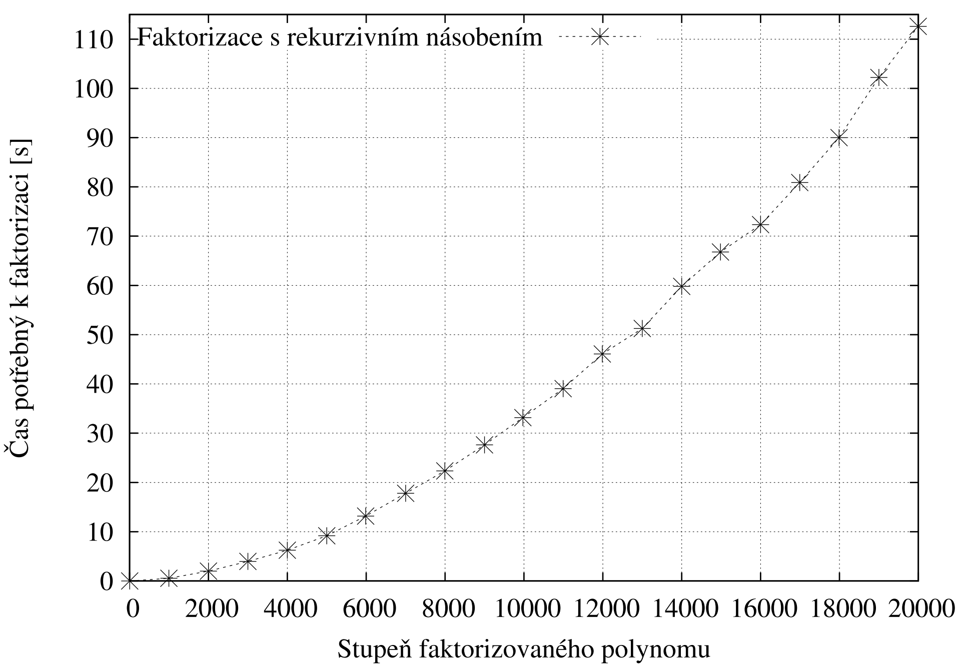
Pro představu skutečné časové náročnosti této implementace jsem na osobním počítači s procesorem AMD Athlon XP 2700+ vytvořil graf závislosti časové náročnosti faktorizace náhodných polynomů na jejich stupni.

Použité rekurzivní násobení má jednu zajímavou vlastnost. Přestože použitá reprezentace polynomů není řídká, časová složitost násobení klesá s počtem nenulových koeficientů v polynomu. Násobení tedy dokáže řídkost polynomů využít.

Tento fakt se projeví například při faktorizaci polynom $x^q - x$ nad tělesem $\mathbb{F}_q$, jejíž skutečnou časovou náročnost zachycuje další graf. Pro jeho vytvoření jsem, stejně jako pro vytvoření všech ostatních grafů, použil již zmiňovaný osobní počítač.




**Graf 4.1: Časová náročnost faktorizace náhodného polynomu**




**Graf 4.2: Časová náročnost faktorizace polynomu tvaru $x^q - x$ nad $\mathbb{F}_q$**

Operace, kterou faktorizace používá nejčastěji, je násobení polynomů. Experimentálně jsem zjistil, že přibližně 75% celého času faktorizace se tráví násobením polynomů. To je pochopitelné, protože dělení se převádí na násobení a velmi mnoho operací s polynomy se provádí modulo polynom.

Proto jsem se rozhodl zkusit toto násobení urychlit, a to pomocí jediného mě známého postupu – pomocí FFT, což je algoritmus provádějící diskrétní Fourierovu transformaci. Máme-li polynom stupně $n$, kde $n$ je mocnina dvojky, FFT ho dokáže v čase $\mathcal{O}(n \log n)$ vyčíslit v $n + 1$ speciálních bodech. Dva polynomy $f$ a $g$ můžeme vynásobit následovně: Označme $n = \deg f + \deg g$. Provedeme FFT na polynom $f$ stupně $n$ (dodáme nulové koeficienty na začátek) a ještě na polynom $g$ stupně $n$. Poté vynásobíme hodnoty polynomů $f$ a $g$ ve stejných bodech a provedeme inverzní FFT, čímž dostaneme koeficienty polynomu, který má v $n+1$ bodech spočtené součiny hodnot polynomů $f$ a $g$. Tento polynom je přesně součin $f \cdot g$.

