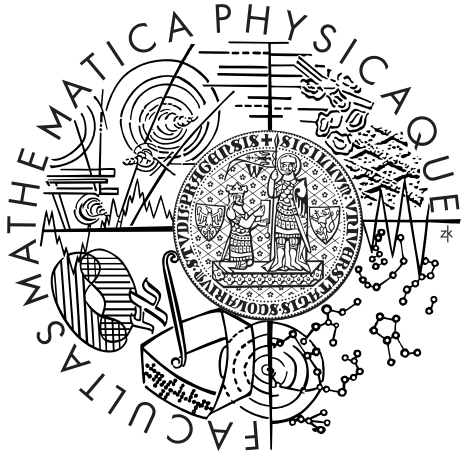
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Adéla Zajíčková

QR kód a jeho dekódování

Katedra algebry

Vedoucí bakalářské práce:	prof. RNDr. Drápal Aleš, CSc., DSc.
Studijní program:	Matematika
Studijní obor:	Matematické metody informační bezpečnosti

Praha 2013

Ráda bych poděkovala mému vedoucímu práce prof. RNDr. Alešovi Drápalovi, CSc., DSc. za konzultace a cenné rady. Dále Marianovi Kechlibarovi, Jiřímu Nekolovi a Petrovi Dopitovi za trpělivost a pomoc při práci s knihovnou ZXing. A nakonec Zdeňkovi Haníkovi a Vladimíře Zajíčkové za kontrolu a podporu při psaní práce.

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

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

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

Název práce: QR kód a jeho dekódování

Autor: Adéla Zajíčková

Katedra: Katedra algebry

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

Abstrakt: Práce se zabývá popisem QR kódů. Nejprve se zaměřuje na popis struktury, a poté se věnuje dekódování, konkrétně té části, která využívá teorii samoopravných kódů. Z hlediska této teorie mohou být QR kódy vnímány jako případ Reed-Solomonových kódů. Proto se práce věnuje základním vlastnostem RS kódů a jednomu ze způsobů jejich dekódování, konkrétně Euklidovu dekódovacímu algoritmu. Poznatky jsou ukázány na konkrétním příkladu a dále je práce doplněna detailním popisem matematického principu konkrétního použitého algoritmu.

Klíčová slova: QR kód, Reedův-Solomonův kód, Samoopravné kódy

Title: QR Code and decoding

Author: Adéla Zajíčková

Department: Department of Algebra

Supervisor: prof. RNDr. Drápal Aleš, CSc., DSc. , Department of Algebra

Abstract: The aim of this paper is to descript QR codes. First of all we focus on a description of a structure and then we deal with decoding, more specifically with the part which uses the theory of error-correcting codes. From the viewpoint of this theory QR codes can be perceived as an instance of Reed-Solomon codes. For that reason the paper deals with elementary properities of RS codes and with one of the technique of decoding, concretely Euclidean decoding algorithm. An example shows the knowledge and futhermore the paper includes detailed description of mathematical principle of used algorithm.

Keywords: QR code, Reed-Solomon code, Error-correcting codes

Obsah

Úvod

QR kód nebo-li Quick Response kód je typ 2-D čárového kódu, který byl vynalezen roku 1994 v Japonsku společností DENSO WAVE pro automobilový průmysl. 2-D čárový kód znamená, že data jsou uložena vertikálně i horizontálně; QR kód díky tomu obsáhne na stejné ploše mnohonásobně více dat než klasické čárové kódy, které známe ze zboží v obchodech. Při jeho navrhování se kromě obsažnosti kódu myslelo také na to, aby se dal přečíst rychle, z jakéhokoli úhlu, a aby byl alespoň zčásti odolný proti poškození. K jeho přečtení stačí obyčejný mobilní telefon s fotoaparátem a příslušnou aplikací, a proto se dostal postupně i na veřejnost. Do QR kódu se dají například ukrýt kontaktní informace vytištěné na vizitce a pouhým vyfocením, bez jakéhokoli přepisování, je možné je uložit jako kontakt. S nástupem internetu do mobilu se ale do QR kódů mnohem častěji ukládají webové adresy.

Aby byla splněna podmínka „být odolný proti poškození“, využívá se technologie samoopravných kódů, konkrétně Reed-Solomonových kódů. Cílem této práce je popsat stavbu QR kódu, princip dekódování a zaměřit se na přesný popis algoritmu pro dekódování Reed-Solomonova kódu.

Tím, jak QR kód vypadá, jaké části musí obsahovat a proč, se zabývá kapitola 1. Veškeré technické informace a nástin dekódovacího algoritmu, který je zde popsán, jsou shrnuty z [1].

V kapitole 2 jsou popsány Reed-Solomonovy kódy (zkráceně RS). Pro četbu této kapitoly se předpokládá, že čtenář zná základy teorie samoopravných kódů. Kromě zavedení RS kódů se tu zabýváme i způsobem jejich dekódování. Existuje několik dekódovacích algoritmů. Jelikož ale jedním z cílů této práce bylo osvojit si a ověřit výkonnost dekodéru, jehož struktura by byla v rámci práce plně prozkoumána a v přiměřené míře okomentována, zabýváme se zde pouze jedním algoritmem; a to tím, který je použit v programu. Popis kódu z kapitoly 1 a algoritmus dekódování, popsaný a zdůvodněný v kapitole 2, jsou v kapitole 3 dány do souvislosti a komentovány na konkrétním příkladu. Pro její účely byl vytvořen poškozený QR kód, který je postupně dekódován. Algoritmus kapitoly 3 se v některých částech liší od stručného algoritmu z kapitoly 1. Je to dáno tím, že v kapitole 3 již popisujeme implementaci knihovny ZXing.

V závěrečné kapitole 4 je implementace testována na pořízených datech. Jsou zde porovnávány rozdíly mezi dekódováním různě velkých a různě poškozených QR kódů.

Kapitola 1

O QR kódech

Název Quick Response znamená v překladu „rychlá odezva“ a poukazuje na to, že kódy lze načíst a dekódovat velmi rychle. Napomáhá tomu hlavně skutečnost, že čtečka je schopna detekovat a přečíst QR kód i při jeho libovolném otočení vůči čtecímu zařízení, což je zajištěno třemi značkami v rozích a dalšími pomocnými značkami. Význam a umístění těchto značek a dalších částí QR kódu popíšeme v této kapitole.

Část 1.1 vychází ze stránky oficiálního webu [4]. Zbylé informace jsou čerpány z ISO normy [1], ve které jsou kódy detailně popsány. Je zde popsán i algoritmus pro detekování kódu v obrázku, ale samotným dekódováním už se norma příliš nezabývá. Krom QR kódů jsou v nejnovější normě popsány i micro QR kódy, které, jak název napovídá, jsou menší verzí QR kódů. Micro QR kódy se v této práci zabývat nebudeme.

1.1 Historie a současnost

Jak bylo zmíněno v úvodu, QR kódy byly vynalezeny již roku 1994. Je to produkt společnosti DENSO WAVE INCORPORATED (viz. webová adresa [7]), což je dceřiná společnost DENSO Corporation (viz. webová adresa [8]), která se zabývá výrobou automobilových dílů a je členem Toyota Group. V automobilové továrně byly QR kódy také poprvé použity, k označování jednotlivých autodílů, pro které nejspíš nestačily klasické čárové kódy.

Nyní, s nástupem chytrých mobilních telefonů a mobilního internetu, se ukázaly být QR kódy skvělým doplňkem reklamy. Málokomu se chce přepisovat webovou adresu z reklamního panelu do mobilu a než dorazí k počítači nebo notebooku, na reklamu zapomene; naskenovat ale adresu pomocí čtečky a otevřít si ji v mobilu je otázka několika vteřin. Proto se objevují na reklamách i různých výrobcích stále častěji.

1.2 Data v QR kódech

QR kód se skládá z mnoha malých bílých nebo černých čtverečků, těmto čtverečkům budeme v následujícím textu říkat moduly. Při načtení QR kódu pak každý modul reprezentuje jeden bit, černý 0 a bílý 1.

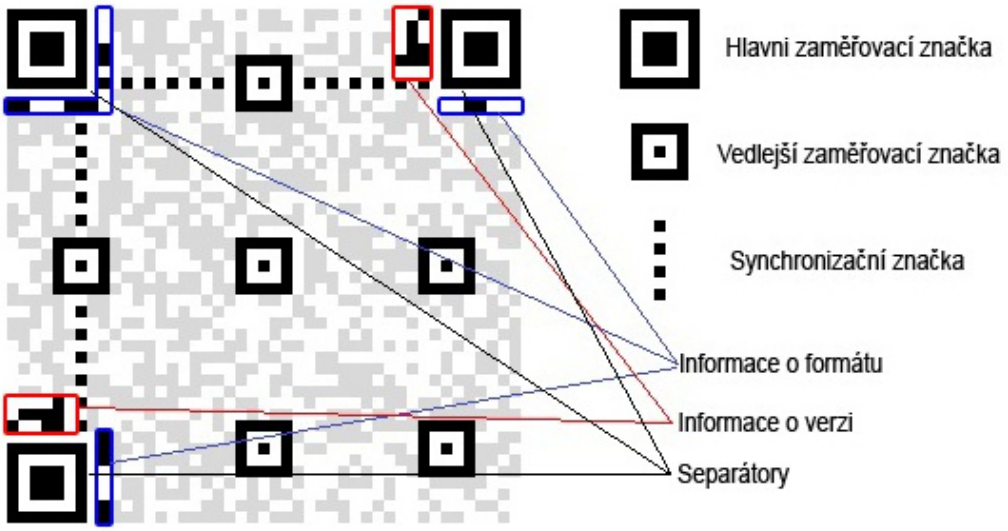
QR kódy můžou obsahovat data v několika různých formátech a od toho

se také odvíjí maximální kapacita QR kódu. Můžeme kódovat numerická data, alfanumerická data, klasický text v kódování UTF-8 nebo znaky Kanji.

V alfanumerickém formátu je k dispozici 45 různých znaků, mezi které patří číslice 0-9, znaky A-Z, mezera a znaky $, %, *, +, -, ., /, :. Nejpoužívanějším formátem bude ale pravděpodobně kódování textu v UTF-8. Potom každý znak zabírá jeden byte, proto se tento formát nazývá bytový.

QR kódy mají velké rozpětí objemu ukládaných dat, tomu odpovídá 40 různých verzí. V každé z nich lze navíc nastavit jednu ze čtyř hodnot „úrovně zabezpečení“(error correction level). Tyto hodnoty jsou značeny L, M, Q, H. Podle dokumentace zajišťují po řadě odolnost proti poškození až 7%, 15%, 25% či 30% plochy QR kódu. Nejmenší verze 1 pak pojme 17 bytů dat při stupni opravy chyb L či 7 bytů při stupni opravy chyb H, zatímco nejvyšší verze 40 pojme 2953 bytů při stupni opravy chyb L a 1273 při stupni opravy chyb H.

1.3 Struktura QR kódu



Obrázek 1.1: QR kód verze 7, úroveň zabezpečení L

Hlavní zaměřovací značky (Finder pattern)

Hlavní zaměřovací značky jsou nejnápadnější znak QR kódu, díky kterému je možné detekovat QR kód libovolně otočený. Značky jsou vždy 3, v levém horním, levém dolním a pravém horním rohu, a jsou identické. Můžeme je vnímat jako tři překrývající se čtverce, z nichž největší je černý a má rozměry 7×7 modulů. Na něm leží bílý s rozměry 5×5 modulů a nakonec je černý 3×3, přičemž tyto čtverce mají společný střed. Čtečky QR kódu jako první hledají tento vzor a pokud najdou tyto tři značky, otočí si snímek QR kódu tak, aby byly na správných pozicích.

