VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA VÝPOČETNÍ NÁROČNOSTI SAMOOPRAVNÝCH KÓDŮ

ANALYSIS OF COMPUTATIONAL EFFORT OF SELF-CORRECTING CODES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ BÁRTŮ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

143962

Ústav: Ústav počítačových systémů (UPSY)
Student: **Bártů Tomáš**
Program: Informační technologie
Specializace: Informační technologie
Název: **Analýza výpočetní náročnosti samoopravných kódů**
Kategorie: Bezpečnost
Akademický rok: 2022/23

Zadání:

1. Nastudujte problematiku samoopravných kódů se zaměřením na pokročilé kódy schopné korekce vícenásobných chyb (BCH kódy, případně Reed-Solomonovy kódy).
2. Implementujte alespoň jeden z uvedených typů kódů a ověřte jeho funkčnost s různými požadavky na délku zabezpečovaných dat a nejvyšší předpokládanou násobnost chyb.
3. Na základě dostupné literatury zvolte alespoň dva postupy pro nalezení chyb v kódových slovech. Vykonejte sadu měření výpočetní náročnosti těchto postupů ve zvoleném prostředí s kódovými scénáři zavedenými v bodu 2.
4. Zdokumentujte a zhodnoťte dosažené výsledky a diskutujte možnosti případné další optimalizace navržených řešení.

Literatura:
Dle doporučení vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:
Splnění prvních dvou bodů zadání, přičemž v bodě 2 postačí demonstrovat funkční prototyp alespoň jednoho zvoleného kódu.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Práce se zabývá samoopravnými kódy. Konkrétně kódováním a dekódováním Reed-Solomonových kódů. Je zde popsán úvod do samoopravných kódů, dále princip kódování následovaný popisem dekódování Reed-Solomonových kódů pomocí Petterson-Gorenstein-Zierlerova, Berlekamp-Masseyho a Euklidova algoritmu. Posléze je zde popsána implementace, jež realizuje některé ze zmíněných algoritmů. Následují experimenty s aplikací, které porovnávají časovou a iterační náročnost kódovacího a dekódovacího procesu.

Abstract

The work deals with error-correcting codes, specifically encoding and decoding Reed-Solomon codes. An introduction to error-correcting codes is provided, followed by a description of the encoding and decoding principle of Reed-Solomon codes using the Petterson-Gorenstein-Zierler, Berlekamp-Massey, and Euclidean algorithms. Implementation is then described, which realizes some of the mentioned algorithms. This is followed by experiments with applications that compare the time and iteration complexity of the encoding and decoding processes.

Klíčová slova

Reed-Solomonův kód, Samoopravný kód, Berlekamp-Masseyův algoritmus, Euklidův algoritmus, Petterson-Gorenstein-Zierlerův algoritmus

Keywords

Reed-Solomon code, Error correction code, Berlekamp-Massey algorithm, Euclidean algorithm, Petterson-Gorenstein-Zierler algorithm

Citace

BÁRTŮ, Tomáš. *Analýza výpočetní náročnosti samoopravných kódů.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Analýza výpočetní náročnosti samoopravných kódů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Bártů
7. května 2023

Poděkování

Rád bych tímto poděkoval panu Ing. Michalovi Bidlovi Ph.D. za jeho ochotu a cenné rady, které mi v průběhu mé práce na bakalářské práci poskytl.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato bakalářská práce se zabývá tématem samoopravných kódů (anglicky *Self-Correcting Codes* nebo pod známějším názvem *Error Correcting Codes – ECC*). Tyto kódy slouží k detekci a opravě chyb, které mohly nastat na datech při přenosu po přenosovém médiu vlivem nežádoucích rušení. Ideou samoopravných kódů je to, že se do původní zprávy zakomponují dodatečné informace (redundance), s jejichž pomoci je možno za použití konkrétních algoritmů zajistit detekci chyb a jejich případnou opravu.

Samoopravné kódy lze v určité podobě přirovnat k následující situaci vyskytující se běžně v životě. Sedíte v kině a díváte se na dlouho očekávané pokračování vašeho oblíbeného filmu, kde vlivem rušení způsobeného šustěním pytlíků od brambůrků slyšíte, jak hlavní hrdina praví: „Není to o letadle, je to o bilatovi.“ Běžný smysl nám říká, že se ve větě nachází chyba ve slově „bilatovi“. Smysl nám také říká, že správná forma slova je „pilotovi“. Jak je to možné? Náš mozek provedl analýzu věty, kde označil slovo „bilatovi“ jako chybné, jelikož ho nezná. V dalším kroku navrhl seznam podobných slov, jež zná a mohla by být správná. Následně z nich vybral slovo „pilotovi“, které v daném kontextu má největší smysl. Tento a další podobné příklady názorně demonstrují, co jsou a jak fungují samoopravné kódy.

Oblastí samoopravných kódů se začal zabývat Richard Hamming, který v roce 1950 popsal tzv. Hammingův kód [11]. Tento kód slouží k opravě jednonásobné chyby a k detekci až dvojnásobných chyb. S rostoucí složitostí systémů a se zvětšujícími se objemy dat přestaly dostačovat existující samoopravné kódy a bylo potřeba pokročilejších kódů umožňujících opravovat vícenásobné chyby. U těchto kódů bylo typicky požadováno, aby existoval systematický a efektivní postup kódování a dekódování zprávy, a aby existovala možnost, kterou lze kódy parametrizovat na základě jejich využití v různých systémech, kde každý systém může mít různé požadavky na předpokládanou nejvyšší násobnost chyb ve zprávě. Těmto požadavkům vyhovují zejména BCH kódy [6, 13] a Reed-Solomonovy (RS) kódy [10].

Reed-Solomonovy kódy byly popsány v roce 1960 Irvingem S. Reedem a Gustavem Solomonem. Rozšířené uplatnění našly až po roce 1977, kdy byly použity ve vesmírném programu NASA Voyager, kde v systému sondy Voyager 2 byl nainstalován efektivní kodér RS kódů, avšak efektivní dekodér ještě nebyl objeven. Vědci tímto počínáním riskovali, když předpokládali, že v době, kdy sonda dosáhne Uranu, bude již dekodér vyvinut. Tento risk se jim opravdu vyplatil a v roce 1986, kdy se sonda přiblížila k Uranu, byly systémy Voyageru vylepšeny o efektivní systém dekódování RS kódů [9]. V současné době lze nalézt uplatnění těchto kódů v běžně se vyskytující technice, jako například v CD, DVD, SSD, QR kódech nebo se tyto kódy hojně využívají v telekomunikačních systémech.

Cílem práce je provést analýzu výpočetní náročnosti samoopravných kódů. Konkrétně se práce zaměřuje na problematiku Reed-Solomonových kódů. U implementace těchto kódů je změřena jejich výpočetní náročnost, přičemž jsou zde zvoleny dva postupy pro nalezení chyby v kódových slovech. Implementace umožňuje nastavit různé požadavky na délku zabezpečovaných dat a nejvyšší předpokládanou násobnost chyb. Toto řešení umožňuje zavést různé kódové scénáře. Dosažené výsledky těchto měření jsou následně analyzovány a zdokumentovány.

Kapitola 2

Galoisova tělesa

S myšlenkou těchto algebraických struktur přišel francouzský matematik Évariste Galois po němž, jsou tyto struktury pojmenovány. Konkrétně se jim říká Galoisova tělesa (anglicky *Galois fields* – *GF*) nebo se lze setkat s překlady jako konečná tělesa či pole.

Nechť p je prvočíslo a $q = p^m$, kde m patří do množiny přirozených čísel, pak konečná tělesa budou značena jako $GF(p^m)$ nebo jako $GF(q)$.

Konečné těleso lze sestrojit za pomoci generujícího polynomu tělesa $F(X)$. Polynomů $F(X)$ stupně m může být pro každý stupeň obecně více, přičemž každý z těchto polynomů generuje jinou abecedu prvků tělesa (pokaždé se však zachovávají všechny vlastnosti).

V této kapitole bude popsána konstrukce konečného tělesa za pomocí primitivního nebo generujícího polynomu a dále budou popsány operace nad konečnými tělesy. Tato kapitola primárně vychází z [4, 15].

2.1 Konstrukce tělesa pomocí primitivního polynomu

Nechť existuje těleso $GF(2^m)$. Každý jeho prvek lze vypočítat jako zbytek po dělení x^i tzv. primitivním polynomem, kde $i = 0, \ldots, 2^{2m-2}$.

Primitivní polynom $p(x)$[1] je takový polynom stupně m, jenž beze zbytku dělí polynom $x^j + 1$, kde $j = p^m - 1 = 2^m - 1$. Zároveň však beze zbytku nedělí polynom $x^k + 1$, kde $k < j$ [4].

2.2 Konstrukce tělesa pomocí generujícího polynomu a primitivního prvku

Nechť existuje těleso $GF(2^m)$. Každý jeho prvek lze vypočítat jako zbytky po dělení mocnin tzv. primitivního prvku $\alpha \in GF(2^m)$, respektive α^i, kde $i = 0, \ldots, m-1$, generujícím polynomem stupně m, jehož prvky jsou z $GF(2)$ a zároveň je nerozložitelný v $GF(2)$ [4]. Více o generujících polynomech viz sekci 3.3.1.

Primitivní prvek je takový prvek tělesa $GF(q)$, jehož řád je $q-1$, kde se řádem rozumí takové nejmenší kladné celé číslo n, pro které $\alpha^n = 1$. Tímto prvkem lze vygenerovat všechny nenulové prvky $GF(q)$. Takovýto prvek existuje alespoň jeden pro každé konečné těleso [16].

[1] Existují tabulky primitivních polynomů, například https://www.partow.net/programming/polynomials/index.html

Příklad 1
Nechť je dán primitivní polynom $p(x) = x^4 + x + 1$. Prvky konečného tělesa $GF(2^4)$, které lze z $p(x)$ vytvořit, jsou sepsány v tabulce 2.1.

Tabulka 2.1: Tabulka konečného tělesa vytvořená za pomoci primitivního polynomu $p(x) = x^4 + x + 1$.

Exponenciální reprezentace	Polynomiální reprezentace	Vektorová reprezentace
0	0	$\begin{pmatrix}0 & 0 & 0 & 0\end{pmatrix}$
α^0	1	$\begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}$
α^1	x^1	$\begin{pmatrix}0 & 0 & 1 & 0\end{pmatrix}$
α^2	x^2	$\begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}$
α^3	x^3	$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}$
α^4	$x^1 + 1$	$\begin{pmatrix}0 & 0 & 1 & 1\end{pmatrix}$
α^5	$x^2 + x^1$	$\begin{pmatrix}0 & 1 & 1 & 0\end{pmatrix}$
α^6	$x^3 + x^2$	$\begin{pmatrix}1 & 1 & 0 & 0\end{pmatrix}$
α^7	$x^3 + x^1 + 1$	$\begin{pmatrix}1 & 0 & 1 & 1\end{pmatrix}$
α^8	$x^2 + 1$	$\begin{pmatrix}0 & 1 & 0 & 1\end{pmatrix}$
α^9	$x^3 + x^1$	$\begin{pmatrix}1 & 0 & 1 & 0\end{pmatrix}$
α^{10}	$x^2 + x^1 + 1$	$\begin{pmatrix}0 & 1 & 1 & 1\end{pmatrix}$
α^{11}	$x^3 + x^2 + x^1$	$\begin{pmatrix}1 & 1 & 1 & 0\end{pmatrix}$
α^{12}	$x^3 + x^2 + x^1 + 1$	$\begin{pmatrix}1 & 1 & 1 & 1\end{pmatrix}$
α^{13}	$x^3 + x^2 + 1$	$\begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$
α^{14}	$x^3 + 1$	$\begin{pmatrix}1 & 0 & 0 & 1\end{pmatrix}$

Vektorová reprezentace je zkonstruována z polynomiální reprezentace tak, že chybějící mocniny x^j určitého polynomu jsou reprezentovány hodnotou 0 a výskyty mocniny x^j jsou reprezentovány hodnotou 1, pro $j = 0, 1, \ldots, m-1$. Například prvky konečného tělesa α^7, α^8 lze zapsat následujícími reprezentacemi:

$$\alpha^7 = x^3 + 0 + x^1 + 1 = \begin{pmatrix}1 & 0 & 1 & 1\end{pmatrix},$$
$$\alpha^8 = 0 + x^2 + 0 + 1 = \begin{pmatrix}0 & 1 & 0 & 1\end{pmatrix}.$$

Za případu, že se stále bude dodržovat stejný formát zápisu, nezáleží na tom v jakém pořadí bude, jak vektorová, tak polynomiální reprezentace prvku konečného tělesa zapsána. [9].

2.3 Operace nad konečnými tělesy

Jelikož operace neprobíhají na binární úrovni, vytvářejí se složitější operační tabulky, kde stačí definovat pouze operaci násobení a operaci sčítaní. Operaci sčítaní lze nejjednodušeji aplikovat tak, že se nad vektorové reprezentaci obou operandů z konečného tělesa provede operace XOR. Operaci násobení lze provádět sečtením exponentů obou operandů a tento součet podrobit operaci modulo $2^m - 1$ [9]. V tabulkách 2.2 a 2.3 jsou sepsány operační tabulky pro operaci sčítání a násobení v $GF(2^4)$, jež byly vygenerovány pomocí primitivního polynomu $p(x) = x^4 + x + 1$.

Operace v operačních tabulkách 2.2 a 2.3, jež jsou pro přehlednost uspořádány pohromadě na této stránce, jsou symetrické, a proto za účelem lepší čitelnosti v tabulkách je zobrazena pouze nezbytná část těchto tabulek. Tyto tabulky se budou dále využívat při výpočtech určitých příkladů, jež jsou v této práci uvedeny.

Tabulka 2.2: Operační tabulka znázorňující operaci "+" prvků z $GF(2^4)$.