Nad komplexními čísly je tento algoritmus (až na doplňování $n$ na nejbližší vyšší mocninu dvou) přímočarý a vždy proveditelný. Bohužel nad konečným tělesem tomu tak vždy není. FFT totiž požaduje, aby použité těleso (stačí dokonce okruh) obsahovalo $n$-tou primitivní odmocninu z jedničky. To je takové číslo $\omega$, že $\omega^n = 1$ a pro každé $1 \leq i < n$ je $\omega^i \neq 1$. Takové $\omega$ se v konečném tělese $\mathbb{F}_q$ vyskytuje právě tehdy, když $n \setminus q - 1$ (plyne to z cykličnosti multiplikativní grupy nenulových prvků tělesa).

Pokud násobíme polynomy nad $\mathbb{F}_q$ se součtem stupňů $n = 2^k$, FFT se tedy musí obecně provádět nad jiným než původním konečným tělesem, a to nad takovým, jehož počet prvků je ve tvaru $2^k \cdot c + 1$. Protože ale dostaneme výsledek v jiném tělese, musíme z něj ještě získat správné řešení v tělese původním. To se dá udělat několika způsoby:

- těleso, nad kterým bude FFT pracovat, bude mít alespoň $(n+1)(q-1)^2 \approx nq^2$ prvků. Toto je největší koeficient, který se může objevit v součinu dvou polynomů z $\mathbb{F}_q$, protože koeficient u $x^i$ dostaneme jako $\sum_{j=0}^{i} f_j \cdot g_{i-j} \leq (n+1)(q-1)^2$. Koeficienty výsledného polynomu nad tělesem $\mathbb{F}_q$ pak získáme jako zbytky po celočíselném dělení koeficientů polynomu nad větším tělesem číslem $q$.
  Tento postup má výhodu v tom, že mu stačí pouze jedno násobení pomocí FFT. Bohužel použité těleso je o mnoho větší než těleso původní.
- na předchozím způsobu nám vadilo, že potřebuje pro FFT moc velké těleso. Necháme tedy FFT pracovat nad dvěma tělesy, přičemž každé z nich bude mít alespoň $\sqrt{n+1}(q-1)$ prvků a jejich velikosti budou navzájem nesoudělné. Nad oběma z nich dané polynomy pomocí FFT vynásobíme. O koeficientech výsledného polynomu budeme pak vědět, jaké jsou jejich zbytky po dělení dvěma nesoudělnými čísly velikosti alespoň $\sqrt{n+1}(q-1)$, takže dle Čínské věty o zbytcích budeme znát i jejich zbytek po dělení součinem těchto čísel, čili číslem o velikosti alespoň $(n+1)(q-1)^2$. Pak už stačí vzít zbytky koeficientů po dělení číslem $q$ a získáme hledaný polynom.
  Tento postup už požaduje menší tělesa než postup první, ale FFT nad nimi musí počítat dvakrát.

Teoreticky bychom mohli používat stále menší tělesa a stále více FFT, ale dostali bychom se do problému, pokud by velikost těchto těles byla menší než $q$, velikost původního tělesa. Pak totiž už nedokážeme původní polynomy jednoznačně reprezentovat.