Separátor (Separator)

Pruhy široké jeden modul a složené pouze z bílých modulů, které jsou umístěny okolo každé hlavní zaměřovací značky a oddělují ji od oblasti, ve které jsou uložena data.

Vedlejší zaměřovací značky (Alignment patterns)

Vedlejší zaměřovací značky vypadají skoro stejně jako hlavní zaměřovací značky, jen jsou menší. Největší černý čtverec má rozměry 5×5 modulů, v něm je bílý s rozměry 3×3 moduly a uprostřed něj leží samostatný černý modul. V kódu verze 1 se jako v jediném nenachází žádná, v dalších verzích může být 1, 6, 13, 22, 33 nebo 46 značek. Platí zde, že čím vyšší verze, tedy čím větší plocha kódu, tím více vedlejších zaměřovacích značek. Slouží hlavně k detekci zkosení u velkých kódů.

Synchronizační značky (Timing pattern)

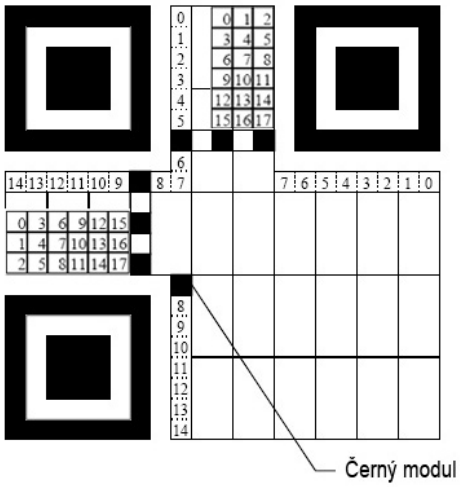
Jsou dva pruhy, horizontální a vertikální, široké jeden modul tvořené pravidelně se střídajícími bílými a černými moduly. Každý z pruhů začíná i končí černým modulem. Horizontální pruh tvoří 6. řádek QR kódu a vertikální 6. sloupec. Slouží k určení velikosti modulu na obrázku, k určení provizorní verze (viz. bod 1, sekce 1.4) kódu a jako jakési pomocné souřadnice.

Informace o formátu (Format information)

Informace o formátu jsou rozmístěny okolo všech hlavních zaměřovacích značek. Jsou tvořeny 15 bity a reprezentovány tedy 15 moduly. Samotné informace ale zabírají pouze pět bitů. První dva označují úroveň zabezpečení (01=L, 00=M, 11=Q, 10=H) a zbylé tři udávají kód masky (viz 1.3). Zbylých 10 bitů je dopočítáno pomocí [15,5] BCH kódu. Aby se nemohlo stát, že vznikne nulové slovo, je na závěr binárně přičtena maska 101010000010010. Protože jsou tato data klíčová, jsou nejen chráněna samoopravným BCH kódem, ale navíc uložena na dvou místech. Prvních 15 bitů obklopuje separátor levé horní zaměřovací značky, druhých 15 bitů je rozděleno, přičemž nultý až sedmý bit tvoří řádek pod pravou horní hlavní zaměřovací značkou, zatímco osmý až čtrnáctý tvoří sloupec napravo od levé dolní hlavní zaměřovací značky. Tento sloupec má výšku 8 bitů, ale je tam uloženo pouze 7 bitů informace, poslední, nejvýše položený bit se nechává vždy černý (viz obrázek 1.2). Informace o formátu nepřekrývají, ale „přeskakují“ synchronizační značky, jak je vidět na obrázku.

Informace o verzi (Version information)

Informace o verzi jsou v QR kódech verze 7 a výše. Jsou tvořeny osmnácti bity, z nichž šest bitů jsou data, v tomto případě číslo verze, zbylých dvanáct bitů je dopočítáno pomocí [18,6] Golayova kódu. Navíc jsou také uloženy na dvou různých místech. 18 modulů je poskládáno do obdélníku 3×6 a umístěno jednou horizontálně nad levou dolní hlavní zaměřovací značkou a podruhé vertikálně nalevo od pravé horní hlavní zaměřovací značky tak, že svou kratší stranou vždy

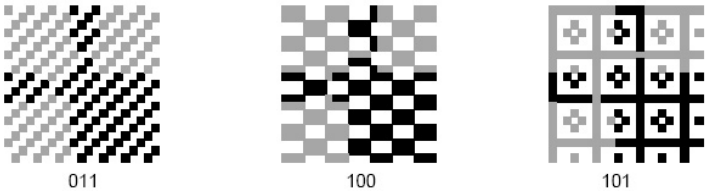


Obrázek 1.2: Uložení informací o formátu a o verzi

přiléhají k pruhu synchronizační značky. Jak přesně jsou bity uloženy, je zobrazeno na obrázku 1.2. Obrázek je pouze ilustrační, neboť bity jsou vyznačeny na QR kódu verze 1, ve které ve skutečnosti nejsou.

Maska (Mask pattern)

Na plochu každého QR kódu je binárně přičten jeden z osmi různých černobílých vzorů, kterým se říká masky. Maska je vybírána vždy tak, aby poměr mezi černými a bílými moduly byl co nejblíže poměru 50:50 a zároveň tak, aby nevznikaly vzory, které by připomínaly hlavní nebo vedlejší zaměřovací značky a ztěžovaly tak detekování kódu. Maska se neaplikuje na zaměřovací značky, synchronizační značky, ani na informace o verzi a formátu.



Obrázek 1.3: Některé z možných masek i s jejich identifikátory

1.4 Čtení QR kódu

Nyní stručně popíšeme algoritmus dekódování QR kódu. Podrobněji pak bude popsán na konkrétním příkladu v kapitole 3. Tam budou popsány i některé části implementace v jazyku C++. Implementace byla převzata z projektu ZXing ("Zebra crossing"), jenž lze najít na webové adrese [6], a upravena v některých detailech.

1. Lokalizace QR kódu na obrázku, rozlišení tmavých a světlých modulů a pro další zpracování převedení na matici bitů. Z velikosti matice je určeno provizorní číslo verze.

2. Přečtení informací o formátu. Tím získáme použitou úroveň zabezpečení a masku.

3. Přečtení informací o verzi, pokud je provizorní číslo verze větší než 6.

4. Naxorování masky určené z informací o formátu na oblast obsahující data.

5. Načtení veškerých dat podle pravidel pro danou verzi a rozdělení do jednotlivých bloků.

6. Detekce a případná oprava chyb v každém bloku.

7. Sestavení původní zprávy.

Krokem číslo 1 se zabývat nebudeme, neboť je to téma zpracování obrazu a samo o sobě by bylo dosti rozsáhlé. Naším cílem v dalších kapitolách je detailně rozebrat krok 6.

1.5 Korekce chyb

Už víme, že u QR kódu se dá zvolit míra odolnosti proti poškození, které může vzniknout například zmuchláním papíru, na kterém je QR kód vytištěný, nebo špatným odleskem světla při snímání fotoaparátem mobilu. Tuto odolnost zajišťují tzv. samoopravné kódy; v našem případě to jsou Reed-Solomonovy kódy, které chrání data, [15,5] BCH kód, který chrání informace o formátu, a [18,6] Golayův kód, který chrání číslo verze.

Kapitola 2

Reed-Solomonovy kódy

Roku 1960 vyšel v Journal of the Society for Industrial and Applied Mathematics (SIAM) článek I. S. Reeda a G. Solomona s názvem „Polynomial codes over certain finite fields“ [3], v překladu polynomiální kódy nad určitými konečnými tělesy. V tomto článku byly popsány kódy, kterým se dnes říká Reed-Solomonovy (zkráceně RS). RS kódy jsou lineární, MDS, odolné proti shlukujícím se chybám a existuje pro ně efektivní dekódovací algoritmus. To všechno jsou důvody, proč se tyto kódy staly jedněmi z nejpoužívanějších v moderní době.

Kapitola vychází převážně z knihy [2] kapitoly 5 a 6 a část 2.2 čerpá z ISO normy [1]. Výklad byl doplněn o koevaluační polynom, převzatý z [5] kapitoly 5.3; tím se formálně značně zjednodušil důkaz platnosti klíčové rovnice. Dále jsem přidala důkaz k lemmatu 2.6, lemma 2.3 i s důkazem a více rozepsala některé části důkazu tvrzení 2.7.

V kapitole se předpokládá znalost základních pojmů teorie lineárních samoopravných kódů jako je dimenze, prověrková matice apod. Kromě těchto elementárních pojmů je ještě třeba znát definici MDS kódu: je to každý $[n,k,d]_q$, ve kterém $d = n - k + 1$. MDS kódy jsou tedy právě ty kódy, ve kterých Singletonova nerovnost $d \leq n - k + 1$ platí jako rovnost.

2.1 Konstrukce RS kódů

Původní myšlenka RS kódů uveřejněná v [3] byla velmi prostá. Mějme těleso $\mathbb{F}_q$, kde $q = 2^n$, prvek α, který generuje $\mathbb{F}_q$, a zprávu $\mathbf{m} = (m_0,...,m_{s-1}), m_i \in \mathbb{F}_q, s < q$. Vytvoříme polynom $P(x) = m_0 + m_1x + ... + m_{s-1}x^{s-1}$ a m-tici $(m_0,...,m_{s-1})$ kódujeme následovně:

$$(m_0,...,m_{s-1}) \to (P(0), P(\alpha), P(\alpha^2),...,P(\alpha^{q-2}), P(1)).$$

Množina zpráv odpovídá vektorům koeficientů polynomů do stupně s nad $\mathbb{F}_q$, což je vektorový prostor. Zobrazení zprávy na kódové slovo je definováno v každé složce dosazovacím homomorfismem $F_q[x] \to F_q$, což je homomorfismus okruhů, který je zároveň lineární formou. Proto je celé zobrazení homomorfismem vektorových prostorů, takže množina zpráv, která je jeho obrazem, tvoří lineární kód. Jeho dimenze je rovna s, neboť uvedený homomorfismus má zjevně triviální jádro.

Takový kód není ovšem cyklický. Jak se rozvíjela teorie cyklických kódů i samoopravných kódů obecně, náhled na RS kódy i jejich definice se změnily a

vznikly krom klasických RS kódů také generalizované Reed-Solomonovy kódy (zkráceně GRS).

V dnešní době vypadají nejpoužívanější definice takto:

Definice 2.1. *Nechť $n > k \geq 1$, $\alpha_1,...,\alpha_n$ jsou po dvou různé nenulové prvky z $\mathbb{F}_q$ a $v_1,...,v_n \in \mathbb{F}_q$ nenulové prvky.* Generalizovaný Reed-Solomonův kód *nad $\mathbb{F}_q$ je lineární $[n,k,d]_q$ kód s prověrkovou maticí*

$$H = \begin{pmatrix} 1 & 1 & ... & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \cdots & \alpha_n^{n-k-1} \end{pmatrix} \begin{pmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_n \end{pmatrix}.$$

Prvky α_i se nazývají lokátory a v_i multiplikátory.

Pokud $n = q - 1$, nazýváme GRS kód primitivní.

Pokud $v_i = 1\ \forall i = 1,...,n$, nazýváme GRS kód normalizovaný.

Definice 2.2. *Nechť $n \in \mathbb{N}$ takové, že $n \mid q-1$, $\alpha \in \mathbb{F}_q$ řádu n.* Reed-Solomonův *$[n,k]_q$ kód je GRS kód s lokátory $\alpha_j = \alpha^j, 1 \leq j \leq n$ a multiplikátory $v_j = \alpha^{b(j-1)}, 1 \leq j \leq n$.*

Pokud $b = 0$, mluvíme o RS kódu v normalizovaném tvaru.