	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α^0	0	α^4	α^8	α^{14}	α^1	α^{10}	α^{13}	α^9	α^2	α^7	α^5	α^{12}	α^{11}	α^6	α^3
α^1		0	α^5	α^9	α^0	α^2	α^{11}	α^{14}	α^{10}	α^3	α^8	α^6	α^{13}	α^{12}	α^7
α^2			0	α^6	α^{10}	α^1	α^3	α^{12}	α^0	α^{11}	α^4	α^9	α^7	α^{14}	α^{13}
α^3				0	α^7	α^{11}	α^2	α^4	α^{13}	α^1	α^{12}	α^5	α^{10}	α^8	α^0
α^4					0	α^8	α^{12}	α^3	α^5	α^{14}	α^2	α^{13}	α^6	α^{11}	α^9
α^5						0	α^9	α^{13}	α^4	α^6	α^0	α^3	α^{14}	α^7	α^{12}
α^6							0	α^{10}	α^{14}	α^5	α^7	α^1	α^4	α^0	α^8
α^7								0	α^{11}	α^0	α^6	α^8	α^2	α^5	α^1
α^8									0	α^{12}	α^1	α^7	α^9	α^3	α^6
α^9										0	α^{13}	α^2	α^8	α^{10}	α^4
α^{10}											0	α^{14}	α^3	α^9	α^{11}
α^{11}												0	α^0	α^4	α^{10}
α^{12}													0	α^1	α^5
α^{13}														0	α^2
α^{14}															0

Tabulka 2.3: Operační tabulka znázorňující operaci "·" prvků z $GF(2^4)$.

	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α^0	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α^1		α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^0
α^2			α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^0	α^1
α^3				α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^0	α^1	α^2
α^4					α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^0	α^1	α^2	α^3
α^5						α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^0	α^1	α^2	α^3	α^4
α^6							α^{12}	α^{13}	α^{14}	α^0	α^1	α^2	α^3	α^4	α^5
α^7								α^{14}	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^8									α^1	α^2	α^3	α^4	α^5	α^6	α^7
α^9										α^3	α^4	α^5	α^6	α^7	α^8
α^{10}											α^5	α^6	α^7	α^8	α^9
α^{11}												α^7	α^8	α^9	α^{10}
α^{12}													α^9	α^{11}	α^{12}
α^{13}														α^{11}	α^{12}
α^{14}															α^{13}

Příklad 2 slouží jako názorný příklad, jak násobit a sčítat prvky z konečného tělesa bez použití operačních tabulek.

Příklad 2

Nechť platí konečné těleso z příkladu 1. Příklad součtu dvou operandů může vypadat následovně

$$\alpha^7 + \alpha^8 = \alpha^7 \text{ XOR } \alpha^8 = (1011) \text{ XOR } (0101) = (1110) = \alpha^{11}$$

a příklad násobení dvou operandů může vypadat následovně

$$\alpha^7 \alpha^8 = \alpha^{7+8} = \alpha^{15 \bmod 15} = \alpha^0 = 1$$

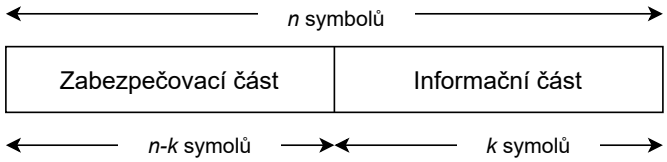
Nechť $f(x)$ je polynom stupně m s koeficienty z $GF(2)$, který je ireducibilní v $GF(2)$ (tj. nemá 0 ani 1 jako svůj kořen) a zároveň má kořeny z $GF(2^m)$. Asociované kořeny jsou všechny mocniny β^{2^k}, kde $k \geq 0$, pro které platí, že $f(\beta) = 0$ (tj. β je kořenem $f(x)$), pro $\beta \in GF(2^m)$. Minimální polynom $\phi(x)$ je polynom nejnižšího stupně, pro který platí $\phi(\beta) = 0$ [4].

Kapitola 3

Samoopravné kódy

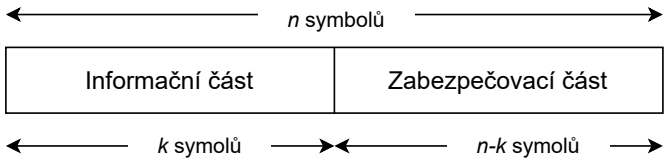
Všechny kódy pro opravu chyb se zakládají na stejném základním principu a to takovém, že se redundantní informace přidá k informacím (zprávě), aby bylo možné opravit případné chyby, které mohou vzniknout při přenosu nebo ukládání. V základní formě se redundantní symboly připojují k informačním symbolům, aby se získala zakódovaná posloupnost nebo kódové slovo [20].

V této práci se bude pracovat s kódovými slovy jejichž struktura je znázorněna na obrázku 3.1, kde jednotlivé parametry jsou detailněji popsány v podsekci 3.1.



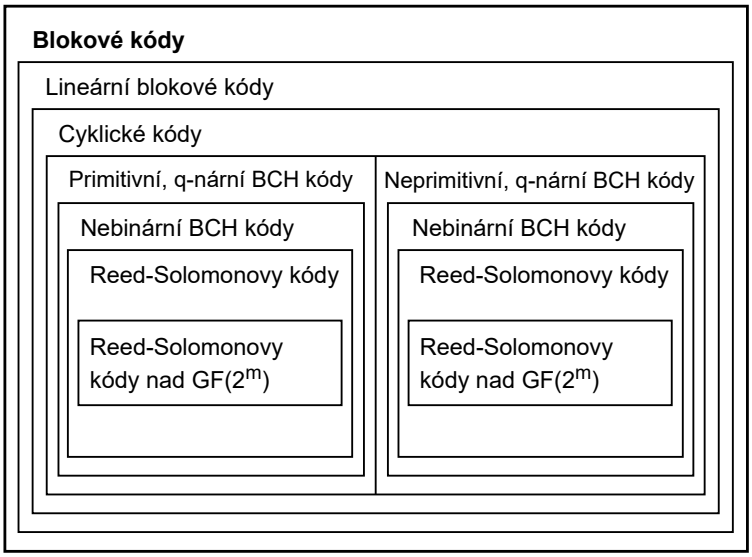
Obrázek 3.1: Struktura kódového slova. Obrázek byl převzat s úpravami z [20].

Formát kódového slova může být také zapsán v opačném pořadí, tedy nejdříve se vyskytuje část, jež reprezentuje informaci a po ní se nachází zabezpečovací část. Tento popis je znázorněn na obrázku 3.2. Avšak oba ze znázorněných způsobů nejsou jediné. V znázorněných případech se konkrétně jedná o kódová slova zapsaná v systematickém tvaru, více o tomto tvaru v podsekci 3.4. Druhým způsobem, jak znázornit kódové slovo, je v tzv. nesystematickém tvaru, kde nelze jednoduše z kódového slova odlišit jeho informační a zabezpečovací část.



Obrázek 3.2: Alternativní struktura kódového slova uváděná ve většině literatur. Obrázek byl převzat z [20].

V této kapitole budou popsány základní třídy samoopravných kódů. Mezi těmito třídami existují závislosti, jež jsou znázorněny na diagramu 3.3. Konkrétně třída Reed-Solomonových kódů bude obsáhle popsána v následující kapitole.



Obrázek 3.3: Diagram reprezentující závislost mezi různými druhy kódu. Diagram byl převzat z [9]

3.1 Blokové kódy

Hlavní charakteristikou blokových kódu je jejich práce s bloky dat o pevně dané velikosti. Kodér tedy vstupní informaci rozdělí na sekvence na sobě nezávislých bloků, kde každý blok nese k informačních bitů. Jinými slovy, blokové kódy díky tomu, že jsou bloky zprávy na sobě nezávislé, nepotřebují si uchovávat stavy mezi jednotlivými bloky[1]. Informační bity jsou reprezentovány jako k-tice $u = (u_0, u_1, \ldots, u_{k-1})$, jež se nazývá zpráva. Celkový počet zpráv, jež lze takto vytvořit je 2^k. Kodér každou zprávu u transformuje a na svůj výstup odešle n-tici $v = (v_0, v_1, \ldots, v_{n-1})$ nazývanou jako kódové slovo. Celkový počet kódových slov je shodný s počtem zpráv, tedy 2^k.

Přidanou zabezpečovací část tvoří tzv. redundantní bity, jejichž počet je $n - k$ za předpokladu, že $k < n$. Redundanci R definoval R. Hamming jako poměr délky kódového slova k délce zprávy tedy jako

$$R = \frac{n}{k}, \tag{3.1}$$

respektive kolik bitů slouží k zabezpečení jednoho bitu. Rovnice se používá k měření účinnosti kódu [11].

Převrácená hodnota k rovnici 3.1 je

$$r = \frac{k}{n} \tag{3.2}$$

a reprezentuje poměr počtu bitů zprávy k délce kódového slova. Tento poměr se anglicky označuje jako *„code rate“*. Do češtiny ho lze přeložit jako informační poměr [1].

Reed-Solomonovy kódy mají kódová slova o pevně dané délce, patří tedy do třídy blokových kódů. Sekce primárně vychází z literatury [15].

[1] Protikladem blokových kódů jsou konvoluční kódy, kde jsou vstupy a výstupy závislé na předešlých vstupech a výstupech

Příklad 3
Nechť platí, že kodér jistého blokového kódu dokáže zpracovat 4bitovou zprávu a jeho výstupem je 7bitové kódové slovo. Tabulka 3.1 uvádí, jak by mohl vypadat příklad binárního blokového kódu, přičemž první tři bity kódového slovo jsou redundantní.

Tabulka 3.1: Příklad binárního blokového kódu, kde $k = 4$ a $n = 7$. Tabulka byla převzata z [15]

Zpráva	**Kódové slovo**
$(0\ 0\ 0\ 0)$	$(0\ 0\ 0\ 0\ 0\ 0\ 0)$
$(1\ 0\ 0\ 0)$	$(1\ 1\ 0\ 1\ 0\ 0\ 0)$
$(0\ 1\ 0\ 0)$	$(0\ 1\ 1\ 0\ 1\ 0\ 0)$
$(1\ 1\ 0\ 0)$	$(1\ 0\ 1\ 1\ 1\ 0\ 0)$
$(0\ 0\ 1\ 0)$	$(1\ 1\ 1\ 0\ 0\ 1\ 0)$
$(1\ 0\ 1\ 0)$	$(0\ 0\ 1\ 1\ 0\ 1\ 0)$
$(0\ 1\ 1\ 0)$	$(1\ 0\ 0\ 0\ 1\ 1\ 0)$
$(1\ 1\ 1\ 0)$	$(0\ 1\ 0\ 1\ 1\ 1\ 0)$
$(0\ 0\ 0\ 1)$	$(1\ 0\ 1\ 0\ 0\ 0\ 1)$
$(1\ 0\ 0\ 1)$	$(0\ 1\ 1\ 1\ 0\ 0\ 1)$
$(0\ 1\ 0\ 1)$	$(1\ 1\ 0\ 0\ 1\ 0\ 1)$
$(1\ 1\ 0\ 1)$	$(0\ 0\ 0\ 1\ 1\ 0\ 1)$
$(0\ 0\ 1\ 1)$	$(0\ 1\ 0\ 0\ 0\ 1\ 1)$
$(1\ 0\ 1\ 1)$	$(1\ 0\ 0\ 1\ 0\ 1\ 1)$
$(0\ 1\ 1\ 1)$	$(0\ 0\ 1\ 0\ 1\ 1\ 1)$
$(1\ 1\ 1\ 1)$	$(1\ 1\ 1\ 1\ 1\ 1\ 1)$

3.2 Lineární blokové kódy

Uvažujeme samoopravný kód K. Nechť K je podmnožinou n-dimenzionálního binárního vektorového prostoru $V_2 = \{0, 1\}^n$. V takovémto případě je kód K lineární. To znamená, že výsledek součtu jakýchkoliv dvou kódových slov z K je rovněž kódové slovo z K. Prostor V_2 je téže někdy označován jako Hammingův prostor [20].

Kódování může být realizováno pomocí maticového násobení. V logických obvodech je realizace provedena za pomoci exkluzivní disjunkce a konjunkce, respektive pomocí hradla XOR a hradla AND. Dále v této práci, pokud se bude pracovat na binární úrovni, budou výpočty prováděny dle tabulky 3.2.

Řekněme, že vektory $\{\bar{v}_0, \bar{v}_1, \ldots, \bar{v}_{k-1}\}$ jsou báze k-dimenzionálního vektorového podprostoru kódu K. V takovém případě může být jakékoliv kódové slovo $\bar{v} \in K$ reprezentováno jako lineární kombinace bázových vektorů vynásobených vhodnými binárními koeficienty. Hodnotu $\bar{v}$ lze zapsat jako:

$$\bar{v} = u_0\bar{v}_0 + u_1\bar{v}_1 + \cdots + u_{k-1}\bar{v}_{k-1}, \tag{3.3}$$

kde $u_i \in \{0, 1\}, 1 \leq i < k$.

Tabulka 3.2: Hodnoty A a B znázorňují vstupy (např. hradel). Operace + a · v pořadí značí exkluzivní disjunkci a konjunkci. Operace − znázorňuje také exkluzivní disjunkci, jelikož z binárního pohledu operace $A - B$ a $A + B$ (*modulo* 2) poskytují stejný výsledek [20]. Tyto operace tedy lze navzájem zaměňovat.

A	B	$A+B$	$A \cdot B$	$A-B$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Rovnici 3.2 lze zapsat jako součin matic, kde G značí generující matici a hodnota $\bar{u} = \begin{pmatrix} u_0 & u_1 & \dots & u_{k-1} \end{pmatrix}$ reprezentuje zprávu.

$$\bar{v} = \bar{u}G, \tag{3.4}$$

kde

$$G = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{k-1} \end{pmatrix} = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k-1,0} & v_{k-1,1} & \cdots & v_{k-1,n-1} \end{pmatrix}. \tag{3.5}$$

Kontrolní maticí H nazýváme takovou matici, jež splňuje podmínku $GH^\intercal = 0$, kde $H^\intercal$ symbolizuje transponovanou matici H. Pro každé platné kódové slovo $\bar{v}$ musí platit rovnice

$$\bar{v}H^\intercal = 0. \tag{3.6}$$

Lineární kódy se v literatuře značí nejčastěji buďto jako (n, k) nebo jako (n, k, d_{min}), kde d_{min} reprezentuje Hammingovu vzdálenost. Asi nejznámějším lineárním blokovým kódem je standardní Hammingův kód, který lze označit za lineární $(7, 4)$ kód.