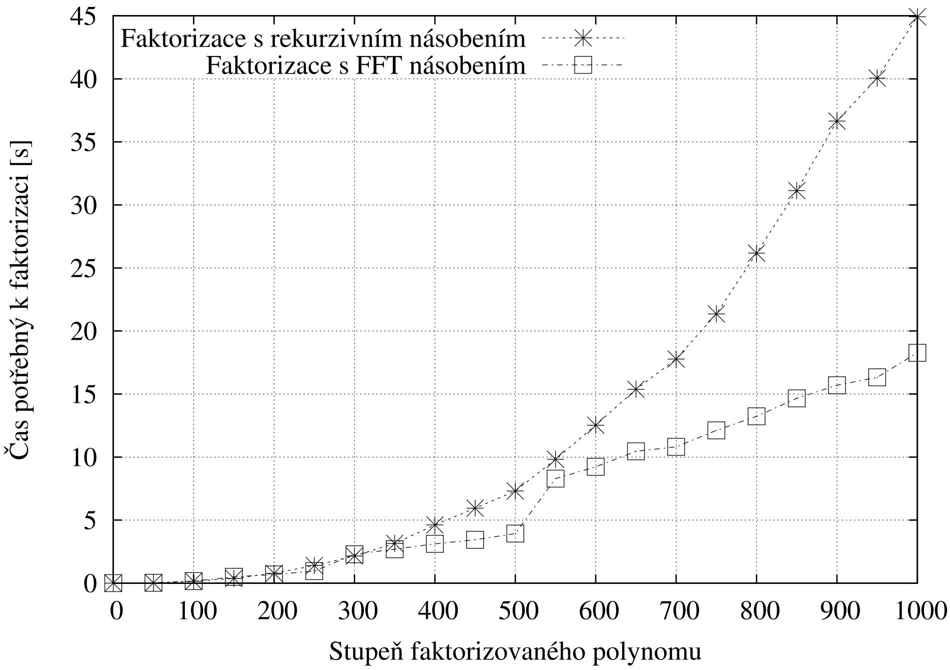
Tělesa, nad kterými budeme provádět FFT, a $n$-té primitivních odmocniny z jedničky v nich musíme navíc umět efektivně nalézt.

Já jsem pro jednoduchost použil první variantu a problém s hledáním těles pro FFT jsem přesunul do kompetence implementace konečných těles. Pro svou vlastní implementaci konečných těles, která dovoluje jen tělesa o prvočíselné velikosti do $2^{31} - 1$, jsem našel největší rozumné prvočíslo požadovaného tvaru, $15 \cdot 2^{27} + 1 = 2013265921$, a jeho $2^{27}$-primitivní odmocninu z jedničky, $\omega = 440564289$. FFT vždy provádím nad $\mathbb{F}_{2013265921}$ a jen pro $n \leq 2^{27}$. Tento postup není příliš praktický z hlediska uživatele, ale ke změření časové náročnosti faktorizace s touto metodou násobení se zcela dostačující. Navíc by nebyl velký problém implementaci upravit tak, aby používala složitější variantu s FFT násobením nad dvěma různými tělesy (jejichž velikost by také byla pevná), přičemž její časová náročnost by se nejhůř zdvojnásobila.

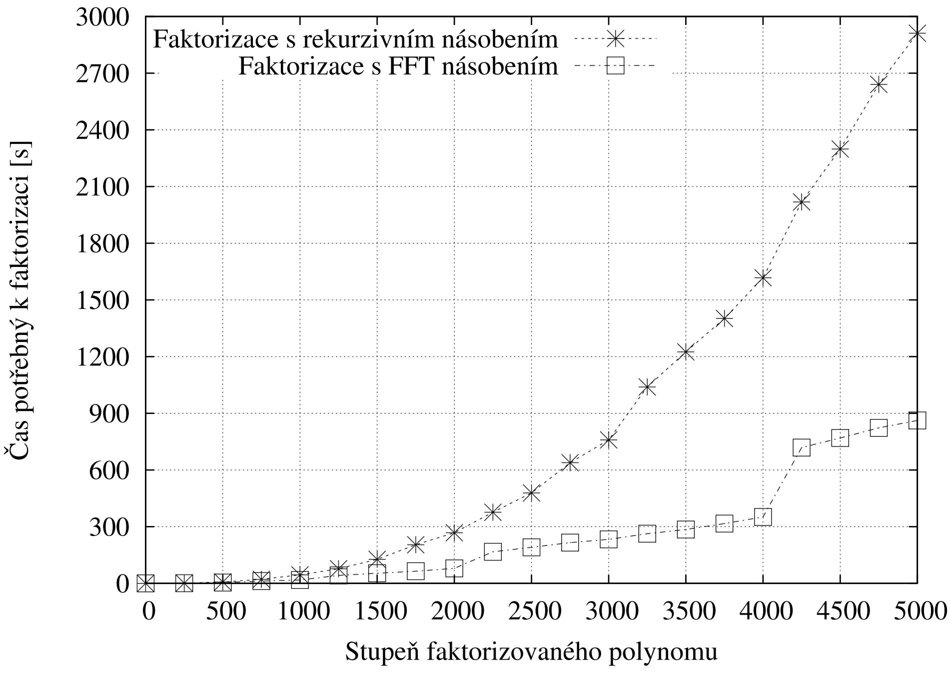
Výsledky implementace s FFT násobením prezentují následující dva grafy. Z nich je vidět, že časová složitost je jak asymptoticky, tak skutečně lepší v případě násobení založeného na FFT. Dokonce i v případě složitější varianty s dvěma tělesy pro FFT by tato varianta byla rychlejší pro polynomy stupně řádově tisíc a více.