Je-li $b = 1$, nazýváme kód RS kódem v užším slova smyslu.

Vidíme, že GRS kódy jsou lineární přímo z definice, neboť jen pro lineární kódy je definována prověrková matice. Dokážeme ještě, že to jsou MDS kódy.

Lemma 2.3. *(Vandermondův determinant) Nechť $\beta \in \mathbb{F}_q$ a $m > 1$. Determinant Vandermondovy matice*

$$\begin{pmatrix} 1 & \beta_1 & \beta_1^2 & \cdots & \beta_1^{m-1} \\ 1 & \beta_2 & \beta_2^2 & \cdots & \beta_2^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \beta_m & \beta_m^2 & \cdots & \beta_m^{m-1} \end{pmatrix}$$

je roven $\prod_{1 \leq i < j \leq m} (\beta_j - \beta_i)$.

Důkaz. Důkaz provedeme indukcí. Pro $m = 2$ je Vandermondova matice tvaru $\begin{pmatrix} 1 & \beta_1 \\ 1 & \beta_2 \end{pmatrix}$ a determinant roven $\beta_2 - \beta_1$. Mějme Vandermondovu matici řádu m a nechť lemma platí pro $m - 1$. Nejprve využijeme toho, že přičteme-li k sloupci libovolnou lineární kombinaci ostatních sloupců, determinant se nezmění, a postupně vždy od k-tého sloupce odečteme $k - 1$ sloupec vynásobený prvek β_1, pro $k = 2...m$:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & \beta_2 - \beta_1 & (\beta_2 - \beta_1)\beta_2 & \cdots & (\beta_2 - \beta_1)\beta_2^{m-2} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \beta_m - \beta_1 & (\beta_m - \beta_1)\beta_m & \cdots & (\beta_m - \beta_1)\beta_m^{m-2} \end{pmatrix}. \tag{2.1}$$

Využijeme rozvoj determinantu podle prvního řádku, takže determinant matice (2.1) je roven determinantu (2.2).

$$\begin{vmatrix} \beta_2-\beta_1 & (\beta_2-\beta_1)\beta_2 & \cdots & (\beta_2-\beta_1)\beta_2^{m-2} \\ \beta_3-\beta_1 & (\beta_3-\beta_1)\beta_3 & \cdots & (\beta_3-\beta_1)\beta_3^{m-2} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_m-\beta_1 & (\beta_m-\beta_1)\beta_m & \cdots & (\beta_m-\beta_1)\beta_m^{m-2} \end{vmatrix} = \tag{2.2}$$

$$= (\beta_2-\beta_1)(\beta_3-\beta_1)...(\beta_m-\beta_1)\begin{vmatrix} 1 & \beta_2 & \beta_2^2 & \cdots & \beta_2^{m-2} \\ 1 & \beta_3 & \beta_3^2 & \cdots & \beta_3^{m-2} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \beta_m & \beta_m^2 & \cdots & \beta_m^{m-2} \end{vmatrix} \tag{2.3}$$

A to je podle indukčního předpokladu rovno $\prod_{1\le i<j\le m}(\beta_j-\beta_i)$.

□

Věta 2.4. *Generalizovaný Reed-Solomonův $[n,k,d]_q$ kód je MDS kód.*

Důkaz. V důkazu využijeme krátké lemma: Pro lineární kód je d největší číslo takové, že každých $d-1$ sloupců prověrkové matice je nezávislých.

Pak tedy kód je MDS $\Leftrightarrow d = n-k+1 \Leftrightarrow$ každých $n-k$ sloupců prověrkové matice je lineárně nezávislých. Vezměme si tedy libovolnou $(n-k)\times(n-k)$ podmatici prověrkové matice H:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \cdots & \beta_{n-k} \\ \beta_1^2 & \beta_2^2 & \cdots & \beta_{n-k}^2 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_1^{n-k-1} & \beta_2^{n-k-1} & \cdots & \beta_{n-k}^{n-k-1} \end{pmatrix} \begin{pmatrix} v_{i_1} & 0 & \cdots & 0 \\ 0 & v_{i_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_{i_{n-k}} \end{pmatrix}.$$

$\beta_i = \alpha_j$ pro nějaké j. Z lineární algebry víme, že determinant součinu je součin determinantů. Determinant diagonální matice je součin prvků na diagonále, tedy $\prod_{j=1}^{n-k} v_{i_j}$, a je nenulový. Podíváme-li se na druhou matici, zjistíme, že je to matice Vandermondova, jejíž determinant známe a vypadá následovně: $\prod_{1\le i<j\le n-k}(\beta_j-\beta_i)$. Vzhledem k tomu, že α_i jsou vzájemně různé jsou i β_i navzájem různé a všechny členy součinu jsou nenulové. Tudíž i determinant je nenulový a sloupce matice jsou lineárně nezávislé.

□

2.2 GRS kódy v QR kódech

Dle [1] bylo pro QR kódy q zvoleno jako $2^8 = 256$. Důvod je prostý, nejjednodušší při práci s počítačem je pracovat s byty. Prvky tělesa $\mathbb{F}_{2^8}$ jsou reprezentovány polynomy nad $\mathbb{F}_2$ stupně menšího než 8, přičemž násobení dvou prvků se

provádí modulo $x^8 + x^4 + x^3 + x^2 + 1$. Vzhledem k velkému počtu verzí a možnosti vybrat si úroveň zabezpečení je použito mnoho různých GRS kódů, všechny jsou však nad stejným tělesem. Nejdelším z použitých kódů je [153,123,31] kód, například ve verzi 38 L.

Jednotlivé kódy jsou určeny generujícími polynomy, jejichž výčet najdeme v [1]. Celkem jich je 33, nejmenší stupeň je 7, naopak nejvyšší je 68 a jsou dány následujícím předpisem:

$$g_h(x) = (x-1)(x-\alpha)...(x-\alpha^{h-1}),$$

kde h je stupeň polynomu, $h = n - k$ a α je primitivní prvek $\mathbb{F}_{2^8}$; α zastupuje x z popisu reprezentace a je kořenem polynomu $x^8 + x^4 + x^3 + x^2 + 1$:

$$\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0 \pmod{\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1}.$$

V [1] je generující polynom definován jako $g_h(x) = (x-1)(x-2)...(x-2^{h-1})$, což nedává v popsané reprezentaci tělesa $\mathbb{F}_{2^8}$ smysl. Je to míněno tak, že prvek tělesa reprezentován polynomem je zapsán jako vektor koeficientů. Pak 2 odpovídá řetězci 00000010, tedy polynomu α.

Pro libovolný kód použitý v QR kódu jsou kódová slova všechny polynomy délky n, které jsou násobky generujícího polynomu $g(x)$ stupně $n - k$. Pokud tedy chceme zjistit, zda jsme obdrželi kódové slovo, stačí zjistit, zda polynom $c(x)$ určený kódovým slovem je násobkem $g(x)$. To platí právě tehdy, když polynom $c(x)$ má za kořeny $1, \alpha,...,\alpha^{n-k-1}$. Prověrková matice tedy vypadá následovně:

$$\begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^{2\cdot 2} & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k-1} & \alpha^{2(n-k-1)} & \cdots & \alpha^{(n-k-1)(n-1)} \end{pmatrix}. \tag{2.4}$$

Je tedy vidět, že jsou zde použity GRS kódy, jejichž multiplikátory jsou vždy všechny rovny jedné a lokátory α_i se rovnají α^{i-1} pro všechna $i \in \{1,...,n\}$. V ISO normě jsou kódy označovány jako RS kódy a mohlo by se zdát, že tomu tak opravdu je. Multiplikátory by v tom případě byly tvaru $v_i = \alpha^{-1}$. Problém je, že pro všechny použité kódy je α řádu 255 a libovolný RS kód by měl délku 255; takovou délku nemá ale žádný z použitých kódů. Nesoulad mezi [1] a definicí 2.2 svědčí o tom, že terminologie není zcela ustálena.

2.3 Dekódování GRS kódů

Proces dekódování GRS kódů popíšeme na kódu s prověrkovou maticí (2.4), kterou budeme značit H. Ať $\mathbf{r} = (r_1,...,r_n)$ je naše přijaté slovo a $\mathbf{e} = (e_1,...,e_n)$ chybové slovo; víme, že platí $\mathbf{r} = \mathbf{m} + \mathbf{e}$. Dále víme, že v $\mathbf{e}$ jsou nenulové právě ty pozice, na kterých se vyskytly chyby. Označme J množinu těchto pozic, takže $e_\kappa \neq 0 \Leftrightarrow \kappa \in J$. Předpokládejme, že $|J| \leq \frac{1}{2}(d-1)$, abychom dekódovali slovo správně.

Prvním krokem je spočítání syndromu:

$$\begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{d-2} \end{pmatrix} = H\mathbf{r}^\top.$$

Jelikož $\mathbf{r} = \mathbf{m} + \mathbf{e}$ a $H\mathbf{m}^\top = 0$, platí rovnost $H\mathbf{r}^\top = H\mathbf{e}^\top$. Pro naši matici H dostáváme

$$S_l = \sum_{j=1}^{n} r_j \alpha^{l(j-1)} = \sum_{j=1}^{n} e_j \alpha^{l(j-1)}, \forall l \in \{0,1,...,d-2\}, \tag{2.5}$$

tedy vyhodnocení $\mathbf{r}$ jakožto polynomu v bodech α^l. Pokud je každé S_l rovno nule, je i $\mathbf{e}$ nulové a přenos proběhl bez chyby. Co dělat, pokud je některé S_i nenulové?

Celý dekódovací algoritmus vychází z několika málo rovnic. Syndrom budeme chápat jako koeficienty *syndromového polynomu* (zkráceně SP) $S(x) = \sum_{l=0}^{d-2} S_l x^l$. S_l můžeme z definice množiny J přepsat také jako $\sum_{j\in J} e_j \alpha^{l(j-1)}$, a celý syndromový polynom lze vyjádřit jako:

$$S(x) = \sum_{l=0}^{d-2} \left(x^l \sum_{j\in J} e_j \alpha^{l(j-1)} \right) = \sum_{j\in J} e_j \sum_{l=0}^{d-2} x^l \alpha^{l(j-1)}. \tag{2.6}$$

Dále zavedeme *lokalizační polynom* (zkráceně LP):

$$\Lambda(x) = \prod_{j\in J} (1 - \alpha^{j-1} x),$$

evaluační polynom (zkráceně EP):

$$\Gamma(x) = \sum_{j\in J} e_j \prod_{m\in J\setminus\{j\}} (1 - \alpha^{m-1} x)$$

(součin přes prázdnou množinu je roven 1) a *koevaluační polynom* (zkráceně KP):

$$\Omega(x) = \sum_{j\in J} e_j \alpha^{(j-1)(d-1)} \prod_{m\in J\setminus\{j\}} (1 - \alpha^{m-1} x).$$

Pro takto definované polynomy platí několik tvrzení. Za prvé, že

$$\Lambda(\alpha^{-\kappa+1}) = 0 \Leftrightarrow \kappa \in J.$$

Kořeny lokalizačního polynomu nám tedy určí pozice chyb. Naopak pro $\kappa \in J$ je

$$\Gamma(\alpha^{-\kappa+1}) = e_\kappa \prod_{m\in J\setminus\{\kappa\}} (1 - \alpha^{m-1}\alpha^{-\kappa+1}) \neq 0,$$

a tedy

$$\gcd(\Lambda(x),\Gamma(x)) = 1. \tag{2.7}$$

Za druhé zjevně platí:

$$\deg \Gamma < \deg \Lambda = |J| \leq \frac{1}{2}(d-1) \tag{2.8}$$

a třetí vztah si napíšeme jako lemma.