Při dekódovacím procesu je klíčovým konceptem tzv. syndrom, jež je slovo z V_2 definované jako

$$\bar{s} = \bar{r}H^\intercal. \tag{3.7}$$

Pomocí syndromu $\bar{s}$, jenž je chápán jako soubor příznaků, se indikuje, zda se v přeneseném slově $\bar{r}$ vyskytly nějaké chyby. Kódové slovo $\bar{v} \in K$ je doručeno přenosovým kanálem, který do slova zanesl chyby, které jsou definované jako chybový vektor $\bar{e} \in \{0, 1\}^n$. Přijaté slovo, které bylo tímto kanálem přeneseno, lze zapsat jako

$$\bar{r} = \bar{v} + \bar{e}, \tag{3.8}$$

přičemž syndrom chyby slova $\bar{r}$ lze vypočítat jako

$$\bar{s} = \bar{r}H^\intercal = (\bar{v} + \bar{e})H^\intercal = \bar{e}H^\intercal. \tag{3.9}$$

Sekce vychází z literatury [20].

Příklad 4

Nechť existuje Hammingův kód (7, 4), jehož generující matice G a kontrolní matice H z [21] jsou dány následovně

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \quad H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

a je dána zpráva $\bar{u} = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}$ k zakódování.

Pomocí rovnice 3.4 lze vypočítat kódové slovo v jako

$$\bar{v} = \bar{u}G = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Kontrolu, zda je výsledné kódové slovo korektní, lze provést za pomoci 3.6, kdy výsledek násobení musí být roven nulovému vektoru, respektive syndrom chyby musí být nulový.

$$\bar{v}H^{\intercal} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Zanesením chyby $\bar{e} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$ do kódového slova v vznikne

$$\bar{r} = \bar{v} + \bar{e} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Za pomoci rovnice 3.9 lze zjistit syndrom kódového slova následovně

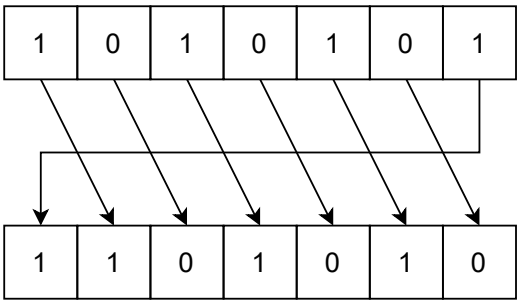
$$\bar{s} = \bar{r}H^{\intercal} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

3.3 Cyklické kódy

Kódy spadající do této třídy disponují cykličností. Cykličnost je vlastnost, jež je popsána v následujícím odstavci.

Nechť $\overleftarrow{v}_0 = \begin{pmatrix} v_0 & v_1 & \dots & v_{n-2} & v_{n-1} \end{pmatrix}$ je kódové slovo lineárního blokového kódu C. Rotací tohoto slova o jednu pozici doleva se získá slovo $\overleftarrow{v}_1 = \begin{pmatrix} v_1 & \dots & v_{n-2} & v_{n-1} & v_0 \end{pmatrix}$. Opakovaným prováděním této operace dojde k vytvoření dalších slov $\overleftarrow{v}_2, \overleftarrow{v}_3, \dots, \overleftarrow{v}_{n-1}$. Orotuje-li se původního slovo celkem n krát v jednom směru, výsledkem rotace bude původní kódové slovo, tudíž je postačující provést operaci rotace o jednu pozici doleva nebo doprava celkem $n-1$ krát. Je však nutné tuto operaci provádět pouze v jednom směru. Pokud by tato podmínka nebyla splněna, nedošlo by ke kontrole každého možného případu, jež mohl při rotaci nastat. V případě, že každé ze slov $\overleftarrow{v}_1, \overleftarrow{v}_2, \overleftarrow{v}_3, \dots, \overleftarrow{v}_{n-1}$ je kódové slovo kódu C, tak tento kód patří do třídy cyklických kódů.

Na obrázku 3.4 je názorně ukázána rotace kódového slova $\bar{v}$ z příkladu 4 o jednu pozici doprava.



Obrázek 3.4: Rotace kódového slova o jednu pozici doprava[2].

Tato sekce a následující podsekce primárně vychází z literatury [22].

3.3.1 Generující polynom

Nechť všechna kódová slova $\bar{v}(x)$ reprezentovaná jako polynom jsou násobky jistého polynomu $g(x)$, přičemž se polynomiální reprezentací myslí například slovo $\overleftarrow{v}_0$ zapsané jako

$$\overleftarrow{v}_0 = v_0 + v_1 x + \cdots + v_{n-2} x^{n-2} + v_{n-1} x^{n-1}, \tag{3.10}$$

Polynomu $g(x)$ se obecně říká generující polynom kódu. Lze dokázat, že je možné polynom $x^n - 1$ vydělit $g(x)$. Polynom $x^n - 1$ lze rozložit na ireducibilní faktory $\phi_j(x)$, kde $j = 1, 2, \dots, l$. Tento vztah lze zapsat s přihlédnutím k tabulce 3.2 jako

$$(x^n + 1) = \phi_1(x)\phi_2(x)\dots\phi_l(x). \tag{3.11}$$

Z vlastností popsaných výše lze generující polynom zapsat jako

$$g(x) = \prod_{j \in J \subseteq \{1,2,\dots,l\}} \phi_j(x). \tag{3.12}$$

Generující polynom $g(x)$ lze tedy vyjádřit jako součin určitého počtu ireducibilních faktorů, jež vznikly rozložením polynomu $(x^n - 1)$.

[2]Inspirace obrázkem z https://en.wikipedia.org/wiki/Cyclic_code#/media/File:Rotate_right.svg

Příklad 5
Rozložením polynomu $x^7 + 1$ na jeho ireducibilní faktory vzniknou polynomy

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Některé ze známých cyklických kódů o délce kódového slova 7, jež vznikly z generujícího polynomu, který je dán rovnicí 3.12 a ireducibilními faktory, jsou:

- Binární cyklický Hammingův $(7, 4)$ kód je generovaný pomocí $g_1(x) = (x^3 + x + 1)$.
- Binární cyklický paritní $(7, 6)$ kód je generovaný pomocí $g_2(x) = (x^3 + x + 1)(x^3 + x^2 + 1)$.
- Binární MLS[3] $(7, 3)$ je generovaný pomocí $g_3(x) = (x + 1)(x^3 + x + 1)$.

Nechť generující polynom $g(x)$ je dán následovně

$$g(x) = g_0 + g_1x + g_2x^2 + \cdots + g_{n-k}x^{n-k}. \tag{3.13}$$

V takovém případě lze každé kódové slovo z C zapsat jako

$$v(x) = m(x)g(x), \tag{3.14}$$

kde $m(x)$ je polynom, jenž reprezentuje zprávu

$$m(x) = m_0 + m_1x + \cdots + m_{k-1}x^{k-1}. \tag{3.15}$$

Musí platit, že stupeň polynomu $m(x)$ bude menší než k.

Lze tedy rovnici 3.14 rozepsat jako

$$\begin{aligned} v(x) &= v_0 + v_1x + v_2x^2 + \cdots + v_{n-1}x^{n-1} \\ &= (g_0 + g_1x + g_2x^2 + \cdots + g_{n-k}x^{n-k})(m_0 + m_1x + \cdots + m_{k-1}x^{k-1}), \end{aligned} \tag{3.16}$$

kde stupeň polynomu kódového slova je menší nebo roven $n - 1$ [20].

[3] Plným názvem „*Maximum-Length-Sequence*“

3.4 Systematický formát kódového slova

Cyklické kódy nemusí být vždy zapsány v systematickém tvaru. Systematický tvar kódového slova je takový tvar, jež je reprezentován na obrázku 3.1, respektive lze jednoduše odlišit informační část od zabezpečovací části.

Pomocí menší úpravy lze cyklický kód zakódovat do systematického tvaru. Nechť existuje vektor zprávy

$$\bar{m} = \begin{pmatrix} m_0 & m_1 & \dots & m_{k-1} \end{pmatrix}. \tag{3.17}$$

Tento vektor lze zapsat také v polynomiálním tvaru, viz rovnici 3.15.

K zakódování zprávy do systematického tvaru je potřeba vytvořit prostor pro zabezpečovací část, a to se docílí tím, že se posune polynom zprávy doprava o $n-k$ pozic. Cílem je dostat následující tvar

$$\begin{pmatrix} 0 & 0 & \dots & 0 & m_0 & m_1 & \dots & m_{k-1} \end{pmatrix}, \tag{3.18}$$

kde část obsahující nulové hodnoty má velikost $n-k$. Pomocí polynomu lze tento tvar zapsat do následující podoby

$$x^{n-k} m(x) \tag{3.19}$$

Pokud generátorem $g(x)$ vydělíme $x^{n-k}m(x)$ dostaneme kvocient společně se zbytkem

$$x^{n-k} m(x) = q(x)g(x) + d(x), \tag{3.20}$$

kde $q(x)$ je podíl a polynom $d(x)$ znázorňuje zbytek, jehož stupeň je menší než $n-k$. Můžeme také říci, že

$$d(x) = x^{n-k} m(x) \bmod g(x), \tag{3.21}$$

kde hodnotu $d(x)$ lze zapsat jako

$$\begin{pmatrix} d_0 & d_1 & \dots & d_{n-k-1} & 0 & 0 & \dots & 0 \end{pmatrix}. \tag{3.22}$$

Úpravou rovnice 3.20 vznikne

$$x^{n-k} m(x) - d(x) = q(x)g(x) = c(x). \tag{3.23}$$

Jelikož $x^{n-k}m(x) - d(x)$ je násobkem generujícího polynomu $g(x)$ musí se tedy jednat o kódové slovo, jehož reprezentace je

$$\begin{pmatrix} -d_0 & -d_1 & \dots & -d_{n-k-1} & m_0 & m_1 & \dots & m_{k-1} \end{pmatrix}. \tag{3.24}$$

Zabezpečovací informace se nachází na prvních $n-k$ pozicích a zpráva je obsažena na posledních k pozicích kódového slova. Sekce primárně vychází z literatury [18].

3.5 BCH kódy

Tyto samoopravné kódy popsali ve své práci [6] pánové Raj Chandra Bose a Dijen K. Ray-Chaudhuri a nezávisle na nich ve své práci, pán Alexis Hocquenghem [13].

Tyto kódy lze rozdělit na primitivní a neprimitivní BCH kódy nebo na binární a nebinární BCH kódy. Primitivní BCH kódy mají délku kódového slova n rovnou $p^m - 1$, zatímco ne-primitivní mají délku různou od $p^m - 1$. Popis binárních a nebinárních BCH kódů bude popsán níže. Sekce a její podsekce primárně vychází z literatury [5, 9].

3.5.1 Binární BCH kódy

Kódová slova jsou tvořena ze symbolů z $GF(p) = GF(2)$. Prvky konečného tělesa jsou z $GF(q) = GF(p^m) = GF(2^m)$, kde q značí počet prvků v tělese. Koeficienty generujícího polynomu $g(x)$ jsou z $GF(2)$ a jeho kořeny jsou z $GF(2^m)$.

Nechť t značí korekční schopnost, pak pro každé celé číslo $m > 2$ a $t < 2^{m-1}$ existuje binární BCH kód s následujícími vlastnostmi
Délka kódového slova:

$$n = 2^m - 1 \tag{3.25}$$

Hammingova vzdálenost:

$$d_{min} \geq 2t + 1 \tag{3.26}$$

Počet paritních bitů:

$$n - k \leq mt \tag{3.27}$$

3.5.2 Nebinární BCH kódy

Kódová slova jsou tvořena ze symbolů z $GF(2^m)$ a prvky konečného tělesa jsou také z $GF(2^m)$. Koeficienty generujícího polynomu $g(x)$ jsou z $GF(2^m)$ a jeho kořeny jsou z $GF(q^z)$. Platí, že $m > 1$ a $z > 2$ a délka kódových slov je $n = q^z - 1$ symbolů, kde $q = 2^m$.

Nechť platí vlastnosti z binárních BCH kódů. Tyto vlastnosti lze zobecnit na [16]
Délka kódového slova:

$$n = q^m - 1 \tag{3.28}$$

Hammingova vzdálenost:

$$d_{min} \geq 2t + 1 \tag{3.29}$$

Počet paritních bitů:

$$n - k \leq 2mt \tag{3.30}$$

3.6 Úvod do Reed-Solomonových kódů

Jsou nejdůležitější podtřídou nebinárních BCH kódů. Kódy této podtřídy se nazývají Reed-Solomonovy (RS) kódy na počest jejich objevitelů Irvinga S. Reeda a Gustaveho Solomona, kteří je popsali ve své práci [10]. Reed-Solomonovy kódy nalezly uplatnění pro korekci chyb jak v digitálních komunikačních systémech (DVB-T2), tak v úložných systémech (CD, paměti Flash, QR kódy) [16]. Reed-Solomonovy kódy se řadí do třídy nebinárních BCH, cyklických, lineárních blokových kódů. Problematika Reed-Solomonvých kódů bude popsána v následující kapitole.

Kapitola 4

Reed-Solomonovy kódy

Tato kapitola se bude zabývat jednou z nejpokročilejších tříd samoopravných kódů, a to konkrétně Reed-Solomonovými (RS) kódy. V následujících částech bude popsán obecný princip RS kódů, jeho vlastností a parametrů, princip generujícího polynomu, způsob kódování (zabezpečení) informace a posléze problematika dekódování a hledání chyb. Tyto principy budou v ukázány na jednoduchém příkladu, který se bude postupně prolínat jednotlivými sekcemi. Pro názornost se tento příklad bude týkat věty zmíněné v úvodní kapitole, konkrétně věty: „Není to o letadle, je to o pilotovi.“. Pro jednoduchost se zaměříme pouze na jedno slovo, a to přesněji na slovo: „pilotovi“, které podrobíme kódovacímu a dekódovacímu procesu. Jednotlivé polynomy budou v této a v následujících kapitolách psány pomocí velkých písmen, aby bylo jednoznačně určeno, že se již pracuje s Reed-Solomonovými kódy, oproti přechozím kapitolám, kdy byly polynomy zapisovány malými písmeny. Tato kapitola primárně vychází z [9] pokud nebude uvedeno jinak.

4.1 Popis Reed-Solomonových kódů

Kódová slova jsou tvořena ze symbolů z $GF(2^m)$. Totéž také platí pro koeficienty a kořeny generujících polynomů. Reed-Solomonovy kódy jsou speciální třídou nebinárních BCH kódů pro které platí, že $z = 1$, vizte sekci 3.5.2. Obecně pro tento kód platí následující nerovnosti:

$$m > 2, \tag{4.1}$$
$$t < 2^{m-1} \tag{4.2}$$

Lze tak sestrojit Reed-Solomonovy kódy s různými vlastnostmi, jež se týkají například délky kódového slova, zabezpečovací schopnosti, délky informace k zabezpečení.