Metoda založená na FFT má ale i nevýhody. V aktuální implementaci povoluje násobení jen nad velmi malými tělesy (dalo by se vyřešit implementací složitější metody).

Dále násobení pomocí FFT vůbec nezohledňuje řídkost polynomu, což rekurzivní násobení dělalo. Navíc se časová náročnost FFT výrazně skokově zvětší po „přeskočení“ mocniny dvou.




**Graf 4.3: Porovnání implementací faktorizace pro polynomy do stupně 1000**




**Graf 4.4: Porovnání implementací faktorizace pro polynomy do stupně 5000**

Protože mají obě použité metody své kladné stránky, program dokáže použít libovolnou z nich podle přání uživatele.

Program, jeho zdrojový kód a podrobnější informace o jeho používání lze nalézt na přiloženém CD nebo na webových stránkách `http://www.ucw.cz/~fox/work/factoring/`.

# 5. Použitá literatura

[1] Agrawal, M., Kayal, N., Saxena, N.: *PRIMES is in P,* Annals of Mathematics, vol. 160, no. 2, 781−793, 2004.

[2] Berlekamp, R. E.: *Factoring polynomials over large finite fields,* Mathematics of Computation, vol. 24, no. 111, 713−735, 1970.

[3] Cantor, D. G., Zassenhaus, H.: *A new algorithm for factoring polynomials over finite fields,* Mathematics of Computation, vol. 36, no. 154, 587−592, 1981.

[4] Coppersmith, D. and Winograd, S.: *Matrix multiplication via arithmetic progressions,* Journal of Symbolic Computation, vol. 9, no. 3, 251−280, 1990.

[5] von zur Gathen, J. and Gerhard, J.: *Modern Computer Algebra,* Cambridge University Press, ISBN 0521826462, 2003.

[6] von zur Gathen, J. and Shoup, V.: *Computing Frobenius maps and factoring polynomials,* Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, 97−105, 1992.

[7] Geddes, K. O., Labahn, G. and Czapor S. R.: *Algorithms for Computer Algebra,* Springer, ISBN 0792392590, 1992.

[8] Kartofen, E. and Shoup, V.: *Subquadratic-time factoring of polynomials over finite fields,* Mathematics of Computation, vol. 76, no. 223, 1179−1197, 1998.

[9] Rabin, M. O.: *Probabilistic algorithms in finite fields,* SIAM Journal of Computing, vol. 9, no. 2, 273−280, 1980.

[10] Shoup, V.: *A New Polynomial Factorization Algorithm and its Implementation,* Journal of Symbolic Computation, vol. 20, no. 4, 363−397, 1995.

[11] Yun, D. Y. Y.: *On square-free decomposition algorithms,* Proceedings of the third ACM symposium on Symbolic and algebraic computation, 26−35, 1976.