Lemma 2.5. *Pro SP, LP, EP a KP platí, že*

$$\Omega(x)x^{d-1} + \Lambda(x)S(x) = \Gamma(x).$$

Důkaz. Rozepíšeme-li si polynomy, získáme rovnost

$$x^{d-1}\sum_{j\in J} e_j\alpha^{(j-1)(d-1)} \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x) + \prod_{j\in J}(1-\alpha^{j-1}x)\sum_{j\in J} e_j \sum_{l=0}^{d-2} x^l\alpha^{l(j-1)} = \\ = \sum_{j\in J} e_j \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x) \tag{2.9}$$

Vnitřní suma ve druhém sčítanci je součet prvních $d-1$ členů geometrické řady, tedy v $\mathbb{F}_q(x)$ platí:

$$\sum_{l=0}^{d-2} x^l\alpha^{l(j-1)} = \sum_{l=0}^{d-2} (x\alpha^{(j-1)})^l = \frac{1-\alpha^{(j-1)(d-1)}x^{d-1}}{1-\alpha^{j-1}x}.$$

Celý druhý sčítanec pak můžeme přepsat následovně:

$$\sum_{j\in J} e_j \sum_{l=0}^{d-2} x^l\alpha^{l(j-1)} = \sum_{j\in J} \frac{e_j}{(1-\alpha^{j-1}x)} - \sum_{j\in J} \frac{e_j\alpha^{(j-1)(d-1)}x^{d-1}}{(1-\alpha^{j-1}x)}.$$

Vynásobením $\prod_{j\in J}(1-\alpha^{j-1}x)$ dostaneme

$$\sum_{j\in J} e_j \prod_{m\in J\setminus j} (1-\alpha^{m-1}x) - \sum_{j\in J} e_j\alpha^{(j-1)(d-1)}x^{d-1} \prod_{m\in J\setminus j} (1-\alpha^{m-1}x).$$

Nyní tento výraz dosadíme do původní rovnice (2.9):

$$x^{d-1}\sum_{j\in J} e_j\alpha^{(j-1)(d-1)} \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x) + \sum_{j\in J} e_j \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x) - \\ - \sum_{j\in J} e_j\alpha^{(j-1)(d-1)}x^{d-1} \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x) = \sum_{j\in J} e_j \prod_{m\in J\setminus\{j\}} (1-\alpha^{m-1}x)$$

a již je zjevné, že rovnost platí.

□

Z lemmatu 2.5 triviálně vyplývá, že

$$\Lambda(x)S(x) = \Gamma(x) \pmod{x^{d-1}}. \tag{2.10}$$

Rovnice (2.10) je označována jako *klíčová rovnice*, přičemž se u ní předpokládá, že jsou splněny podmínky (2.7) a (2.8). Díky podmínkám existuje jednoznačné řešení této rovnice. Pro určení množiny J pak už jen stačí najít kořeny polynomu $\Lambda(x)$. Určíme-li pozice chyb, je zjištění chybového slova už jen otázka vyřešení soustavy již lineárních rovnic (2.6).

Na vyřešení klíčové rovnice je známo několik algoritmů. My si zde ukážeme postup využívající Euklidův algoritmus, je ale možné použít i Berlekamp-Masseyho algoritmus nebo Gaussovu eliminaci. Ta, ač je matematicky nejjednodušší, má kubickou složitost, na rozdíl od zbylých dvou algoritmů, jejichž složitost je kvadratická.

2.3.1 Použití Euklidova algoritmu

Input: $a(x)$, $b(x)$
Output: $\gcd(a(x),b(x))$
$r_{-1}(x) \leftarrow a(x)$; $r_0(x) \leftarrow b(x)$;
$s_{-1}(x) \leftarrow 1$; $s_0(x) \leftarrow 0$;
$t_{-1}(x) \leftarrow 0$; $t_0(x) \leftarrow 1$;
$i \leftarrow 1$;
while $r_{i-1} \neq 0$ **do**
    $q_i(x) \leftarrow r_{i-2}(x)$ div $r_{i-1}(x)$;
    $r_i(x) \leftarrow r_{i-2}(x) - q_i(x)r_{i-1}(x)$;
    $s_i(x) \leftarrow s_{i-2}(x) - q_i(x)s_{i-1}(x)$;
    $t_i(x) \leftarrow t_{i-2}(x) - q_i(x)t_{i-1}(x)$;
    i++;
end
$\gcd(a(x),b(x)) \leftarrow r_{i-2}(x)$;

Algoritmus 2.1: Rozšířený Euklidův algoritmus

Pro vyřešení klíčové rovnice se dá využít rozšířený Euklidův algoritmus, u kterého upravíme podmínku pro ukončení výpočtu. Euklidův algoritmus funguje pro polynomy v $\mathbb{T}[x]$, kde $\mathbb{T}$ je libovolné těleso. Budeme tedy tvrzení a lemmata v kapitole 2.3.1 formulovat také obecněji.

Nechť $a(x), b(x) \in \mathbb{T}[x]$, $\deg a(x) > \deg b(x)$,$\mathbb{T}$ libovolné těleso. Euklidův algoritmus pro polynomy $a(x)$, $b(x)$ najde polynomy $u(x)$,$v(x) \in \mathbb{T}[x]$ takové, že platí $u(x)a(x) + v(x)b(x) = \gcd(a(x),b(x))$. Algoritmus je popsán v algoritmu 2.1 a použitého značení se budeme držet i nadále.

Označíme jako ν nejvyšší takový index, že $r_\nu \neq 0$. Toto r_ν je rovno právě $\gcd(a(x),b(x))$.

Lemma 2.6. *Pro Euklidův algoritmus platí:*

1. *Pro* $i = -1,0,...,\nu+1$*:* $s_i(x)a(x) + t_i(x)b(x) = r_i(x)$,
2. *Pro* $i = 0,...,\nu+1$*:* $\deg t_i + \deg r_{i-1} = \deg a$.

Důkaz. Obě části lemmatu dokážeme indukcí. Nejprve 1:

Pro $i = -1$ máme $1a(x) + 0b(x) = a(x)$. Pro $i = 0$ je rovnost tvaru $0a(x) + 1b(x) = b(x)$. Pro tyto dva případy lemma zjevně platí. Nechť tedy $\forall k < i, k \in \mathbb{N}$ platí $s_k(x)a(x) + t_k(x)b(x) = r_k(x)$. Z popisu algoritmu víme, že

$$s_i(x)a(x) + t_i(x)b(x) = (s_{i-2}(x) - q_i(x)s_{i-1}(x))a(x) + (t_{i-2}(x) - q_i(x)t_{i-1})b(x) =$$
$$= (s_{i-2}(x) + t_{i-2}(x)) - q_i(x)(s_{i-1}(x) + t_{i-1}) = r_{i-2}(x) - q_i(x)r_{i-1}(x) = r_i(x).$$

A tedy lemma 2.6 část 1 platí.

Nyní část 2:

Pro $i = 0$, $\deg 1 + \deg a = \deg a$ i pro $i = 1$, $\deg(-(a \text{ div } b)) + \deg b = \deg a$, rovnost zjevně platí. Ať tedy platí $\forall k < i, k \in \mathbb{N}$. Víme, že platí $t_i = t_{i-2} - t_{i-1}q_i$. Tedy

$$t_i - t_{i-2} = -t_{i-1}(r_{i-2} \text{ div } r_{i-1}) \Rightarrow \deg(t_i - t_{i-2}) = \deg t_{i-1} + \deg r_{i-2} - \deg r_{i-1}.$$

Za $\deg t_{i-1} + \deg r_{i-2}$ můžeme z indukčního předpokladu dosadit $\deg a$ a dostáváme $\deg(t_i - t_{i-2}) + \deg r_{i-1} = \deg a$. Jelikože $\deg t_i > \deg t_{i-2}$, platí, že $\deg(t_i - t_{i-2}) = \deg t_i$ a získáváme dokazovanou rovnost $\deg t_i + \deg r_{i-1} = \deg a$. □

Tvrzení 2.7. *Předpokládejme, že máme $t(x)$, $r(x), a(x), b(x)$ nenulové polynomy nad $\mathbb{F}_q$, které splňují podmínky:*

1. $\gcd(t(x),r(x)) = 1$,
2. $\deg t + \deg r < \deg a$,
3. $t(x)b(x) = r(x) \pmod{a(x)}$.

Pak, při zachování značení z Euklidova algoritmu, existují index $h \in \{0,1,...,\nu+1\}$ a konstanta $c \in \mathbb{F}_q$ takové, že

$$t(x) = c \cdot t_h(x) \text{ a } r(x) = c \cdot r_h(x)$$

.

Důkaz. Z jednotlivých kroků Euklidova algoritmu je zřejmé, že stupeň r_i ostře klesá. Zároveň z podmínky 2 a inicializace Euklidova algoritmu víme, že $\deg r < \deg a = \deg r_{-1}$. Z toho vyplývá, že existuje index h, pro který

$$r_h \leq \deg r < \deg r_{h-1}. \tag{2.11}$$

Z lemmatu 2.6 víme, že

$$s_h(x)a(x) + t_h(x)b(x) = r_h(x). \tag{2.12}$$

Z podmínky 3 zároveň vyplývá, že existuje polynom, označme ho $s(x)$, takový, že

$$s(x)a(x) + t(x)b(x) = r(x). \tag{2.13}$$

Odečteme-li od rovnice 2.12 vynásobené polynomem $t(x)$ rovnici 2.13 vynásobenou polynomem $t_h(x)$, získáme následující rovnost:

$$(t(x)s_h(x) - t_h(x)s(x))a(x) = t(x)r_h(x) - t_h(x)r(x). \tag{2.14}$$

Podívejme se nyní na stupeň pravé strany této rovnice. Z nerovnosti (2.11) a podmínky 2 víme, že $\deg t + \deg r_h < \deg a$. Pokud naopak spojíme nerovnost (2.11) a lemma 2.6 část 2, získáme $\deg t_h + \deg r = \deg a - \deg r_{h-1} + \deg r < \deg a$. Polynomy $t(x)r_h(x), t_h(x)r(x)$ mají tedy oba stupeň menší než $a(x)$, zároveň se ale jejich součet rovná násobku $a(x)$, a tedy musí platit $t(x)r_h(x) - t_h(x)r(x) = 0$, neboli

$$t(x)r_h(x) = t_h(x)r(x). \tag{2.15}$$

Z lemmatu 2.6 část 2 dostáváme, že $\deg t_h \geq 0$, a $\deg r \geq 0$ dle předpokladů. Z rovnosti (2.15) a podmínky 1 vyplývá, že $r(x)|r_h(x)$, a jelikož zároveň $\deg r_h \leq \deg r$, musí existovat c takové, že $r(x) = c \cdot r_h(x)$. Dosadíme-li za $r(x)$ do rovnice (2.15) a zkrátíme, získáme i rovnost $t(x) = c \cdot t_h(x)$.

□

Nyní můžeme z tvrzení 2.7 odvodit, jak vyřešit klíčovou rovnici. Připomeňme si, jak vypadá: $\Lambda(x)S(x) = \Gamma(x) \pmod{x^{d-1}}$. Porovnáme-li ji s podmínkou 3 z tvrzení 2.7, je zjevné, že budeme počítat Euklidův algoritmus pro $a(x) = x^{d-1}$ a $b(x) = S(x)$. Polynom $r(x)$ je roven $\Gamma(x)$ a $t(x) = \Lambda(x)$. Oba polynomy splňují i zbylé dvě podmínky tvrzení 2.7 a pomocí algoritmu tedy najdeme $t_h(x)$ a $r_h(x)$. Konstantu c zvolíme tak, aby konstantní člen polynomu $\Lambda(x)$ byl roven jedné.