Vlastnosti RS kódů lze popsat následujícími rovnicemi [16]:
Délka kódového slova:

$$n = p^m - 1 \tag{4.3}$$

Počet paritních symbolů:

$$n - k = 2t \tag{4.4}$$

Informační délka zprávy:

$$k = p^m - 1 - 2t \tag{4.5}$$

Hammingova vzdálenost:

$$d_{min} = 2t + 1 \tag{4.6}$$

Poměr chyb t, který je kód schopný opravit, k celkové délce kódového slova definuje hodnota c. Rovnice pro jeho výpočet je

$$c = \frac{t}{n}. \tag{4.7}$$

Na rozdíl od dříve zmíněných BCH kódů, tak u Reed-Solomonových kódů lze z parametru n a k určit korekční schopnost t kódu. U kódů BCH existují speciální tabulky (podobné tabulce z příkladu 6), ve kterých jsou vypsány korekční schopnosti BCH kódů při určitém n a k, viz literaturu [15, 16]. Korekční schopnost kódu lze vyjádřit z rovnice 4.4 jako [23]

$$t = \left\lfloor \frac{n-k}{2} \right\rfloor. \tag{4.8}$$

Příklad 6

Nechť platí rovnice 3.2, 4.1, 4.3, 4.5, 4.7, 4.8 a $m \leq 4$, pak lze setrojit tabulku 4.1. Z této tabulky je zřejmé, že pro každou chybu t jsou potřebné dvě paritní informace z kódového slova, aby bylo možné případnou chybu najít a opravit. Také je z tabulky patrné, že se zvyšujícím se m roste počet možností, jaký zvolit Reed-Solomonův kód v závislosti na požadovaném počtu symbolů k zabezpečení či na jeho korekční schopnosti.

Tabulka 4.1: Seznam Reed-Solomonových kódu a jeho vlastností splňující podmínku, kde $m \leq 4$.

m	n	k	t	r	c
[1]	[1]	[1]	[0]	[100.0 %]	[0.0 %]
2	3	1	1	33.3 %	33.3 %
3	7	5	1	71.4 %	14.3 %
3	7	3	2	42.9 %	28.6 %
3	7	1	3	14.3 %	42.9 %
4	15	13	1	86.7 %	6.7 %
4	15	11	2	73.3 %	13.3 %
4	15	9	3	60.0 %	20.0 %
4	15	7	4	46.7 %	26.7 %
4	15	7	5	33.3 %	33.3 %
4	15	3	6	20.0 %	40.0 %
4	15	1	7	6.7 %	46.7 %

Poznámka: Řádek uvedený v hranatých v závorkách v tabulce 4.1 reprezentuje RS kód, kde délka datové části je rovna délce celého kódového slova. Z toho vyplývá, že se zde nevyskytuje redundance, která by informaci zabezpečila, a proto $t = 0$. Z hlediska bezpečnosti nemá tento kód žádný praktický účinek. Nicméně je však možné sestrojit takovýto Reed-Solomonův kód [9].

Reed-Solomonovy kódy mají dvě důležité vlastnosti: [9]

- délka kódu je o jedna menší, než je velikost kódové abecedy,
- minimální vzdálenost je o jedna větší, než je počet paritních symbolů.

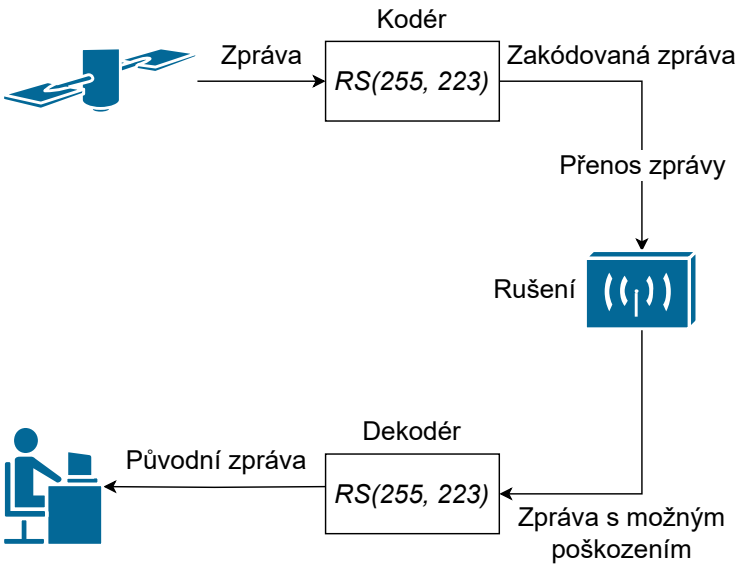
Kódy, jež mají minimální vzdálenost o jedna větší než počet paritních symbolů, se nazývají MDS kódy (anglicky *maximum distance separable codes*). Nejdůležitější třídu MDS kódů tvoří právě RS kódy [16].

Pro správnou funkčnost Reed-Solomonových kódů z hlediska zabezpečení je třeba existence jejich kodéru a dekodéru, viz příběh sondy Voyager z úvodní kapitoly.

Vstupem kodéru je zpráva, jež má být zakódována a doručena přenosovým kanálem k cílové destinaci. Jeho výstupem je zpráva s přidanou redundantní informací. Vstupní zprávu $M(X)$ lze zapsat jako polynom, kde jednotlivé koeficienty M_i polynomu odpovídají částem celkové zprávy, pro $M_i \in GF(p^m)$

$$M(X) = M_0 + M_1 X + \ldots M_{k-2} X^{k-2} + M_{k-1} X^{k-1}. \tag{4.9}$$

Vstupem dekodéru je výstupní kódové slovo z kodéru, jež mohlo být při přenosu po přenosovém médiu zatíženo rušením, které mohlo ve zprávě vytvořit jednu či více chyb. Chybou se rozumí nežádoucí změna hodnoty koeficientu kódového slova na jinou hodnotu. Výstupem dekodéru je původní zpráva $M(X)$. Není však pokaždé zaručeno, že se dekodéru podaří vždy obnovit zprávu do původní podoby, neboť poškozená zpráva mohla být zatížena větším počtem chyb, než je korekční schopnost Reed-Solomonova kódu, tím i schopnost dekodéru opravovat chyby. Názorný příklad principu přenosu zprávy za použití kodéru a dekodéru Reed-Solomonova kódů je na obrázku 4.1.



Obrázek 4.1: Příklad přenosu zprávy inspirovaný sondou Voyager

Detailnější popis kodérů lze nalézt v sekci 4.3 a dekodéru v sekci 5.1

Příklad 7

Na jednoduchém příkladu budou nastíněny vlastnosti Reed-Solomonových kódů, které doposud byly zmíněny. Slovo k zakódování bude slovo z úvodní kapitoly, respektive slovo „pilotovi“. Toto slovo se skládá z osmi písmen, která budou zabezpečena Reed-Solomonovým kódem. Pro názornost bude jednotlivým písmenům přiřazena určitá hodnota dle tabulky 4.2.

Tabulka 4.2: Tabulka znázorňující, jaká číselná hodnota náleží určitému písmenu.

1	2	3	4	5	6
P	I	L	O	T	V

Aplikací tabulky na slovo „pilotovi“ vznikne posloupnost čísel, jež lze zapsat jako vektor

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 6 & 2 \end{pmatrix}$$

nebo jako polynom z rovnice 4.9

$$M(X) = 1 + 2X + 3X^2 + 4X^3 + 5X^4 + 4X^5 + 6X^6 + 2X^7.$$

Jednotlivých Reed-Solomonových kódů může být nespočetně mnoho, proto je pro jejich lepší rozeznání zavedeno několik druhů jejich značení jako například $RS(n, k)$ nebo $RS(n, k, t)$. V této práci se bude nadále používat pouze značení $RS(n, k)$, jelikož se používá ve většině literatur.

Příklad 8

Dle popisu problému z příkladu 7 je patrné, že abeceda symbolů se skládá z šesti znaků a délka zprávy je 8 symbolů. Vycházíme z tabulky z příkladu 6. Z této tabulky problému nejvíce odpovídá řádek s délkou kódového slova 15 a délkou informační části 9, jelikož zpráva k zakódování $M(X)$ je délky 8. Jedná se tedy o kód $RS(15, 9)$ se schopností opravy tří symbolů. Nadále v práci bude však zvolen kód $RS(15, 8)$, jelikož tento kód více odpovídá požadavku na délku informační části kódového slova a zároveň zůstane zachována korekční schopnost. V tabulce 4.4 je sepsán výčet vlastností zvoleného $RS(15, 8)$, jež byly spočteny v tabulce 4.4.

Tabulka 4.3: Výčet prvků z tabulky 4.4

m	**n**	**k**	**t**	**r**	**c**
...	...	...	...	...	...
4	15	8	3	53.3 %	20.0 %
...	...	...	...	...	...

Tabulka 4.4: Výčet určitých vlastností $RS(15, 8)$

Délka kódového slova (rovnice 4.3)	$n = 15$ symbolů
Počet paritních symbolů (rovnice 4.4)	$n - k = 7$ symbolů
Informační délka zprávy (rovnice 4.5)	$k = 8$ symbolů
Hammingova vzdálenost (rovnice 4.6)	$d_{min} = 2t + 1 = 7$
Poměr zabezpečených symbolů (rovnice 3.2)	$r = \frac{k}{n} = 53,3\,\%$
Poměr chyb (rovnice 4.7)	$c = \frac{t}{n} = 20,0\,\%$
Počet prvků GF, viz (tabulka 2.1)	$q = n + 1 = 16$ symbolů
Počet možných kódových slov [9]	$p^{mk} = 2^{32} > 10^9$ slov
Počet možných přijatých slov [9]	$p^{mn} = 2^{60} > 10^{18}$ slov
Polynom generátoru tělesa	$F(X) = X^4 + X + 1$

4.2 Generující polynom

Nechť α je primitivní prvek $GF(q)$. Všechny kořeny generujícího polynomu $G(x)$ Reed-Solomonova kódu, jenž má korekční schopnost t, jsou $\alpha, \alpha^2, \ldots, \alpha^{2t}$. Nechť $\phi_i(X)$ je minimální polynom $\alpha^i \in GF(q)$, tak lze tento polynom zapsat jako $X - \alpha^i$. Generující polynom lze vypočítat jakou součin minimální polynomů následovně [16]

$$G(X) = \prod_{i=b}^{b+2t-1} (X + \alpha^i) = (X + \alpha)(X + \alpha^2) \ldots (X + \alpha^{b+2t-1}), \tag{4.10}$$

kde b je obvykle 0 nebo 1 [20]. Generující polynom je možné také zapsat jako

$$G(X) = \sum_{j=0}^{2t} g_j x^j = g_0 + g_1 X + \cdots + g_{2t-1} X^{2t-1} + X^{2t}, \tag{4.11}$$

kde $g_i \in GF(q)$ pro $0 \leq i < 2t$. Jelikož $\alpha, \alpha^2, \ldots, \alpha^{2t}$ jsou kořeny $X^n - 1$, tak lze $X^n - 1$ vydělit beze zbytku polynomem $G(X)$. Z tohoto důvodu je délka kódového slova $n = q - 1$ s délkou zabezpečovací části $2t$ [16].

Příklad 9

Nechť α je primitivní prvek tělesa $GF(2^5)$ z příkladu 1 a uvažujeme kód $RS(15, 8)$. Generující polynom $G(X)$ má kořeny $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$. Výsledný polynom lze za předpokladu $b = 1$ vypočítat následovně

$$\begin{aligned} G(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6) \\ &= a^6 + a^9 X + a^6 X^2 + a^4 X^3 + a^{14} X^4 + a^{10} X^5 + X^6 \end{aligned}$$

Při výpočtech se používají operační tabulky sčítání 2.2 a násobení 2.3 Galoisových těles, přičemž budou používány i v následujících příkladech.

4.3 Princip kódování

Nechť zabezpečovací část je $CK(X)$ a informační část je $M(X)$, tak lze $CK(X)$ získat za pomoci $M(X)$ při použití modulo operace společně s generujícím polynomem $G(X)$, respektive

$$CK(X) = X^{n-k}M(X) \bmod G(X) \tag{4.12}$$

nebo hodnotu $CK(X)$ lze získat také z

$$X^{n-k}M(X) = Q(X)G(X) + CK(X), \tag{4.13}$$

kde $Q(X)$ značí podíl a X^{n-k} reprezentuje posuv, aby kódové slovo bylo v systematickém tvaru, více viz podsekci 3.4.

Přičtením hodnoty $CK(X)$ k oběma stranám rovnice 4.13 vznikne

$$X^{n-k}M(X) + CK(X) = Q(X)G(X). \tag{4.14}$$

Kódové slovo $C(X)$, jež má být přeneseno, obsahuje v systematickém tvaru zabezpečovací část $CK(X)$ a informační část $M(X)$, kódové slovo lze tedy zapsat následovně

$$C(X) = X^{n-k}M(X) + CK(X), \tag{4.15}$$

kde stupeň polynomu $CK(X)$ je v intervalu $\langle 0, n-k-1\rangle \in \mathbb{N}$ a stupeň $X^{n-k}M(X)$ je v intervalu $\langle n-k, n-1\rangle \in \mathbb{N}$ pro $M(X) \neq 0$ jinak je stupeň 0. Pokud platí $n > k$, tak zabezpečovací část $CK(X)$ nikdy nepřekryje informační část $M(X)$. Pokud je stupeň $M(X)$ nulový, tak v takovémto případě bude celé kódové slovo rovno 0, respektive $C(X) = M(X) = CK(X) = 0$.

Kódové slovo $C(X)$ lze také zakódovat do nesystematického tvaru následovně

$$C(X)_{non-systematic} = G(X)M(X). \tag{4.16}$$

Na obrázku 4.2 je zobrazen diagram Reed-Solomonova kodéru, jehož vstupem je zpráva a výstupem je zakódovaná zpráva, respektive kódové slovo. Pro přehlednost jsou všechny polynomiální hodnoty kodéru, které jsou používány, rozepsané níže

- Informační část (zpráva) složená ze symbolů M_i, kde $i = 0, 1, \ldots, k-1$:

$$M(X) = M_0 + M_1X + \cdots + M_{k-2}X^{k-2} + M_{k-1}X^{k-1}$$

- Generující polynom $G(X)$ složený ze symbolů G_i, kde $i = 0, 1, \ldots, 2t-1$:

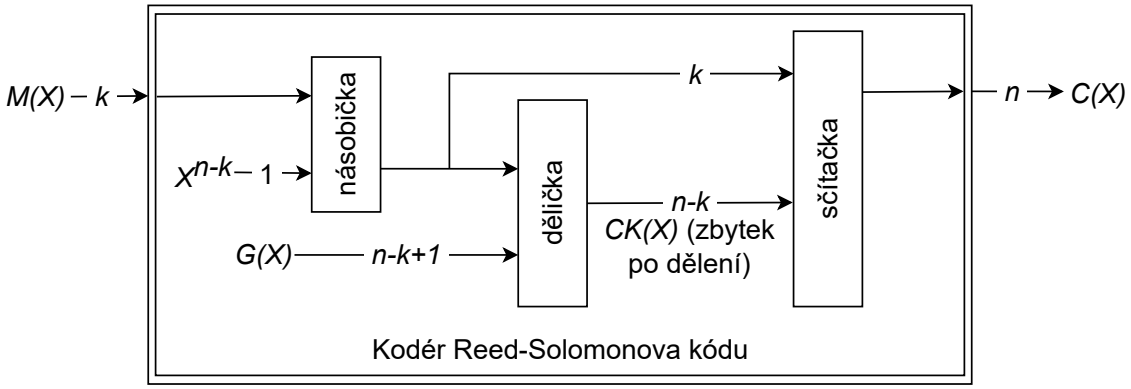
$$G(X) = G_0 + G_1X + \cdots + G_{2t-1}X^{2t-1} + X^{2t}$$

- Zabezpečovací část složená ze symbolů CK_i, kde $i = 0, 1, \ldots, n-k-1$:

$$\begin{aligned} CK(X) &= X^{n-k}M(X) \bmod G(X) \\ &= CK_0 + CK_1X + \cdots + CK_{n-k-2}X^{n-k-2} + CK_{n-k-1}X^{n-k-1} \end{aligned}$$

- Kódové slovo složené ze symbolů C_i, kde $i = 0, 1, \ldots, n-1$:

$$\begin{aligned} C(X) &= X^{n-k}M(X) + CK(X) \\ &= X^{n-k}M(X) + X^{n-k}M(X) \bmod G(X) \\ &= CK_0 + \cdots + CK_{n-k-1}X^{n-k-1} + M_0X^{n-k} + \cdots + M_{k-1}X^{n-1} \\ &= C_0 + C_1X + \cdots + C_{n-2}X^{n-2} + C_{n-1}X^{n-1} \end{aligned} \tag{4.17}$$



Obrázek 4.2: Diagram Reed-Solomonova kodéru. Diagram je převzatý z [9]

Příklad 10

Nechť máme zprávu $M(X) = 1 + 2X + 3X^2 + 4X^3 + 5X^4 + 4X^5 + 6X^6 + 2X^7$ z příkladu 7 a generující polynom $G(X) = a^6 + a^9X + a^6X^2 + a^4X^3 + a^{14}X^4 + a^{10}X^5 + X^6$ z příkladu 9. Nejdříve bude převedena zpráva $M(X)$ do exponenciální reprezentace tak, že jednotlivé koeficienty polynomu budou převedeny do binární podoby a za pomoci tabulky 2.1 se vyhledají jejich exponenciální reprezentace. Bude využit vektor M z příkladu 7:

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 6 & 2 \end{pmatrix} = \begin{pmatrix} 0001 & 0010 & 0011 & 0100 & 0101 & 0100 & 0110 & 0010 \end{pmatrix},$$

kde jednotlivým binárním hodnotám odpovídá vektor v exponenciálním zápisu

$$M = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^4 & \alpha^2 & \alpha^8 & \alpha^2 & \alpha^5 & \alpha^1 \end{pmatrix},$$

jenž ve polynomiálním zápisu vypadá

$$M(X) = \alpha^0 + \alpha^1X + \alpha^4X^2 + \alpha^2X^3 + \alpha^8X^4 + \alpha^2X^5 + \alpha^5X^6 + \alpha^1X^7$$

Výsledné kódové slovo $C(X)$ lze vypočítat za pomoci vzorce 4.15 následovně

$$C(X) = X^{n-k}M(X) + CK(X),$$

kde zabezpečovací část je

$$\begin{aligned} CK(X) &= (X^7)(\alpha^0 + \alpha^1X + \alpha^4X^2 + \alpha^2X^3 + \alpha^8X^4 + \alpha^2X^5 + \alpha^5X^6 + \alpha^1X^7) \bmod G(X) \\ &= \alpha^0X^7 + \alpha^1X^8 + \alpha^4X^9 + \alpha^2X^{10} + \alpha^8X^{11} + \alpha^2X^{12} + \alpha^5X^{13} + \alpha^1X^{14} \bmod \\ &\quad a^6 + a^9X + a^6X^2 + a^4X^3 + a^{14}X^4 + a^{10}X^5 + X^6 \\ &= a^8X + a^{14}X^2 + a^{11}X^3 + a^{14}X^4 + a^{13}X^5 + a^9X^6 \end{aligned}$$

a informační část je

$$\begin{aligned} X^{n-k}M(X) &= (X^7)(\alpha^0 + \alpha^1X + \alpha^4X^2 + \alpha^2X^3 + \alpha^8X^4 + \alpha^2X^5 + \alpha^5X^6 + \alpha^1X^7) \\ &= \alpha^0X^7 + \alpha^1X^8 + \alpha^4X^9 + \alpha^2X^{10} + \alpha^8X^{11} + \alpha^2X^{12} + \alpha^5X^{13} + \alpha^1X^{14}. \end{aligned}$$

Výsledné kódové slovo tedy vypadá následovně:

$$\begin{aligned} C(X) = {} & \alpha^8X + \alpha^{14}X^2 + \alpha^{11}X^3 + \alpha^{14}X^4 + \alpha^{13}X^5 + \alpha^9X^6 + \\ & \alpha^0X^7 + \alpha^1X^8 + \alpha^4X^9 + \alpha^2X^{10} + \alpha^8X^{11} + \alpha^2X^{12} + \alpha^5X^{13} + \alpha^1X^{14}. \end{aligned}$$

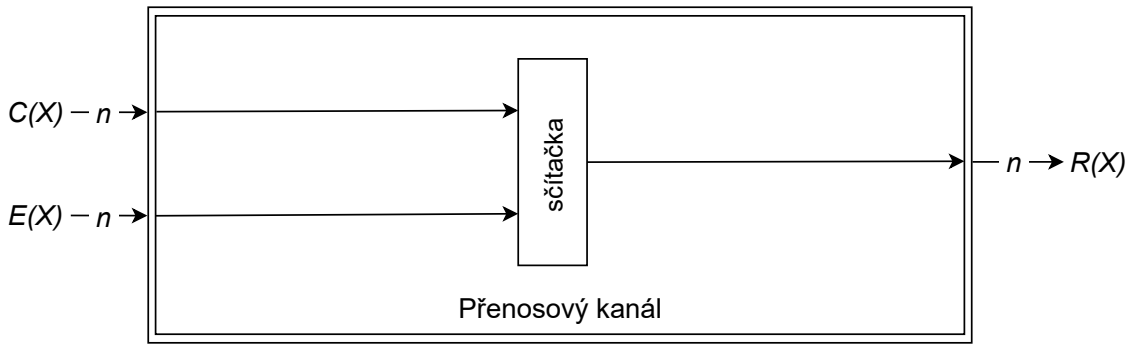
Kapitola 5

Dekódování Reed-Solomonových kódů

V předchozí kapitole bylo ukázáno, jak zakódovat zprávu pomocí Reed-Solomonova kódu. V této kapitole bude nejdříve do zakódovaného slova zanesena chyba a posléze budou ukázány techniky a postupy, jak toto poškozené slovo opravit zpět do původní podoby. Konkrétně bude popsán postup dekódování přijatého kódového slova. Bude popsán výpočet syndromů chyb a jak ze syndromů určit počet chyb nacházející se v přijatém slově. Následně bude popsána technika, jak nalézt pozice chyb a konkrétní hodnoty chyb s následnou opravou slova pomocí zjištěných informací. V závěru kapitoly budou popsány konkrétní algoritmy pro nalezení pozic chyb.

5.1 Princip dekódování

Před samotným dekódováním prochází kódové slovo tzv. přenosovým kanálem. V tomto kanálu mohou nastat na přenášeném kódovém slově $C(X)$ chyby, jež jsou reprezentovány polynomem $E(X)$. Tyto chyby mohly nastat jak na informační, tak i na zabezpečovací části kódového slova a dají vzniku kódovému slovu $R(X)$. Tento princip je znázorněn na modelu 5.1.



Obrázek 5.1: Model přenosového kanálu. Model je převzatý z [9]

Dekódovací proces je oproti kódovacímu značně složitější, rovněž je potřeba mít k dispozici výkonnější prostředky. Pro menší a méně výkonné Reed-Solomonovy kódy je snadné pracovat i při vysoké frekvenci přenášených dat (kódových slov). Toto ovšem neplatí pro

větší, tím pádem i výkonnější Reed-Solomonovy kódy, kdy je potřeba zajistit, aby i při vysokých přenosových rychlostech nedocházelo v komunikaci k dlouhému zpoždění. Samotný dekódovací proces lze rozdělit do pěti fází:

1. Vypočítat syndrom z přijatého slova.
2. Vypočítat lokalizační polynom chyby.
3. Vypočítat pozice chyb z lokalizačního polynomu.
4. Vypočítat původní hodnoty za pomoci syndromů a pozic chyb.
5. Vypočítat původní kódové slovo za pomoci pozic chyb a chybových hodnot.

Oproti binárním kódům, kdy stačí najít pouze pozici chyby a následně prohodit binární 0 za 1 nebo naopak, tak u nebinárních je navíc ještě potřeba vypočítat původní hodnotu. V dalších sekcích bude popsána fáze hledání syndromu chyby, lokalizačního polynomu a chybových hodnot. Ke konci kapitoly bude pomocí Petersonova-Gorensteinova-Zierleova algoritmu (PGZ) dekódováno slovo, jež bylo zakódováno v předchozí kapitole, jelikož je tento algoritmus a jeho výpočet jednodušší na pochopení. Složitější techniky hledání lokalizačního polynomu budou popsány posléze. Konkrétně bude popsán Berlekamp-Masseyho algoritmus (BMA) a Euklidův algoritmus. Tato sekce primárně vychází z [9].

Dekódovací proces lze zaznamenat pomocí diagramu, viz obrázek 5.2. Jednotlivé znázorněné polynomy společně s polynomy zobrazené na modelu přenosového kanálu na obrázku 5.1 jsou popsány dále jako

- Přijaté kódové slovo složené ze symbolů R_i, kde $i = 0, 1, \ldots, n-1$:

$$\begin{aligned} R(X) &= C(X) + E(X) \\ &= (C_0 + E_0) + (C_1 + E_1)X + \cdots + (C_{n-2} + E_{n-2})X^{n-2} + \\ &\quad (C_{n-1} + E_{n-1})X^{n-1} \\ &= R_0 + R_1 X + \cdots + R_{n-2}X^{n-2} + R_{n-1}X^{n-1} \end{aligned} \tag{5.1}$$

- Chybový vzor $E(X)$ je složený ze symbolů $X^0, X^1, \ldots, X^{n-1}$ reprezentující pozice chyb a ze symbolů $E_0, E_1, \ldots, E_{n-1}$ reprezentující hodnotu chyby [18]:

$$E(X) = E_0X^0 + E_1X^1 + \cdots + E_{n-1}X^{n-1}$$

- Dekódované kódové slovo $C(X)'$ složené ze symbolů C'_i, kde $i = 0, 1, \ldots, n-1$:

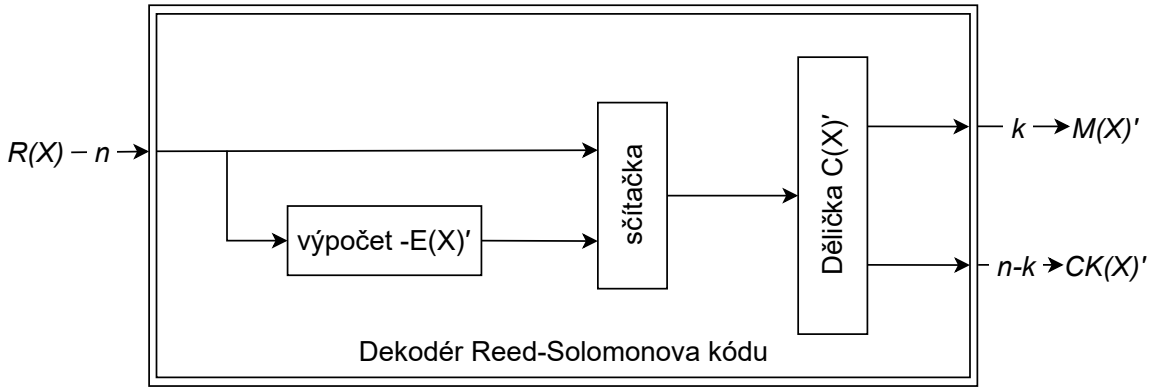
$$\begin{aligned} C(X)' &= R(X) - E(X) = R(X) + E(X) \\ &= X^{n-k}M(X)' + CK(X)' \\ &= X^{n-k}M(X)' + X^{n-k}M(X)' \bmod G(X) \\ &= C'_0 + C'_1X + \cdots + C'_{n-2}X^{n-2} + C'_{n-1}X^{n-1} \end{aligned}$$

- Dekódovaná zpráva $M(X)'$ obsahující symboly M'_i, kde $i = 0, 1, \ldots, k-1$:

$$\begin{aligned} M(X)' &= C'_{n-k} + C'_{n-k+1}X + \cdots + C'_{n-2}X^{k-2} + C'_{n-1}X^{k-1} \\ &= M'_0 + M'_1X + \cdots + M'_{k-2}X^{k-2} + M'_{k-1}X^{k-1} \end{aligned}$$

- Dekódovaná zabezpečovací část obsahující symboly CK'_i, kde $i = 0, 1, \ldots, n-k-1$:

$$\begin{aligned} CK(X)' &= C'_0 + C'_1 X + \cdots + C'_{n-k-2} X^{n-k-2} + C'_{n-k-1} X^{n-k-1} \\ &= CK'_0 + CK'_1 X + \cdots + CK'_{n-k-2} X^{n-k-2} + CK'_{n-k-1} X^{n-k-1} \end{aligned}$$



Obrázek 5.2: Diagram Reed-Solomonova dekodéru. Model je převzatý z [9]

5.2 Syndrom chyby

Slovo „syndrom“ je ve slovníku definováno jako skupina znaků a symptomů, které se vyskytují společně a charakterizují konkrétní abnormalitu nebo stav. Syndrom lze také definovat jako soubor souběžných věcí, které obvykle tvoří identifikovatelný vzorec[1]. Složky syndromu S_i jsou určité charakteristiky, které charakterizují konkrétní chybový vzor. Akumulované složky syndromu dohromady tvoří syndrom $S(X)$, kde $S_i = s(\alpha^i)$ a $i = 1, 2, \ldots, n-k$.