Jediným problémem nyní je, jak poznáme onen index h. V tvrzení 2.7 je to nejnižší index takový, že $\deg r_h \leq \deg r$. Polynom $r(x) = \Gamma(x)$ je ale právě polynom, který chceme získat a nevíme předem, jaký bude mít stupeň. Ke zjištění indexu nám ale pomůže vlastnost 2.8, která je silnější, než předpoklady tvrzení 2.7.

Tvrzení 2.8. *Mějme polynomy $t(x)$,$r(x)$ jako v tvrzení 2.7, pro něž navíc platí, že*

$$\deg t \leq \frac{1}{2}\deg a \;\; a \;\; \deg r < \frac{1}{2}\deg a.$$

Potom index h je právě ten index, pro který platí:

$$\deg r_h < \frac{1}{2}\deg a \leq \deg r_{h-1}.$$

Důkaz. Pokud bychom vzali nižší index i, měl by polynom r_i vysoký stupeň. Naopak pokud bychom vzali index $i > h$, platil by vztah $\deg r_i < \frac{1}{2}\deg a$. Ale z lemmatu 2.6 víme, že $\deg t_{i+1} + \deg r_i = \deg a$. Vzhledem k tomu, že $i \geq h+1$, platí $\deg t_i \geq \deg t_{h+1} = \deg a - \deg r_h > \frac{1}{2}\deg a$.

□

Nyní tedy umíme pomocí Euklidova algoritmu nalézt lokalizační polynom $\Lambda(x)$. Dále bychom chtěli najít jeho kořeny, abychom zjistili, na kterých pozicích jsou chyby.

2.3.2 Chienovo vyhledávání

Kořeny lokalizačního polynomu se hledají jednoduše tak, že se zkouší dosadit všechny nenulové prvky tělesa. Algoritmus pojmenovaný *Chienovo vyhledávání* popisuje, jak by se mělo dosazování prvků implementovat, aby bylo výpočetně co nejrychlejší.

Opírá se o dvě jednoduchá pozorování:

- každý nenulový prvek tělesa lze zapsat jako α^i, kde α je primitivní prvek,
- při dosazování platí následující vztahy:

$$\begin{aligned}
\Lambda(\alpha^i) &= \lambda_0 + \lambda_1(\alpha^i) + \lambda_2(\alpha^i)^2 + ... + \lambda_t(\alpha^i)^t \\
\Lambda(\alpha^{i+1}) &= \lambda_0 + \lambda_1(\alpha^{i+1}) + \lambda_2(\alpha^{i+1})^2 + ... + \lambda_t(\alpha^{i+1})^t = \\
&= \lambda_0 + \lambda_1(\alpha^i)\alpha + \lambda_2(\alpha^i)^2\alpha^2 + ... + \lambda_t(\alpha^i)^t\alpha^t
\end{aligned}$$

Označíme-li si $\lambda_j(\alpha^i)^j$ jako $\gamma_{j,i}$, máme $\Lambda(\alpha^i) = \sum_{j=0}^{t} \gamma_{j,i}$ a $\gamma_{j,i+1} = \gamma_{j,i}\alpha^j$. Při počítání tedy začneme s $i = 0$ a $\gamma_{j,0} = \lambda_j$, $\gamma_{j,i+1}$ počítáme podle zmíněného vztahu a kontrolujeme, zda $\sum_{j=0}^{t} \gamma_{j,i} = 0$.

Poté, co projdeme všechny nenulové prvky tělesa ($i \in \{0,...,q-1\}$) známe pozice chyb. Stále ale nevíme, jakou hodnotu chyby měly a nemůžeme je tedy opravit.

2.3.3 Forneyho algoritmus

Forneyho algoritmus se používá k určení hodnot chyb. Pro výpočet budeme potřebovat derivaci polynomu $\Lambda(x)$. Ze vzorečku pro derivaci součinu dostáváme, že

$$\Lambda'(x) = \sum_{j \in J} -\alpha^{j-1} \prod_{m \in J \setminus \{j\}} (1 - \alpha^{m-1}x). \tag{2.16}$$

Pro každou pozici chyby $\kappa \in J$ pak platí

$$\Lambda'(\alpha^{-\kappa+1}) = -\alpha^{\kappa-1} \prod_{m \in J \setminus \{\kappa\}} (1 - \alpha^{m-1}\alpha^{-\kappa+1}).$$

Pokud dosadíme $\alpha^{-\kappa+1}$ do evaluačního polynomu, dostáváme

$$\Gamma(\alpha^{-\kappa+1}) = e_\kappa \prod_{m \in J \setminus \{\kappa\}} (1 - \alpha^{m-1}\alpha^{-\kappa+1}).$$

Jak je na první pohled vidět, součiny v obou rovnicích jsou stejné. $\Lambda(x)$, $\Gamma(x)$ i množinu J již známe, můžeme tedy vydělit:

$$\frac{\Gamma(\alpha^{-\kappa+1})}{\Lambda'(\alpha^{-\kappa+1})} = \frac{e_\kappa \prod_{m \in J \setminus \{\kappa\}} (1 - \alpha^{m-1}\alpha^{-\kappa+1})}{-\alpha^{\kappa-1} \prod_{m \in J \setminus \{\kappa\}} (1 - \alpha^{m-1}\alpha^{-\kappa+1})}.$$

A po úpravách máme vzorec pro e_κ:

$$e_\kappa = -\alpha^{\kappa-1} \frac{\Gamma(\alpha^{-\kappa+1})}{\Lambda'(\alpha^{-\kappa+1})}.$$

2.3.4 Kompletní algoritmus

Celý proces dekódování je stručně shrnut v algoritmu 2.2.

Input: obdržené slovo $\mathbf{r}= (r_1, r_2,..., r_n)$
Output: původní zpráva $\mathbf{m}= (m_1, m_2,..., m_n)$
for $i = 0$ **to** $d - 1$ **do** *//spočti syndromy*
$S_i = \sum_{j=0}^{n} y_j \alpha^{i(j-1)}$
end
if $S_i = 0 \; \forall i$ **then**
vrať $\mathbf{m}= \mathbf{r}$
end
vytvoř syndromový polynom: $S(x) = \sum_{l=0}^{d-2} S_l x^l$;
prováděj Euklidův algoritmus pro $a(x) = x^{d-1}$ a $b(x) = S(x)$, skonči, jakmile $\deg r_h < \frac{1}{2}(d-1)$;
$\Lambda(x) = t_h(x)$; $\Gamma(x) = r_h(x)$;
for $i = 0$ **to** 256 **do** *//spočti pozice chyb*
if $\Lambda(\alpha^i) = 0$ **then**
$J = J \cup \{-i + 1\}$
end
end
foreach $\kappa \in J$ **do** *//spočti hodnoty chyb*
$e_\kappa = -\alpha^{\kappa-1} \frac{\Gamma(\alpha^{-\kappa+1})}{\Lambda'(\alpha^{-\kappa+1})}$
end
vrať $\mathbf{m} = \mathbf{r} + \mathbf{e}$;

Algoritmus 2.2: Algoritmus opravy chyb přijatého slova

Kapitola 3

Příklad

Ukážeme nyní všechny teoretické poznatky na konkrétním příkladu a popíšeme některé části implementace ZXing, dostupné na [6], která byla použita k vytvoření programu pro čtení QR kódů. Na obrázku 3.1 je QR kód, který budeme dekódovat. Vygenerován byl na webové adrese http://zxing.appspot.com/generator a do jeho středu bylo vloženo logo MFF UK.

Obrázek 3.1: QR kód, který bude dekódován

3.1 Detekce

Prvním krokem je detekovat QR kód. Vstupem programu je 24-bitová bitmapa, tedy matice 24-bitových hodnot, jejíž každé pole nese informaci o barvě jednoho pixelu. Typicky je to fotka QR kódu z mobilního telefonu, která má rozměry v rámci stovek až tisíců pixelů, většinou ale do tří tisíc. Matice je následně převedena na matici bytů, tedy barva každého pixelu již není reprezentována dvaceti čtyřmi bity, ale pouze osmi bity. Tato matice je předána knihovně ZXing. Ta vytvoří histogram obrázku a pomocí něho rozdělí barvy pouze na černou a bílou, vytvoří tedy z matice bytů matici bitů. Pokud bychom celý proces viděli, změnil by se původní barevný obrázek na obrázek černobílý.

V tomto černobílém obrázku se pak hledá vzor, který by odpovídal QR kódu. Algoritmus v knihovně ZXing vypadá ve stručnosti takto:

1. Hledají se hlavní zaměřovací značky.

2. Pokud jsou nalezeny, je podle jejich souřadnic určeno otočení QR kódu. Z každé značky je z jejich velikosti (která je 7 modulů) určena velikost jednoho modulu, tyto tři velikosti jsou pak zprůměrovány.

3. Z velikosti modulu a souřadnic značek je spočítána dimenze (celkový rozměr) kódu.

4. Z dimenze je pak spočítána provizorní verze. Ta je nyní potřeba, aby algoritmus věděl, zda hledat i vedlejší zaměřovací značky.

5. Pokud mají být v QR kódu i vedlejší zaměřovací značky, spočítají se jejich přibližné souřadnice a značky se hledají v okolí vypočtených souřadnic.

6. Provede se transformace matice reprezentující obrázek, pokud je potřeba. Odstraní se tedy například zkosení.

Výstupem algoritmu je transformovaná (zkosená nebo pootočená) matice bitů, souřadnice, na kterých se v matici nacházejí hlavní a případné vedlejší zaměřovací značky, a velikost modulu.

3.2 Dekódování

Prvním krokem dekódování je vytvoření bitové matice, ve které každý bit reprezentuje jeden modul. Ta se vytvoří z bitové matice z algoritmu detekování tak, že se „vyřízne“ čtverec, který je určen souřadnicemi hlavních zaměřovacích značek. Pro náš QR kód je výsledek na obrázku 3.2.

Obrázek 3.2: QR kód, jak jej „vidí“ program

3.2.1 Verze

Při určování verze se spočítá provizorní verze (dimenze $-$ 17) : 4 a pokud je menší, než 7, funkce vrátí hodnotu provizorní verze. To je i náš případ, jelikož dimenze kódu je 29 a tedy provizorní verze je $(29-17) : 4 = 3$. Pokud je provizorní verze 7 a víc, přečtou se a dekódují informace o verzi kódované Golayovým kódem,

které jsou uložené vedle pravé horní značky. V případě, že nastane chyba, zkusí se přečíst informace u levé dolní značky a pokud ani to se nepovede, QR kód není možno dekódovat.

Provizorní verze se zde počítá znovu, jelikož výpočet není časově náročný a je jednodušší ji znovu spočítat, než si ji od výpočtu v detekování kódu nějakým způsobem pamatovat.

3.2.2 Formát

Informace o formátu jsou uloženy okolo separátorů, jak je ukázáno na obrázku 1.2. V našem případě jsou tedy tvaru:

$$000011101100010.$$

Chráněny jsou [15,5] BCH kódem, jak bylo zmíněno v kapitole 1.3. Kromě toho bylo také v kapitole 1.3 řečeno, že ke každému kódovému slovu je přičtena maska.

Při dekódování se ale maska neodečítá. Dekódování informací o formátu probíhá jednoduše pomocí hledání nejbližšího kódového slova. S maskou formátu je to vyřešeno tak, že kódová slova, mezi kterými se hledá nejbližší, nejsou skutečná kódová slova [15,5] BCH kódu. Označíme-li $C = [15,5]$ BCH, hledáme při dekódování nejbližší slovo v množině $C + maska$.