Nechť platí, že $R(X)$ je přijaté kódové slovo, tak jednotlivé složky lze vypočítat jako

$$S_i = R(\alpha^i). \tag{5.2}$$

Pokud kódové slovo $R(X)$ neobsahuje žádné chyby, tak jednotlivé složky syndromu S_i musí být nulové, jinak se ve slově $R(X)$ vyskytuje alespoň jedna chyba. Respektive platí, že kódové slovo $C(X)$ je násobkem generujícího polynomu $G(X)$, vizte sekci o systematickém formátu slova 3.4 a rovnici 4.14. Generující polynom má kořeny $\alpha, \alpha^2, \ldots, \alpha^{2t}$ a tudíž i kódové slovo $C(X)$ musí mít stejné kořeny. Postupným dosazením všech kořenů polynomu $G(X)$ do přijatého kódového slova $R(X) = C(X) + E(X)$ musí pokaždé vyjít nulová hodnota za předpokladu, že nenastala v přijatém slově žádná chyba. Jakákoliv chyba v přijatém slově má za následek, že výsledkem tohoto dosazování nebude pokaždé nulová hodnota.

[1] Definice byly převzaty z https://www.merriam-webster.com/dictionary/syndrome

Vzorec 5.2, pokud platí $j = 1, 2, \ldots, n-k$, lze upravit jako

$$S_j = C(\alpha^j) + E(\alpha^j), \tag{5.3}$$

respektive lze napsat

$$S_j = R(\alpha^j) = C(\alpha^j) + E(\alpha^j) = 0 + E(\alpha^j). \tag{5.4}$$

Lze tedy říci, že dosazením kořenů generátoru $G(X)$ jak do přeneseného slova $R(X)$ tak do chybového vzoru má za výsledek stejné složky syndromu S_i.

Rovnici 5.4 lze přepsat, pokud platí $j = 1, 2, \ldots, 2t$, jako [18]

$$S_j = R(\alpha^j) = E(\alpha^j) = \sum_{k=0}^{n-1} E_k \alpha^{jk}. \tag{5.5}$$

Jestliže platí, že přijaté slovo $R(X)$ obsahuje v chyb a jejich lokace jsou $i_1, i_2, \ldots, i_v$ s určitou nenulovou chybovou hodnotou E_{i_j}, a pokud $j = 1, 2, \ldots, 2t$, tak

$$S_j = \sum_{l=1}^{v} E_{i_l} (\alpha^j)^{i_l} = \sum_{l=1}^{v} E_{i_l} (\alpha^{i_l})^j. \tag{5.6}$$

Nechť

$$X_l = \alpha^{i_l}, \tag{5.7}$$

pak lze přepsat rovnice 5.6 jako

$$S_j = \sum_{l=1}^{v} E_{i_l} X_l^j. \tag{5.8}$$

Sekce primárně vychází z [23, 18].

Příklad 11

Nechť právě zakódované kódové slovo bez žádné chyby je dáno jako $C(X) = \alpha^8 X + \alpha^{14} X^2 + \alpha^{11} X^3 + \alpha^{14} X^4 + \alpha^{13} X^5 + \alpha^9 X^6 + \alpha^0 X^7 + \alpha^1 X^8 + \alpha^4 X^9 + \alpha^2 X^{10} + \alpha^8 X^{11} + \alpha^2 X^{12} + \alpha^5 X^{13} + \alpha^1 X^{14}$. Syndromy tohoto slova, jelikož nebylo zatíženo žádnou chybou, musí být nulové. Kořeny generátoru z příkladu 9 jsou $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$ a musí tedy platit, že dosazením každého jednoho kořene do $C(X)$ musí dát za pomoci rovnice 5.2 nulovou hodnotu.

$$S_1 = \alpha^9 + \alpha + \alpha^{14} + \alpha^3 + \alpha^3 + 1 + \alpha^7 + \alpha^9 + \alpha^{13} + \alpha^{12} + \alpha^4 + \alpha^{14} + \alpha^3 + 1 = 0$$
$$S_2 = \alpha^{10} + \alpha^3 + \alpha^2 + \alpha^7 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^2 + \alpha^7 + \alpha^7 + 1 + \alpha^{11} + \alpha + \alpha^{14} = 0$$
$$S_3 = \alpha^{11} + \alpha^5 + \alpha^5 + \alpha^{11} + \alpha^{13} + \alpha^{12} + \alpha^6 + \alpha^{10} + \alpha + \alpha^2 + \alpha^{11} + \alpha^8 + \alpha^{14} + \alpha^{13} = 0$$
$$S_4 = \alpha^{12} + \alpha^7 + \alpha^8 + 1 + \alpha^3 + \alpha^3 + \alpha^{13} + \alpha^3 + \alpha^{10} + \alpha^{12} + \alpha^7 + \alpha^5 + \alpha^{12} + \alpha^{12} = 0$$
$$S_5 = \alpha^{13} + \alpha^9 + \alpha^{11} + \alpha^4 + \alpha^8 + \alpha^9 + \alpha^5 + \alpha^{11} + \alpha^4 + \alpha^7 + \alpha^3 + \alpha^2 + \alpha^{10} + \alpha^{11} = 0$$
$$S_6 = \alpha^{14} + \alpha^{11} + \alpha^{14} + \alpha^8 + \alpha^{13} + 1 + \alpha^{12} + \alpha^4 + \alpha^{13} + \alpha^2 + \alpha^{14} + \alpha^{14} + \alpha^8 + \alpha^{10} = 0$$

Jelikož jsou všechny složky syndromu S_i nulové, tak toto slovo neobsahuje žádné chyby.

Příklad 12

Nechť v kódovém slově $C(X)$ z příkladu 11 vznikly chyby, jež jsou dány chybovým profilem $E(X)$ jako $E(X) = \alpha^6 X^2 + \alpha^{10} X^9$. Přijaté kódové $R(X)$ slovo na vstupu dekodéru je dle rovnice 5.1

$$\begin{aligned} R(X) &= C(X) + E(X) \\ R(X) &= (\alpha^8 X + \alpha^{14} X^2 + \alpha^{11} X^3 + \alpha^{14} X^4 + \alpha^{13} X^5 + \alpha^9 X^6 + X^7 + \alpha X^8 + \alpha^4 X^9 + \\ &\quad \alpha^2 X^{10} + \alpha^8 X^{11} + \alpha^2 X^{12} + \alpha^5 X^{13} + \alpha X^{14}) + (\alpha^6 X^2 + \alpha^{10} X^9) \\ &= \alpha^8 X + \alpha^8 X^2 + \alpha^{11} X^3 + \alpha^{14} X^4 + \alpha^{13} X^5 + \alpha^9 X^6 + X^7 + \alpha X^8 + \alpha^2 X^9 + \\ &\quad \alpha^2 X^{10} + \alpha^8 X^{11} + \alpha^2 X^{12} + \alpha^5 X^{13} + \alpha X^{14} \end{aligned}$$

Chyba v přijatém slově nastala, jak v informační části, tak i v zabezpečovací části. Konkrétně lze $E(X)$ přepsat do binárního tvaru jako $E = (1100X^2 \quad 0111X^9)$. Výsledné přijaté hodnoty, jež byly touto chybou postihnuty, jsou v binárním tvaru $0101X^2$ a $0100X^9$. Z tohoto je patrné, že chyby nastaly na více bitech, a to konkrétně na 4 bitech. Ačkoliv korekční schopnost kódu $RS(15, 8)$ je $t = 3$, tak chyby půjdou opravit, jelikož v Reed-Solomonových kódech se korekční schopnost uvažuje na úrovni symbolů nikoliv bitů (symbolem rozumíme množinu bitů).

Charakteristiky syndromů S_i přijatého kódového slova $R(X)$ jsou

$$\begin{aligned} S_1 &= \alpha^9 + \alpha^{10} + \alpha^{14} + \alpha^3 + \alpha^3 + 1 + \alpha^7 + \alpha^9 + \alpha^{11} + \alpha^{12} + \alpha^4 + \alpha^{14} + \alpha^3 + 1 = \alpha^5 \\ S_2 &= \alpha^{10} + \alpha^{12} + \alpha^2 + \alpha^7 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^2 + \alpha^5 + \alpha^7 + 1 + \alpha^{11} + \alpha^1 + \alpha^{14} = \alpha^9 \\ S_3 &= \alpha^{11} + \alpha^{14} + \alpha^5 + \alpha^{11} + \alpha^{13} + \alpha^{12} + \alpha^6 + \alpha^{10} + \alpha^{14} + \alpha^2 + \alpha^{11} + \alpha^8 + \alpha^{14} + \alpha^{13} = \alpha^2 \\ S_4 &= \alpha^{12} + \alpha + \alpha^8 + 1 + \alpha^3 + \alpha^3 + \alpha^{13} + \alpha^3 + \alpha^8 + \alpha^{12} + \alpha^7 + \alpha^5 + \alpha^{12} + \alpha^{12} = \alpha^7 \\ S_5 &= \alpha^{13} + \alpha^3 + \alpha^{11} + \alpha^4 + \alpha^8 + \alpha^9 + \alpha^5 + \alpha^{11} + \alpha^2 + \alpha^7 + \alpha^3 + \alpha^2 + \alpha^{10} + \alpha^{11} = \alpha^8 \\ S_6 &= \alpha^{14} + \alpha^5 + \alpha^{14} + \alpha^8 + \alpha^{13} + 1 + \alpha^{12} + \alpha^4 + \alpha^{11} + \alpha^2 + \alpha^{14} + \alpha^{14} + \alpha^8 + \alpha^{10} = \alpha^7 \end{aligned}$$

Z rovnice 5.4 by mělo platit, že jednotlivé charakteristiky syndromu S_i by měly být shodné s chybovým vzorem E_i po dosazení kořenů generujícího polynomu $G(X)$.

$$\begin{aligned} E_1 &= \alpha^8 + \alpha^4 = \alpha^5 \\ E_2 &= \alpha^{10} + \alpha^{13} = \alpha^9 \\ E_3 &= \alpha^{12} + \alpha^7 = \alpha^2 \\ E_4 &= \alpha^{14} + \alpha^1 = \alpha^7 \\ E_5 &= \alpha^1 + \alpha^{10} = \alpha^8 \\ E_6 &= \alpha^3 + \alpha^4 = \alpha^7 \end{aligned}$$

Z výsledků je patrné, že $S_i = E_i$ kde $i = 1, 2, \ldots, n - k$ a vskutku jsou jednotlivé charakteristiky syndromu závislé na chybovém profilu, jenž do kódového slova zanesl chyby.

5.3 Výpočet počtu chyb

Sekce se bude zabývat technikou, jak pomocí syndromů zjistit kolik se v přijatém slově nachází chyb. Tato informace následně pomůže při hledání lokalizačního polynomu, jenž bude vysvětlen v následující sekci. Konkrétně se v této sekci bude pojednávat o případech, kdy kódové slovo obsahuje jednu, dvě nebo tři chyby. Tento princip lze aplikovat i na vícenásobnější chyby, kde se však stávají výpočty náročnější, a proto existují pokročilejší principy, jak opravit chyby v kódovém slově. Tyto principy budou vysvětleny v sekcích 5.7 a 5.8. Tato sekce a její podsekce primárně vychází z literatury [22].

Dle [23] lze za použití autoregresní modelovací techniky vytvořit matici, jenž umožňuje za pomoci prvních v syndromů předpovědět následující syndrom. Tuto syndromovou matici lze zapsat jako

$$\begin{bmatrix} S_1 & S_2 & S_3 & \dots & S_{v-1} & S_v \\ S_2 & S_3 & S_4 & \dots & S_t & S_{v+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{v-1} & S_v & S_{v+1} & \dots & S_{2v-3} & S_{2v-2} \\ S_v & S_{v+1} & S_{v+2} & \dots & S_{2v-2} & S_{2v-1} \end{bmatrix} \begin{bmatrix} L_v \\ L_{v-1} \\ \vdots \\ L_2 \\ L_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v-1} \\ -S_{2v} \end{bmatrix}, \tag{5.9}$$

kde hodnoty $L_1, L_2, \dots, L_v$ reprezentují koeficienty lokalizačního polynomu, více v sekci 5.4.

5.3.1 Jednochyba

Uvažujeme jednochybu v přijatém kódovém slově a počet chyb $v = 1$. Rovnici 5.8 lze přepsat za předpokladu, že $j = 1, 2, \dots, 2t$ jako

$$S_1 = E_1 X_1 \quad S_2 = E_1 X_1^2 \quad \dots \quad S_j = E_1 X_1^j. \tag{5.10}$$

Jedná se vlastně o geometrickou posloupnost s kvocientem X_1. Hodnoty E_1 a X_1 jsou tedy

$$X_1 = \frac{S_2}{S_1} \qquad E_1 = \frac{S_1}{X_1}. \tag{5.11}$$

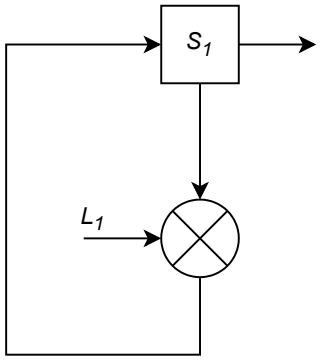
Z informací výše je patrné, že v případě jedné chyby existuje hodnota L_1, jež odpovídá kvocientu geometrické řady. Je možné zapsat následující rovnici za předpokladu $j = 1, 2, \dots, 2t - 1$

$$S_{j+1} = S_j L_1. \tag{5.12}$$

Vyjádřením vznikne $2t - 1$ rovnic s jednou neznámou L_1 nebo dle rovnice 5.9 vznikne jedna rovnice o jedné neznámé

$$S_1 L_1 = S_2.$$

Na obrázku 5.3 je znázorněn obvod pro výpočet charakteristik syndromů za pomoci lineárního zpětnovazebního registru (anglicky *Linear FeedBack Shift Register* – LFSR). Pomocí něho lze vypočítat všechny potřebné charakteristiky syndromu.



Obrázek 5.3: Obvod pro výpočet syndromů s jednochybou za pomoci LFSR [23].

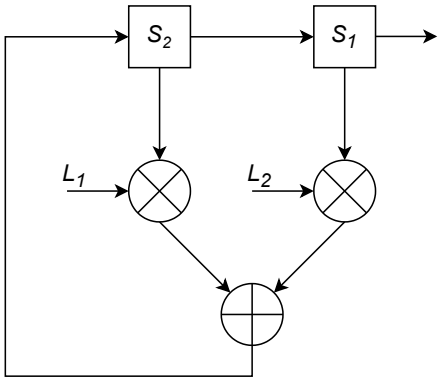
5.3.2 Dvojchyba

Uvažujeme dvojchybu v přijatém kódovém slově, respektive $v = 2$. Dle rovnice 5.9 vznikne soustava rovnic o dvou neznámých

$$S_1 L_2 + S_2 L_1 = S_3$$
$$S_2 L_2 + S_3 L_1 = S_4,$$

kde musí být nenulový determinant $\Delta_2 = \begin{vmatrix} S_1 & S_2 \\ S_2 & S_3 \end{vmatrix}$. Pokud vyjde determinant nenulový je jisté, že se v přijatém slově nachází více než jedna chyba. Pokud by se ve slově nacházela pouze jedna chyba, tak by determinant vyšel nulový, jelikož by byl druhý sloupec determinantu násobkem prvního.

Na obrázku 5.4 je znázorněn obvod pomocí něhož lze vypočítat všechny potřebné charakteristiky syndromu.



Obrázek 5.4: Obvod pro výpočet syndromů s dvojchybou za pomoci LFSR [23].

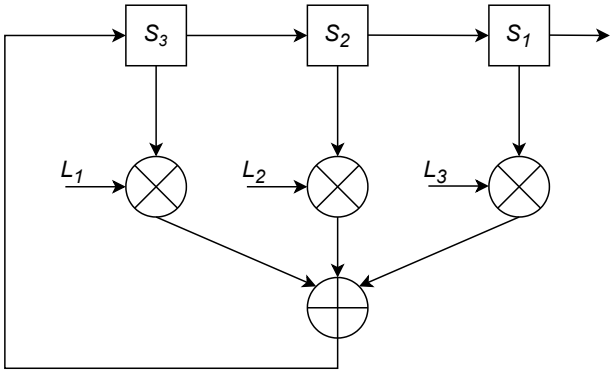
5.3.3 Trojchyba

Pokud se v přijatém slově nachází trojchyba, respektive $v = 3$, tak dle 5.9 lze rozepsat soustavu rovnic o třech neznámých takto

$$\begin{aligned} S_1L_3 + S_2L_2 + S_3L_1 &= S_4 \\ S_2L_3 + S_3L_2 + S_4L_1 &= S_5 \\ S_3L_3 + S_4L_2 + S_5L_1 &= S_6, \end{aligned}$$

kde musí být nenulový determinant $\Delta_3 = \begin{vmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{vmatrix}$. Pokud by se ve slově vyskytovaly pouze dvě chyby, tak by determinant vyšel nulový, jelikož by byl třetí sloupec lineární kombinací prvních dvou sloupců.

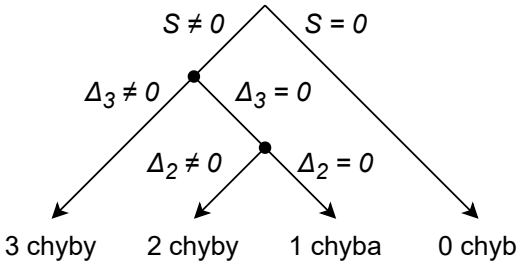
Na obrázku 5.5 je znázorněn obvod, pomocí něhož lze vypočítat všechny potřebné charakteristiky syndromu, respektive S_4, S_5 a S_6.



Obrázek 5.5: Obvod pro výpočet syndromů s trojchybou za pomoci LFSR [23].

Obrázek 5.6 znázorňuje, jak by měl vypadat výpočet počtu chyb při maximální korekční schopnosti kódů $t = 3$. Nejdříve se spočítají syndromy S, pokud je alespoň jeden nenulový, tak se dále spočítá Δ_3, pokud $\Delta_3 = 0$, tak se spočítá $\Delta_2, \ldots$.

Musí se pokaždé začínat od kořene stromu, protože pokud by se například nejdříve spočítala Δ_2 při počtu chyb $v = 3$, tak může být výsledkem, jak $\Delta_2 = 0$, tak i $\Delta_2 \neq 0$. První výsledek evokuje to, že se ve slově nachází pouze jedna chyba. Zatímco pokud výsledek vyjde nenulový, tak se mylně může předpokládat, že ve slově jsou právě dvě chyby místo tří.



Obrázek 5.6: Rozhodovací strom pro maximální korekční schopnost $t = 3$ symboly [23].

5.4 Lokalizační polynom

Nechť platí, že pozice chyb v kódovém slově jsou na pozicích $X^{j_1}, X^{j_2}, \ldots, X^{j_v}$ za předpokladu v chyb, tak chybový polynom může být zapsán jako

$$E(X) = E_{j_1} X^{j_1} + E_{j_2} X^{j_2} + \cdots + E_{j_v} X^{j_v}, \tag{5.13}$$

kde indexy $1, 2, \ldots, v$ určují pořadí chyby a j odpovídá lokaci chyby. K opravě poškozeného slova je potřeba vypočítat chybové hodnoty E_{j_l} a pozice chyb X^{j_l}, kde $l = 1, 2, \ldots, v$.

Nechť platí, že pozice chyby je definována jako $\beta_l = \alpha^{j_l}$. Rozepsáním rovnice 5.8 se získá $2t$ charakteristik syndromů

$$\begin{aligned} S_1 &= R(\alpha) = E_{j_1}\beta_1 + E_{j_2}\beta_2 + \cdots + E_{j_v}\beta_v \\ S_2 &= R(\alpha^2) = E_{j_1}\beta_1^2 + E_{j_2}\beta_2^2 + \cdots + E_{j_v}\beta_v^2 \\ &\ldots \\ S_{2t} &= R(\alpha^{2t}) = E_{j_1}\beta_1^{2t} + E_{j_2}\beta_2^{2t} + \cdots + E_{j_v}\beta_v^{2t} \end{aligned} \tag{5.14}$$

V této soustavě se nachází $2t$ neznámých. Konkrétně t pozic chyb a t chybových hodnot. Jelikož je tato sada rovnic nelineární, protože některé neznámé disponují exponenty, tak tyto rovnice nelze řešit obvyklým způsobem. Postupem, kterým lze vyřešit tuto sadu rovnic, se obecně říká Reed-Solomonoův dekódovací algoritmus.

Lokalizační polynom (anglicky *error-locator polynomial*) lze zapsat jako

$$L(X) = (1 + \beta_1 X)(1 + \beta_2 X) \ldots (1 + \beta_v X) \tag{5.15}$$

$$= 1 + L_1 X + L_2 X^2 + \cdots + L_v X^v, \tag{5.16}$$

kde kořeny polynomu jsou $\frac{1}{\beta_1}, \frac{1}{\beta_2}, \ldots, \frac{1}{\beta_v}$. Převrácené hodnoty těchto kořenů určují pozici chyby v $E(X)$. Nechť je rovnice 5.9 přepsána zjednodušeně jako

$$S \cdot L = S_t. \tag{5.17}$$

Pro sestavení lokalizačního polynomu je potřeba vypočítat všechny jeho koeficienty $L_1, L_2, \ldots, L_v$. Matici L s těmito koeficienty lze vypočítat za pomoci inverzní matice S^{-1}, jež lze vypočítat dle [12] za pomoci adjungované matice S^* jako

$$S^{-1} = \frac{1}{|S|} S^*. \tag{5.18}$$

Následně dle pravidel lze upravit rovnici 5.17 jako

$$S^{-1} \cdot S \cdot L = S^{-1} \cdot S_t \tag{5.19}$$

$$L = S^{-1} \cdot S_t. \tag{5.20}$$

Z jednotlivých prvků matice L lze vyjádřit koeficienty lokalizačního polynomu $L(X)$. Sekce primárně vychází z literatury [23].

Příklad 13
Nechť platí charakteristiky syndromu z příkladu 12

$$S_1 = \alpha^5 \quad S_2 = \alpha^9 \quad S_3 = \alpha^2 \quad S_4 = \alpha^7 \quad S_5 = \alpha^8 \quad S_6 = \alpha^7.$$

Dle rovnice 5.17 lze sestavit syndromovou matici S_3 s předpokládaným počtem chyb $v = t = 3$

$$S_3 = \begin{bmatrix} \alpha^5 & \alpha^9 & \alpha^2 \\ \alpha^9 & \alpha^2 & \alpha^7 \\ \alpha^2 & \alpha^9 & \alpha^8 \end{bmatrix}$$

Vypočteme determinant $|S_{\Delta_3}|$ této matice

$$|S_{\Delta_3}| = \begin{bmatrix} \alpha^5 & \alpha^9 & \alpha^2 \\ \alpha^9 & \alpha^2 & \alpha^7 \\ \alpha^2 & \alpha^7 & \alpha^8 \end{bmatrix} = \alpha^0 + \alpha^3 + \alpha^3 - \alpha^6 - \alpha^4 - \alpha^{11} = 0$$

Jelikož determinant vyšel nulový, tak bylo zjištěno, že se ve slově nevyskytují tři chyby. Nyní se sestaví syndromová matice S_2 s předpokládaným počtem chyb $v = 2$.

$$S_2 = \begin{bmatrix} \alpha^5 & \alpha^9 \\ \alpha^9 & \alpha^2 \end{bmatrix}$$

Determinant této matice $|S_{\Delta_2}|$ je

$$|S_{\Delta_2}| = \begin{bmatrix} \alpha^5 & \alpha^9 \\ \alpha^9 & \alpha^2 \end{bmatrix} = \alpha^7 - \alpha^3 = \alpha^4 \neq 0.$$

Determinant není nulový, tudíž se ve slově vyskytují právě dvě chyby. Nyní je potřeba vypočítat inverzní matici S^{-1} dle rovnice 5.18

$$S^{-1} = \frac{1}{\alpha^4} \begin{bmatrix} \alpha^2 & \alpha^9 \\ \alpha^9 & \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^{13} & \alpha^5 \\ \alpha^5 & \alpha^1 \end{bmatrix}.$$

Nyní je možné aplikací rovnic 5.19 a 5.20 získat matici L s koeficienty lokalizačního polynomu

$$\begin{bmatrix} \alpha^{13} & \alpha^5 \\ \alpha^5 & \alpha^1 \end{bmatrix} \begin{bmatrix} \alpha^5 & \alpha^9 \\ \alpha^9 & \alpha^2 \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} \alpha^{13} & \alpha^5 \\ \alpha^5 & \alpha^1 \end{bmatrix} \begin{bmatrix} \alpha^2 \\ \alpha^7 \end{bmatrix}$$
$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} \alpha^{11} \\ \alpha^{11} \end{bmatrix}$$

Lokalizační polynom dle matice L lze zapsat jako

$$L(X) = 1 + \alpha^{11}X + \alpha^{11}X^2.$$

Dalším krokem bude výpočet kořenů tohoto polynomu.

5.5 Chienovo vyhledávání

Nechť existuje lokalizační polynom $L(X)$. Dalším krokem dekódování je nalezení kořenů $L(X)$. To se provede tak, že se prozkoumá každý prvek konečného tělesa $GF(q^m)$, zda není kořenem $L(X)$. Z hlediska dekódování se tento algoritmus jeví jako nejvíce efektivní.

Vyhodnocením každého nenulového prvku konečného tělesa:
$x = 1, x = \alpha, x = \alpha^2, \ldots, x = \alpha^{q^m-2}$ vznikne

$$\begin{aligned} L(1) &= 1 + L_1(1) + L_2(1)^2 + \cdots + L_v(1)^v \\ L(\alpha) &= 1 + L_1(\alpha) + L_2(\alpha)^2 + \cdots + L_v(\alpha)^v \\ L(\alpha^2) &= 1 + L_1(\alpha^2) + L_2(\alpha^2)^2 + \cdots + L_v(\alpha^2)^v \\ &\vdots \\ L(\alpha^{q^m-2}) &= 1 + L_1(\alpha^{q^m-2}) + L_2(\alpha^{q^m-2})^2 + \cdots + L_v(\alpha^{q^m-2})^v. \end{aligned} \tag{5.21}$$

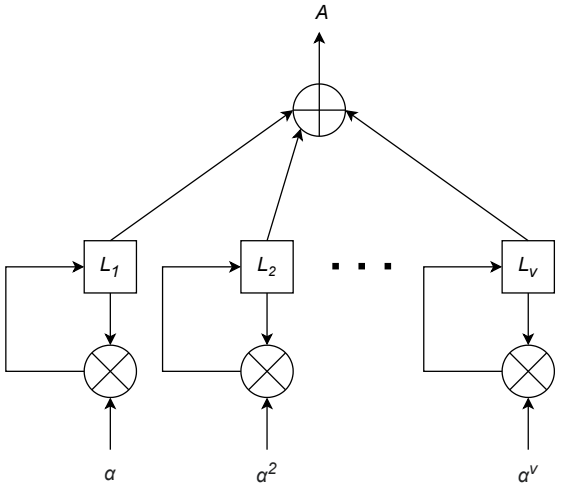
Tento algoritmus může být efektivně realizován pomocí hardwaru. Příklad obvodu, jenž uskutečňuje Chienovo vyhledávání, je zobrazen na obrázku 5.7. Sada registrů je prvně naplněna koeficienty lokalizačního polynomu $L_1, L_2, \ldots, L_v$. Zprvu je výstupem

$$A = \sum_{j=1}^{v} L_j = L(1) - 1.$$

Pokud $A = 1$, tak byla nalezena pozice chyby, protože $L(X) = 0$. V následujících fázích je stav každého registru vynásoben $\alpha, \alpha^2, \ldots, \alpha^v$. V registru se tedy nachází hodnota $L_1\alpha, L_2\alpha^2, \ldots, L_v\alpha^v$. Výstupem je

$$A = \sum_{j=1}^{v} L_j\alpha^j = L(\alpha) - 1.$$

Obdobně se pokračuje, dokud se takto nezkontrolují všechny nenulové prvky konečného tělesa. Sekce vychází z literatury [18].