Kódových slov je v tomto případě tak málo, že je množina $C + maska$ vypsána přímo ve zdrojovém kódu a prohledávání se vyplatí. 000011101100010 se přímo shoduje s kódovým slovem, kterému odpovídá zpráva 10100.

Jak bylo řečeno v kapitole 1, první dva bity informace určují úroveň zabezpečení, přičemž 10 znamená zabezpečení H, a zbylé tři, tedy 100, udávají kód masky.

3.2.3 Maska

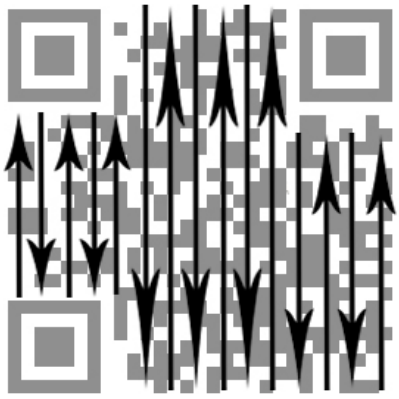
Když už známe kód masky, můžeme ji odečíst. Maska s kódem 100 je ukázána na obrázku 1.3 uprostřed. Matematicky definována je tak, že černé moduly jsou právě ty, pro které $((i \text{ div } 2)+(j \text{ div } 3)) = 0 \pmod 2$, kde i značí řádek, j sloupec a $(i,j) = (0,0)$ je levý horný modul QR kódu. Projdeme tedy celou matici bitů a změníme hodnotu bitu na takových pozicích (i,j), pro které je splněna podmínka a zároveň to nejsou bity z funkčních vzorů. Výsledek je vidět na obrázku 3.3.

3.2.4 Načtení koeficientů

Nyní již můžeme přečíst koeficienty kódových slov a pracovat jen s nimi, čímž se dostáváme k matematické části celého dekódování. Koeficienty zabírají každý osm modulů a jsou uloženy ve sloupcích v útvarech širokých dva moduly. Číst se začíná v pravém dolním rohu QR kódu, pokračuje se prvním sloupcem o šířce dva moduly směrem nahoru až k informacím o formátu, dále vedlejším sloupečkem směrem dolů a tak dále pořád dokola, jak je vyobrazeno na obrázku 3.4.

Podíváme-li se na obdélník 2×4 moduly v pravém dolním rohu QR kódu na obrázku 3.3 a černé moduly přepíšeme jako jedničky, bílé jako nuly, dostaneme 01000001 představující prvek tělesa $\alpha^6 + 1$. Obdélník nad ním přepíšeme jako $01101011 = \alpha + \alpha^2 + \alpha^4 + \alpha^6 + \alpha^7$. A tak bychom mohli pokračovat dál, celkem bychom načetli 70 koeficientů z $\mathbb{F}_{2^8}$.

Obrázek 3.3: QR kód po odečtení masky s kódem 100



Obrázek 3.4: Čtení koeficientů kódových slov

3.2.5 Bloky

V kapitole 1 bylo řečeno, že největší kód pojme až 2953 bytů. Zároveň jsme se ale v kapitole 2.2 dozvěděli, že nejdelší z použitých kódů má délku 153. Je tedy zjevné, že celý QR kód není tvořen jedním kódovým slovem, ale mnoha. Kolika kódovými slovy je tvořen a z jakých kódů jsou jednotlivá slova, se liší pro každou verzi i úroveň zabezpečení. Informace jsou uloženy přímo ve zdrojovém kódů. V našem případě, tedy pro kód verze 3 při úrovni zabezpečení H, QR kód obsahuje 2 bloky, oba představující kódová slova z [35,13,23] kódu.

Jak jsou jednotlivé koeficienty z bloků složeny do výsledné zprávy, která představuje QR kód, je vidět na obrázku 3.5. Posloupnost sedmdesáti prvků, které

	Data				Symboly opravy chyb			
1. blok	D_1	D_2	.....	D_{13}	E_1	E_2	.....	E_{22}
2. blok	D_{14}	D_{15}	.....	D_{26}	E_{23}	E_{24}	.....	E_{44}

Obrázek 3.5: Jak se skládají symboly do výsledné zprávy

máme, je tedy tvaru D_1, D_{14}, D_2,..., D_{26}, E_1, E_{23}, E_2,..., E_{22}, E_{44}. Rozdělíme ji do

dvou bloků, D_1, D_2,...,$D_{13}, E_1, ..., E_{22}$ a D_{14}, D_{15},..., D_{26}, E_{23},..., E_{44}, a každý budeme opravovat zvlášť. Detailně ukážeme opravu prvního bloku, který vypadá takto:

$\{\alpha^6+\alpha^4+\alpha^2+\alpha,\alpha^7+\alpha^5,\alpha^4+\alpha^3+1,\alpha^5+\alpha^4+1,$
$\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2,\alpha^5+\alpha^4+\alpha^3,\alpha^5+\alpha^3+1,$
$\alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^3+1,\alpha^4+\alpha,\alpha^6+\alpha+1,\alpha^5+\alpha^3+\alpha^2+1,$
$\alpha^5+\alpha^4+\alpha^3+1,\alpha^7+\alpha^5+\alpha,\alpha^7+\alpha^6+\alpha^4+\alpha^3,$
$\alpha^6+\alpha^5+\alpha^3+\alpha,\alpha^6+\alpha^4+\alpha^3+\alpha^2+\alpha,\alpha^3,\alpha^6+\alpha^5+\alpha^3+\alpha^2,$
$\alpha^7+\alpha^5+\alpha^4+\alpha^2,\alpha^7+\alpha^6+\alpha^3,\alpha^5+\alpha^3+\alpha+1,$
$\alpha^5+\alpha^4+\alpha^3+1,\alpha^7+\alpha^6+\alpha^5+\alpha^2+\alpha,\alpha^6+\alpha^5+\alpha^4+\alpha,$
$\alpha^7+\alpha^4+\alpha^3+\alpha^2+\alpha+1,\alpha^6+\alpha^5+\alpha^4+\alpha^2+\alpha+1,$
$\alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^2+\alpha+1,\alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha,\alpha^7+\alpha^5+\alpha,$
$\alpha+1,\alpha^6+\alpha^2+\alpha+1,\alpha^6+\alpha^2+\alpha+1,\alpha^7+\alpha^2+\alpha+1,\alpha^6+\alpha^5+\alpha^2+\alpha,\alpha^6+1\}$

3.2.6 Syndromový polynom

Syndrom získáme vynásobením kódového slova, tedy jednoho bloku, s prověrkovou maticí. V tomto případě se ale využívá vzoreček 2.5. Dosadíme tedy do polynomu, jehož koeficienty jsou prvky bloku, postupně prvky $1, \alpha, \alpha^2$,...,α^{21}.

Při dalších výpočtech je potřeba dávat pozor na jednu věc: Při dodržení značení vzorečků z kapitoly 2.3 platí $(r_1,...,r_{35}) = (E_{22},...,E_1,D_{26},...,D_1)$. Z toho důvodu nám při dekódování vyjdou pozice chyb „opačně“. Pokud bude řečeno, že je chyba na třetí pozici, myslí se třetí pozice od konce chybového slova.

Syndromový polynom $S(x)$ vypadá následovně:

$$\begin{aligned}&1+\alpha+\alpha^4+\alpha^5+(1+\alpha^2+\alpha^4+\alpha^6)x+(\alpha+\alpha^4)x^2+\\&+x^3+(1+\alpha+\alpha^2+\alpha^5)x^4+(1+\alpha^5+\alpha^6+\alpha^7)x^5+\\&+(1+\alpha^2+\alpha^6+\alpha^7)x^{20}+(1+\alpha^2+\alpha^3+\alpha^4+\alpha^6+\alpha^7)x^{21}\end{aligned}$$

3.2.7 Euklidův algoritmus

Je zjevné, že syndrom je nenulový, což znamená, že došlo k chybám. Spustíme Euklidův algoritmus na $a(x) = x^{22}$ a $b(x) = S(x)$. Vzhledem k tomu, že $r_{-1} = a$, stupeň r v každém kroku ostře klesá a algoritmus bude ukončen, jakmile $\deg r_h < \frac{1}{2}22$, provede se maximálně 12 průchodů algoritmem. Výsledkem je lokalizační polynom

$$\begin{aligned}&1+(\alpha+\alpha^2+\alpha^3+\alpha^4+\alpha^6)x+(1+\alpha+\alpha^4+\alpha^7)x^2+(1+\alpha+\alpha^3+\alpha^6+\alpha^7)x^3+\\&+(1+\alpha+\alpha^2+\alpha^4+\alpha^5+\alpha^6)x^4+\alpha^2x^5+(1+\alpha^2+\alpha^3+\alpha^4+\alpha^6)x^6+\\&+(\alpha^4+\alpha^5+\alpha^6+\alpha^7)x^7+(1+\alpha+\alpha^2+\alpha^5+\alpha^6+\alpha^7)x^8+(a+\alpha^2+\alpha^3+\alpha^4+\alpha^7)x^9\end{aligned}$$

a evaluační polynom

$$\begin{aligned}&1+\alpha+\alpha^4+\alpha^5+(1+\alpha^4+\alpha^5)x+(\alpha^2+\alpha^6)x^2+(1+\alpha+\alpha^4)x^3+\\&+(1+\alpha+\alpha^2+\alpha^5)x^4+(\alpha^3+\alpha^7)x^5+(1+\alpha^2)x^6+(1+a+\alpha^2+\alpha^3+\alpha^6)x^7+\\&+(1+\alpha+\alpha^2+\alpha^3+\alpha^4+\alpha^5)x^8.\end{aligned}$$

Vzpomeňme si, že lokalizační polynom byl volen tak, že jeho kořeny přesně určují pozice chyb. Vzhledem k tomu, že má stupeň 9, bylo v prvním bloku poškozeno 9 koeficientů.

3.2.8 Chienovo vyhledávání

Nyní je třeba kořeny lokalizačního polynomu najít. Jak bylo popsáno v kapitole 2.3, hledá se postupným dosazováním prvků tělesa a vylepšení jsou pouze technická. Dosazení několika prvků na ukázku je zde:

$$\begin{aligned} &\vdots \\ LP(\alpha^{249}) &= 1 + \alpha^2 + \alpha^4 + \alpha^5 + \alpha^7 \\ LP(\alpha^{248}) &= 0 \\ LP(\alpha^{247}) &= 1 + a + \alpha^2 + \alpha^3 + \alpha^5 + \alpha^7 \\ &\vdots \end{aligned}$$

Do polynomu jsou v ukázce dosazovány mocniny od nejvyšší z prostého důvodu. Víme, že pokud jsme počítali správně, budou kořeny prvky α^κ pro taková κ, že chybový vektor je nenulový na pozici $-\kappa + 1$. Vzhledem k tomu, že náš chybový vektor má délku jen 35, musí být všechna $-\kappa + 1$ v rozmezí 1,...,35 a stačí vyzkoušet pouze 35 prvků tělesa.

V implementaci není paradoxně použito žádné vylepšení. Tam je dosazování řešeno pomocí klasického for cyklu pro i od 1 do 255; do lokalizačního polynomu se vždy dosazuje prvek tělesa, jehož koeficienty odpovídají číslu i v binárním zápisu.

Z výpočtů vidíme, že jedním z kořenů je α^{248} z čehož vyplývá, že na $-248+1$. pozici je chyba, přičemž $-247 = 255 - 247 = 8$. Dalšími kořeny lokaliazčního polynomu jsou $\alpha^{245}, \alpha^{244}, \alpha^{241}, \alpha^{242}, \alpha^{238}, \alpha^{235}, \alpha^{234}, \alpha^{231}$ a tedy chybový vektor má nenulové koeficienty na osmé, jedenácté, dvanácté, čtrnácté, patnácté, osmnácté, dvacáté první, dvacáté druhé a dvacáté páté pozici.

3.2.9 Forneyho algoritmus

Prostým dosazováním do vzorečku dostaneme hodnoty chyb. Jediným zádrhelem by mohlo být počítání derivace lokalizačního polynomu. Jelikož počítáme nad tělesem charakteristiky dva, je formální derivace velmi jednoduchá. Derivace polynomu $\sum_{i=0}^{k} a_i x^i$ je tvaru $\sum_{i=0}^{\lfloor k/2 \rfloor} a_{2i+1} x^{2i}$. V implementaci se ale využívá vzorečku 2.16, do kterého se přímo dosazuje, takže $\Lambda'(x)$ se formálně jako polynom, do kterého bychom dosazovali, v programu nevyskytuje.

Vypočteme zde na ukázku několik chyb:

$$\begin{aligned} e_8 = \alpha^7 \frac{\Gamma(\alpha^{-7})}{\Lambda'(\alpha^{-7})} &= \alpha^7 \alpha^{217} \alpha^{-25} = \alpha^{199} = \alpha + \alpha^2 + \alpha^3 \\ e_{11} = \alpha^{10} \frac{\Gamma(\alpha^{-10})}{\Lambda'(\alpha^{-10})} &= \alpha^{10} \alpha^{136} \alpha^{-160} = \alpha^{241} = \alpha^3 + \alpha^4 + \alpha^6 \\ &\vdots \end{aligned}$$

Hodnoty všech chyb jsou popořadě $\{\alpha^3+\alpha^2+a,\alpha^6+\alpha^4+\alpha^3,1,\alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^3,\alpha^6+\alpha^5+\alpha^4+\alpha^3,\alpha^6+\alpha^5,\alpha^5+\alpha^4,\alpha^6+\alpha^4+\alpha^3+\alpha^2,\alpha^7+\alpha^6+\alpha^5+\alpha^3\}$

3.2.10 Oprava

Binárně sečteme přijatý vektor s chybovým vektorem a získáme kódové slovo. Vzhledem k tomu, že již máme opravené slovo a dále nás zajímají jen skutečná data, stačí sečíst jen prvních 13 koeficientů každého vektoru.

$$\{\alpha^{219},\alpha^{55},\alpha^{193},\alpha^{181},\alpha^{115},\alpha^{201},\alpha^{147},\alpha^{214},\alpha^{224},\alpha^{98},\alpha^{18},\alpha^{154},\alpha^{209}\}$$

$$\{0,0,0,0,0,0,0,0,0,0,\alpha^{11},0,0\}$$

Po sečtení máme

$$\{\alpha^{219},\alpha^{55},\alpha^{193},\alpha^{181},\alpha^{115},\alpha^{201},\alpha^{147},\alpha^{214},\alpha^{224},\alpha^{98},\alpha^{123},\alpha^{154},\alpha^{209}\}$$

Prvky tělesa převedeme do bitové reprezentace a dostaneme následující:

01000001 01100110 10000111 01000111 01000111 00000011 10100010 11110010
11110111 01110111 01110111 01110010 11100110

3.3 Převod na text

Poslední, co zbývá, je z bitů vytvořit čitelnou zprávu. Podstatné pro převod jsou první čtyři bity, ve kterých je uložen formát dat QR kódu. 0100 na začátku naší zprávy značí bytový formát dat. V bytovém módu verze 3 pak dalších osm bitů, tedy 00010110, nese informaci o počtu zakódovaných znaků, v tomto případě 22. Tím pádem z výpočtu a kapitoly 1.2 víme, že následujících sto sedmdesát šest bitů (připojíme-li i druhý blok) je zpráva v UTF-8 kódování. Rozdělíme si tedy zprávu po osmi bitech počínaje třináctým bitem a můžeme převést na znaky:

$$\begin{aligned}
01101000 &\to \text{h}\\
01110100 &\to \text{t}\\
01110100 &\to \text{t}\\
01110000 &\to \text{p}\\
00111010 &\to :\\
00101111 &\to /\\
00101111 &\to /\\
01110111 &\to \text{w}\\
&\vdots
\end{aligned}$$

Celá zpráva zní *http://www.mff.cuni.cz*

3.4 Implementace

Implementace jednotlivých algoritmů, např. Euklidova, Chienova atd., se příliš neliší od popisů uvedených zde. Většinou jsou jen přidané podmínky pro kontrolu předávaných parametrů, aby nedošlo k pádu programu. Na čem vše stojí, je implementace aritmetiky v $\mathbb{F}_{2^8}$ a $\mathbb{F}_{2^8}[x]$.

3.4.1 Počítání v $\mathbb{F}_{2^8}$

Vše, co souvisí s tělesem, je naprogramováno v souboru GenericGF.cpp. Prvním krokem v programu je vůbec vytvořit těleso. O to se stará funkce initialize popsaná a okomentovaná v algoritmu 3.1, jejíž parametry jsou velikost tělesa a primitivní polynom. V paměti vytvoří dvě pole: expTable[i] = α^i a logTable[α^i]=i

Input: size, primitive
Output: expTable, logTable
int x = 1; **for** *(int i = 0; i < size; i++)* **do**
    // Do i-tého políčka pole vlož aktuální x, vzhledem k tomu, že pole jsou indexována od nuly, inicializujeme x jedničkou, platí $\alpha^0 = 1 = expTable[0]$
    expTable[i] = x;
    // Bitový posun x o jedna doleva odpovídá násobení číslem 2, jak už jsme si řekli, 2 reprezentuje α
    x «= 1 ;
    // Pokud je x větší, než velikost tělesa, tedy pokud je stupeň x jakožto polynomu 8 nebo vyšší, musíme modulit primitivní polynomem. Vzhledem k tomu, že x zvyšujeme postupně, bude stupeň x vždy právě 8 a stačí pouze odečíst primitivní polynom
    if *(x >= size)* **then**
        x = x XOR primitive;
        // bitový and v našem případě x nezmění, kód je ale psán, aby vytvořil i jiná tělesa
        x = x AND size-1;
    end
end
for *(int i = 0; i < size-1; i++)* **do**
    logTable[expTable[i]] = i;
end

Algoritmus 3.1: Inicializace tělesa

a dále už je všechno počítání jednoduché. Sčítání je bitový xor, při násobení pouze sčítáme mocniny a při dělení mocniny odečítáme, obojí modulo 255.

3.4.2 Počítání v $\mathbb{F}_{2^8}[x]$

Počítání se samotnými polynomy už je složitější. Polynom je reprezentován strukturou GenericGFPoly obsahující informaci o tělese, ze kterého jsou koeficienty, a polem koeficientů, přičemž první prvek, tedy prvek na nulté pozici, je vedoucím koeficientem a musí být nenulový.

K dekódování QR kódů potřebujeme funkce pro dosazení do polynomu, pro součet a pro násobek dvou polynomů. Násobení je, pravděpodobně kvůli urychlení algoritmu, rozděleno na tři případy, na násobení polynomu skalárem, monomem a nakonec obecným polynomem. Asi nejjednodušší z nich je násobení skalárem, kdy stačí každý prvek v poli koeficientů vynásobit příslušným prvkem. Násobení

monomem znamená zvýšit mocninu původního polynomu o mocninu monomu a pak opět každý prvek v poli koeficientů vynásobit koeficientem monomu.

Pro Euklidův algoritmus bychom měli potřebovat i funkci pro dělení dvou polynomů. Ta je ve zdrojovém kódu opravdu napsána, ale není využita. Dělení polynomů je totiž řešeno přímo v Euklidově algoritmu.

Podrobněji rozebereme obecné násobení dvou polynomů, v němž se nepoužívá typický vzoreček $\sum_{i=0}^{k} a_i x^i \cdot \sum_{j=0}^{l} b_j x^j = \sum_{i=0}^{k+l} x^i \sum_{j=0}^{i} a_j b_{i-j}$, ale ekvivalentní rovnost 3.1:

$$\sum_{i=0}^{k} a_i x^i \cdot \sum_{j=0}^{l} b_j x^j = \sum_{i=0}^{k} \sum_{j=0}^{l} x^{i+j} a_i b_j \tag{3.1}$$

```
Input: polynomy: this, other
Output: this · other
if !(field.object == other->field.object) then
    throw IllegalArgumentException("GenericGFPolys do not have same
    GenericGF field");
end
// Pokud je jeden z polynomů nulový, vrátím nulu, není potřeba
    nic počítat
if (isZero() || other->isZero()) then
    return field->getZero()
end
ArrayRef<int> aCoefficients = coefficients;
int aLength = aCoefficients.size();
ArrayRef<int> bCoefficients = other->getCoefficients();
int bLength = bCoefficients.size();
// Stupeň násobku bude roven součtu stupňů obou polynomů,
    vytvoříme si tedy pole potřebné délky
ArrayRef<int> product(new Array<int>(aLength + bLength - 1));
// Násobek pak počítáme pomocí dvou for cyklů, podle vzorečku
    3.1
for (int i = 0; i < aLength; i++) do
    int aCoeff = aCoefficients[i];
    for (int j = 0; j < bLength; j++) do
        product[i+j] = GenericGF::addOrSubtract(product[i+j],
        field->multiply(aCoeff, bCoefficients[j]));
    end
end
return Ref<GenericGFPoly>(new GenericGFPoly(field, product));
```

Algoritmus 3.2: Násobení dvou polynomů

Paradoxně nejdelším algoritmem je algoritmus pro sčítání dvou polynomů, ačkoli je v principu nejjednodušší.

Ukážeme si ještě implementaci dosazování do polynomu, která není implementována přes základní vztah $f(a) = \sum_{i=1}^{k} f_i a^i$, jak bychom mohli očekávat, ale

pomocí rozkladu:

$$f_n a^n + f_{n-1}a^{n-1} + f_{n-2}a^{n-2} + ... + f_0 = (...((f_n a + f_{n-1})a + f_{n-2})a + f_{n-3})...)a + f_0 \tag{3.2}$$

Zatímco při počítání podle základního vztahu bychom v každém kroku dvakrát násobili (zvednutí mocniny a, krát f_i), takto násobíme pouze jednou a následně sčítáme. Výhoda je hlavně časová, sčítání v tělese odpovídá bitovému xoru, který je velmi rychlý, zatímco násobení prvků a,b obnáší přístup do pole expTable na pozici (logTable[a] + logTable[b]) mod 255.

```
Input: a
Output: f(a)
if a == 0 then // „Zkratka", f(0) = x^0

    return getCoefficient(0);
end
int size = coefficients.size();
if a == 1 then // „Zkratka", f(1) = součet koeficientů

    int result = 0;
    for (int i = 0; i < size; i++) do
        result = GenericGF::addOrSubtract(result, coefficients[i]);
    end
    return result;
end
int result = coefficients[0];
// Postupné počítání výsledku pomocí vzorečku 3.2
for (int i = 1; i < size; i++) do
    result = GenericGF::addOrSubtract(field->multiply(a, result),
    coefficients[i]);
end
return result;
```

Algoritmus 3.3: Vyhodnocování polynomu v bodě a

Kapitola 4

Testy

Testovat jsem původně chtěla jen časové rozdíly mezi dekódováním různých verzí QR kódu a mezi poškozenými a nepoškozenými QR kódy. Během studování kódu jsem ale zjistila, jak bylo již napsáno v kapitole 3.2.8, že Chienovo vyhledávání je implementováno neefektivně. Přidala jsem proto i porovnání různých implementací právě Chienova algoritmu.

4.1 O ZXing

ZXing je open-source knihovna pro dekódování 1D i 2D čárových kódů; převzata z ní však byla pouze část pro dekódování QR kódů. Primárně je psána v Javě, přepsána je však i do mnoha jiných programovacích jazyků, na ty se ale při postupném vývoji spíše zapomíná. Knihovna tedy byla napsána i v jazyce C++, ale odladěna pouze na operačním systému Linux. Při práci s částmi programu, které se přímo netýkají samoopravných kódů, jsem se mohla opřít o externí pomoc. Díky tomu se povedlo knihovnu v jazyce C++ zprovoznit i na operačním systému Windows.

Dalším problémem bylo, že k načtení obrázku do programu pro dekódování byl využíván ImageMagick, program pro zpracování obrázků. ImageMagick byl pro tento účel příliš rozsáhlý a v případném využití v mobilním telefonu by se nedal použít, bylo tedy potřeba načítání obrázků řešit jinak. Přímo do dékodovacího programu bylo tedy připsáno zpracování 24-bitových bitmapových obrázků. Pokud by vznikla potřeba načítat i jiné formáty, bylo by nutné předřadit nějaký konverzní program, nebo vlastnoručně připsat další funkce.

4.2 Chienovo vyhledávání

Ukážeme ještě rozdíl mezi dvěma měřenými implementacemi Chienova vyhledávání.

Vidíme, že v původní implementaci, ukázané v algoritmu 4.1, je prakticky jediné vylepšení to, že při nalezení všech kořenů (numErrors) se for cyklus ukončí. Prochází se ale přes proměnnou i, která postupně prochází množinu {1,...,255}. Jak jsme upozornili už v kapitole 2.2, v paměti jsou prvky tělesa reprezentovány vektory koeficientů. Původní implementace tedy bere postupně prvky tělesa podle stupně polynomu; polynomy s nižším stupněm se zkoušejí jako první. Stupeň po-

```
Input: Λ(x)
Output: J
int e = 0;
for (int i = 1; i < field→getSize() AND e < numErrors; i++) do
    if (errorLocator→evaluateAt(i) == 0) then
        result[e] = field→inverse(i); e++;
    end
end
```

Algoritmus 4.1: Původní implementace

lynomu, který reprezentuje prvek $\mathbb{F}_{2^8}$ nám ale neříká nic o tom, kolikátá mocnina primitivního prvku daný prvek je. Přitom to je to, co nás zajímá, neboť mocnina určuje pozici chyby.

```
Input: Λ(x)
Output: J
int e = 0;
for (int i = field→getSize()-1; i > 0 AND e < numErrors; i–) do
    if (errorLocator→evaluateAt(field→exp(i)) == 0) then
        result[e]= field→exp(255-i);
        e++;
    end
end
```

Algoritmus 4.2: Upravená implementace

V algoritmu 4.2 je popsán změněný algoritmus. Změny jsou minimální, a přesto, jak je vidět v tabulce 4.1, algoritmus urychlily. První změnou je procházení for cyklu nikoli od 1 do 255, ale naopak od 255 do 1. Druhá změna je dosazování nikoli i, ale α^i, které nám vrátí funkce field→exp(i). Tato funkce pouze vrátí hodnotu expTable[i]. Důvody pro tyto změny byly vysvětleny v kapitole 3.2.8. Navazující třetí změnou je určení inversu. Pochopitelně bychom mohli nechat funkci z algoritmu 4.1 a místo i dosadit field→exp(i). Je to ale v případě, kdy známe mocninu inversu zbytečné a volání více funkcí by zabralo více času.

Jaké zlepšení algoritmus přinese, jsem testovala i na QR kódu z kapitoly 3. Původní algoritmus dekóduje daný kód průměrně za $2751 \cdot 10^{-6}$ sekundy, zatímco vylepšený algoritmus v průměru za $2689{,}2 \cdot 10^{-6}$ sekundy. Upravený algoritmus je tedy lepší, i když jen v rámci desetitisícin vteřin. Je ale nutné připomenout, že měření jsou prováděna na počítači, který má lepší výpočetní schopnosti, než mobilní telefony, pro které jsou aplikace pro čtení QR kódů primárně určeny.

4.3 Testování dat

Testovány byly kódy verze 1, 5, 10, 15, 20, všechny při úrovni zabezpečení L a H. Generovány byly opět na adrese http://zxing.appspot.com/generator a zakódován v nich je text Lorem ipsum v příslušné délce. Verze jsou voleny s rozestupy, aby byl zřetelnější časový rozdíl. Verze vyšší než 20 jsem netestovala hlavně proto,

Tabulka 4.1: Testování

Verze	nepoškozený	poškozený, neefektivní Chienovo vyhledávání	poškozený, efektivní Chienovo vyhledávání
1L	711	–	–
1H	616,3	–	–
5L	1302,3	–	–
5H	1246	–	–
10L	2952,6	4145,3	4141,3
10H	2817	4731	4683
15L	3353,6	5592	5512
15H	3019,6	6771,3	6377
20L	7932	9826,6	9865
20H	4973,3	9777,3	9642,6

že nejsou příliš využívány. Už ve verzi 20 je text o délce 879 znaků, které tvoří 139 slov.

Poškození kódů jsem se snažila provést alespoň částečně náhodně. Využila jsem k tomu program Adobe Photoshop, ve kterém jsem nastavila velký rozptyl stopy a barvy štětce.

Pro měření programu jsem využívala funkci std::clock(), která vrací počet „tiknutí“ procesoru. Vzhledem k tomu, že program je velmi rychlý, měřil se při jednom testu čas, za který stihne program kód tisíckrát dekódovat. Testy jsem opakovala vždy třikrát se stejnými daty, neboť na procesoru neběží nikdy jen jeden program. Při opakování se proto hodnoty trochu měnily. V tabulce 4.1 jsou uvedeny průměry měření v jednotkách „tiky procesoru“. Pokud bychom chtěli hodnoty převést na sekundy, je potřeba je vynásobit hodnotou 10^{-3}; pokud bychom chtěli zjistit čas dekódování jednoho kódu, je potřeba vynásobit hodnoty číslem 10^{-6}.

Políčka v tabulce 4.1 u kódů 1L až 5H jsou proškrtaná proto, že program kódy, i přes relativně malou míru poškození, již nebyl schopný detekovat. Detekce obrazu v implementaci by se jistě dala vylepšit.

Z tabulky 4.1 vidíme, že u nepoškozených QR kódů trvá dekódování kódu s úrovní zabezpečení L déle než kódu s úrovní zabezpečení H, ačkoli úroveň H je vyšší. Jedním z možných důvodů, proč tomu tak je, je volba použitých RS kódů.

Obecně kódy s nižší úrovní zabezpečení obsahují méně bloků, které jsou o to delší. Např. ve verzi 15L je to 5 bloků délky 109 a jeden blok délky 110 proti jedenácti blokům délky 36 a sedmi blokům délky 37 ve verzi 15H. V momentě, kdy je kód bez chyby (a není potřeba provádět Euklidův algoritmus), se pravděpodobně projeví to, že dosazování do polynomu stupně 109 či 110 (při počítání syndromu) je výpočetně náročnější, ačkoli se provádí méněkrát. Navíc je v kódu s menší úrovní zabezpečení, ale stejné velikosti, zakódováno více dat, která se musejí zpracovávat.

Naopak při opravování chyb složitost roste postupně. Může za to Euklidův algoritmus, Chienovo vyhledávání a Forneyho algoritmus, neboť ty se, pokud je syndrom nulový, neprovádějí.

Dále si všimněme, že u obou kódů verze 20 je původní implementace Chienova vyhledávání rychlejší. To se pochopitelně může stát, záleží, na kterých místech konkrétně jsou chyby. U vylepšeného algoritmu ale máme jistotu, že budeme při

správných výpočtech dosazovat maximálně 153 hodnot. To je totiž délka nejdelšího RS kódu, viz. kapitola 2.2. Zatímco u staré implementace žádnou takovou jistotu nemáme. Pokud je chyba například na 12 pozici (od konce), bude kořenem $\Lambda(x)$ prvek:

$$\alpha^{-11} = \alpha + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 = (11111010) = 250.$$

Přitom kódová slova delší než dvanáct, u kterých by mohlo dojít k chybě na dvanácté pozici, mají všechny QR kódy.

Na závěr bych ráda podotka, že cílem měření bylo poskytnout ilustraci toho, jak se dají jednotlivé velikosti navzájem porovnat z hlediska rychlosti algoritmu a zda navržené vylepšení implementace Cheinova vyhledávání bude opravdu urychlením. Ambicí nebylo vytvořit sadu měření, která by byla statisticky konkluzivní.

Závěr

Ve stručnosti jsme se seznámili s QR kódy a popsali matematické principy, na kterých jsou založeny. Práce nezabíhá do přílišných technických detailů vzhledu QR kódů, které jsou všechny popsány v [1], ale naopak se snaží shrnout ty nejdůležitější údaje. Poněkud podrobněji se pak zabývá převodem kódu do matematické reprezentace, aby byla lépe vidět souvislost mezi moduly vytištěnými na papíře a reprezentací tělesa pomocí polynomů. Pro názornost je zařazena kapitola, ve které je „ručně“ dekódován QR kód obsahující chyby.

Jedním z výsledků práce je také program schopný dekódovat QR kódy, který je dostatečně kompaktní, aby byl použit v prostředí mobilního telefonu. Ačkoli vycházel z již hotové knihovny ZXing ([6]), bylo v něm potřeba provést řadu úprav. Jedním z přínosů práce je určitě zprovoznění ZXingu v jazyce C++ a odstranění podpůrné grafické knihovny.

Za vlastní příspěvek lze považovat shrnutí popisu QR kódů a vyjasnění některých matematických nepřesností, které jsou uvedeny v [1], především ohledně generujícího polynomu. Oproti [2] se mi podařilo zjednodušit důkaz platnosti klíčové rovnice a výklad dekódování RS kódů pomocí Euklidova algoritmu jsem doplnila o důkaz lemmatu 2.6. Vlastním příspěvkem je jistě i ilustrace a propojení informací z prvních dvou kapitol na konkrétním příkladě. A nakonec uvádím urychlení algoritmu Chienova vyhledávání.

Seznam použité literatury

[1] ISO/IEC 18004:2006(E). *Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification.* 2006.

[2] ROTH, Ron M. *Introduction to Coding Theory.* Cambridge: Cambridge University Press, 2006, xi, 566 s. ISBN 978-0-521-84504-5.

[3] REED, I.S a G. SOLOMON. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics.* 1960, roč. 8, č. 2, s. 5.

[4] DENSO WAVE INCORPORATED. *QRcode.com?DENSO WAVE* [online]. [cit. 2013-05-19]. Dostupné z: http://www.qrcode.com/en/

[5] KLIMA, Richard E, Neil SIGMON a Ernest STITZINGER. *Applications of abstract algebra with Maple.* Boca Raton: CRC Press, c2000, xii, 256 p. ISBN 08-493-8170-3.

[6] *ZXing - Multi-format 1D/2D barcode image processing library with clients for Android, Java - Google Project Hosting* [online]. [cit. 2013-05-19]. Dostupné z: http://code.google.com/p/zxing/

[7] DENSO WAVE INCORPORATED. *DENSO WAVE INCORPORATED* [online]. (c)2007-2013 [cit. 2013-05-19]. Dostupné z: http://www.denso-wave.com/en/

[8] DENSO CORPORATION. *DENSO CORPORATION* [online]. (c)2013 [cit. 2013-05-19]. Dostupné z: http://www.globaldenso.com/en/

Seznam obrázků

Seznam algoritmů

Seznam tabulek