Obrázek 5.7: Příklad obvodu realizující Chienovo vyhledávání z [18].

Příklad 14
Nechť platí lokalizační polynom $L(X)$ z příkladu 13

$$L(X) = 1 + \alpha^{11}X + \alpha^{11}X^2.$$

Konečné těleso má dle příkladu 1 všechny nenulové prvky $\alpha^0, \alpha^1, \ldots, \alpha^{14}$. Všechny tyto prvky se dosadí do 5.21 a pokud vyjde nějaká z rovnic nulová, tak se jedná o kořen lokalizačního polynomu $L(X)$.

$$\begin{aligned}
L(\alpha^0) &= \alpha^0 + \alpha^{11} + \alpha^{11} = \alpha^0 \\
L(\alpha^1) &= \alpha^0 + \alpha^{12} + \alpha^{13} = \alpha^4 \\
L(\alpha^2) &= \alpha^0 + \alpha^{13} + \alpha^0 = \alpha^{13} \\
L(\alpha^3) &= \alpha^0 + \alpha^{14} + \alpha^2 = \alpha^6 \\
L(\alpha^4) &= \alpha^0 + \alpha^0 + \alpha^4 = \alpha^4 \\
L(\alpha^5) &= \alpha^0 + \alpha^1 + \alpha^6 = \alpha^{12} \\
L(\alpha^6) &= \alpha^0 + \alpha^2 + \alpha^8 = 0 \\
L(\alpha^7) &= \alpha^0 + \alpha^3 + \alpha^{10} = \alpha^{11} \\
L(\alpha^8) &= \alpha^0 + \alpha^4 + \alpha^{12} = \alpha^{13} \\
L(\alpha^9) &= \alpha^0 + \alpha^5 + \alpha^{14} = \alpha^{11} \\
L(\alpha^{10}) &= \alpha^0 + \alpha^6 + \alpha^1 = \alpha^{12} \\
L(\alpha^{11}) &= \alpha^0 + \alpha^7 + \alpha^3 = \alpha^1 \\
L(\alpha^{12}) &= \alpha^0 + \alpha^8 + \alpha^5 = \alpha^1 \\
L(\alpha^{13}) &= \alpha^0 + \alpha^9 + \alpha^7 = 0 \\
L(\alpha^{14}) &= \alpha^0 + \alpha^{10} + \alpha^9 = \alpha^6
\end{aligned}$$

Kořeny lokalizačního polynomu tedy jsou α^6 a α^{13}. Dle 5.15 určují převrácené hodnoty kořenů pozice chyb, respektive zde platí, že pozice chyb jsou $\alpha^6 = \frac{1}{\beta_1}$ a $\alpha^{13} = \frac{1}{\beta_2}$. Za předpokladu $\beta_l = \alpha^{j_l}$ lze pozice chyb zapsat jako

$$\beta_1 = \alpha^{j_1} = \alpha^9 \qquad \beta_2 = \alpha^{j_2} = \alpha^2.$$

5.6 Výpočet chybové hodnoty a oprava přijatého slova

V této sekci budou popsány dva postupy pro nalezení chybových hodnot z lokalizačního polynomu $L(X)$ a jeho kořenů β_l. Následně bude popsáno, jak z chybového polynomu a přijatého slova získat původní slovo. Princip opravy přijatého kódového slova popsaného v předchozích sekcích a z následující sekce vychází z PGZ algoritmu. Postup vyhodnocování tohoto algoritmu je znázorněn na vývojovém diagramu 5.8.

5.6.1 Maticový výpočet chybové hodnoty

Sadu rovnic 5.14 lze přepsat do maticové podoby jako [18]

$$\begin{bmatrix} \beta_1 & \beta_2 & \dots & \beta_v \\ \beta_1^2 & \beta_2^2 & \dots & \beta_v^2 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{2t} & \beta_2^{2t} & \dots & \beta_v^{2t} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_v \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix}. \tag{5.22}$$

Počet charakteristik syndromů je $2t$. Jelikož však už byl zjištěn počet chyb v přijatém kódovém slově v a všechny jejich lokace β_l, tak je možné vypočítat chybové hodnoty za pomoci v rovnic, proto je potřeba vypočítat pouze v neznámých [23]. Rovnici 5.22 lze dle těchto skutečností přepsat jako

$$\begin{bmatrix} \beta_1 & \beta_2 & \dots & \beta_v \\ \beta_1^2 & \beta_2^2 & \dots & \beta_v^2 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{2t} & \beta_2^{2t} & \dots & \beta_v^{2t} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_v \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_v \end{bmatrix}. \tag{5.23}$$

Vypočítáním jednotlivých hodnot $E_1, E_2, \dots, E_V$, tak lze výsledný chybový polynom vyjádřit jako

$$\hat{E}(X) = E(X) = E_1\beta_1 + E_2\beta_2 + \dots + E_v\beta_v. \tag{5.24}$$

Výpočet koeficientů lze provést obdobně jako u lokalizačního polynomu pomocí inverzní matice β^{-1} jako

$$E = \beta^{-1} \cdot S_v, \tag{5.25}$$

jestliže platí, že rovnice 5.23 je zjednodušeně zapsána jako

$$\beta \cdot E = S_v. \tag{5.26}$$

Podsekce primárně vychází z literatury [23].

Příklad 15

Nechť platí kořeny $\beta_1 = \alpha^{j_1} = \alpha^9$ a $\beta_2 = \alpha^{j_2} = \alpha^2$ z příkladu 14 a charakteristiky syndromu $S_1 = \alpha^5$ a $S_2 = \alpha^9$ z příkladu 12. Dle rovnice 5.23 lze zapsat dané hodnoty, při použití značení z rovnice 5.26, jako

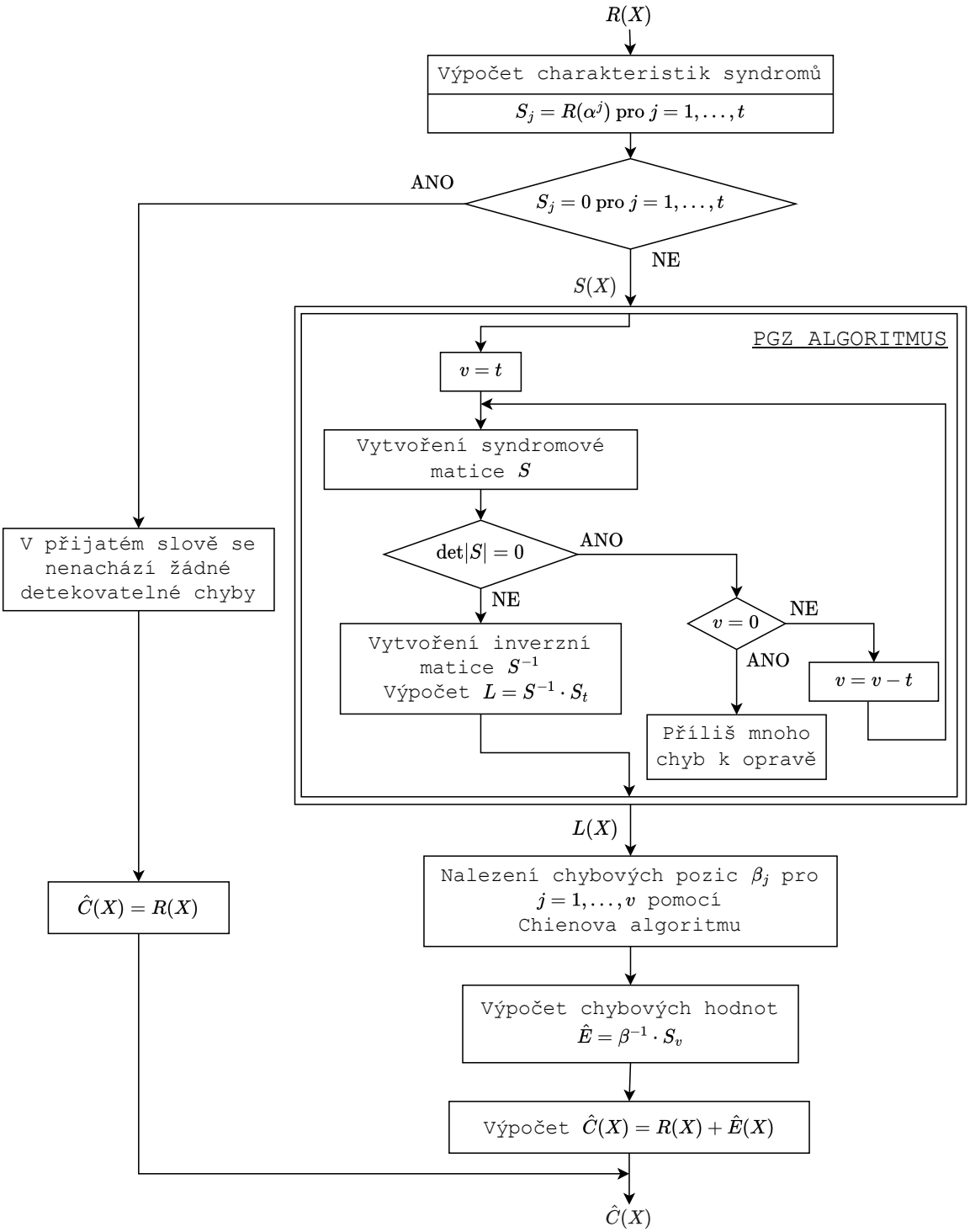
$$\begin{bmatrix} \alpha^9 & \alpha^2 \\ \alpha^3 & \alpha^4 \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} \alpha^5 \\ \alpha^9 \end{bmatrix}$$

Obdobným postupem jako v příkladu 13 lze vypočítat matici E následovně

$$E = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} \alpha^{12} & \alpha^{10} \\ \alpha^{11} & \alpha^2 \end{bmatrix} \begin{bmatrix} \alpha^5 \\ \alpha^9 \end{bmatrix} = \begin{bmatrix} \alpha^{10} \\ \alpha^6 \end{bmatrix}$$

Chybový polynom lze dle rovnice 5.13 zapsat jako

$$\hat{E}(X) = \alpha^6 X^2 + \alpha^{10} X^9.$$



Obrázek 5.8: Vývojový diagram Petersonova-Gorensteinova-Zierleova algoritmu. Obrázek byl převzat s úpravami z [14].

5.6.2 Forneyův algoritmus

V předchozí podsekci byla popsána metoda hledání koeficientů chybového polynomu pomocí lineárních rovnic. Nicméně existuje metoda, jež je výpočetně jednodušší.

Nechť platí matice z rovnice 5.22. Tato matice je vlastně maticí Vandermondových polynomů, jež lze rychle řešit pomocí Forneyho algoritmu. Nechť je definován syndromový polynom jako

$$S(X) = 1 + S_1X + S_2X^2 + \cdots + S_{2t}X^{2t} = 1 + \sum_{j=1}^{2t} S_jX^j. \tag{5.27}$$

Vyhodnocovač chyb (anglicky *error-value evaluator*) je definován jako

$$\Omega(X) = S(X)L(X) \bmod X^{2t+1}. \tag{5.28}$$

Tato rovnice se nazývá klíčovou rovnicí. Podstatným aspektem této rovnice je to, že díky modulo operaci dojde k zahození všech členů s řádem $2t$ či vyšším.

Jednotlivé chybové hodnoty lze vypočítat jako

$$E_l = \frac{(\beta_l)^{2-b}\,\Omega(\beta_l^{-1})}{L'(\beta_l^{-1})}, \tag{5.29}$$

kde $L'(X)$ značí derivaci $L(X)$ a hodnota b je definována v sekci 4.2. Podsekce vychází z literatury [20, 18].

Příklad 16

Nechť platí lokalizační polynom z příkladu 13 $L(X)$ a pozice chyb $\beta_1 a \beta_2$ z příkladu 14. Chybový polynom lze spočítat pomocí Forneyova algoritmu následovně. Předpokládáme charakteristiky syndromu

$$S_1 = \alpha^5 \quad S_2 = \alpha^9 \quad S_3 = \alpha^2 \quad S_4 = \alpha^7 \quad S_5 = \alpha^8 \quad S_6 = \alpha^7.$$

Syndromový polynom lze z těchto charakteristik vypočítat dle rovnice 5.27 jako

$$S(X) = 1 + \alpha^5X + \alpha^9X^2 + \alpha^2X^3 + \alpha^7X^4 + \alpha^8X^5 + \alpha^7X^6.$$

Následně se vypočítá vyhodnocovač chyb z rovnice 5.28

$$\begin{aligned}\Omega(X) &= (1 + \alpha^{11}X + \alpha^{11}X^2)(1 + \alpha^5X + \alpha^9X^2 + \alpha^2X^3 + \alpha^7X^4 + \alpha^8X^5 + \alpha^7X^6) \bmod X^6 \\ &= 1 + \alpha^3X + \alpha^5X^2.\end{aligned}$$

Derivace lokalizačního polynomu je $L'(X) = \alpha^{11}$. Nyní se pro jednotlivé pozice chyb za pomoci rovnice 5.29 zjistí jejich chybové hodnoty za předpokladu $b = 1$ z příkladu 9

$$E_1(\alpha^9) = \frac{\alpha^9(1 + \alpha^3\alpha^6 + \alpha^5(\alpha^6)^2)}{\alpha^{11}} = \frac{\alpha^9(1 + \alpha^9 + \alpha^2)}{\alpha^{11}} = \alpha^6\alpha^4 = \alpha^{10},$$

$$E_2(\alpha^2) = \frac{\alpha^2(1 + \alpha^3\alpha^{13} + \alpha^5(\alpha^{13})^2)}{\alpha^{11}} = \frac{\alpha^2(1 + \alpha + \alpha)}{\alpha^{11}} = \alpha^2\alpha^4 = \alpha^6.$$

Chybový polynom lze dle rovnice 5.13 zapsat jako

$$\hat{E}(X) = \alpha^6X^2 + \alpha^{10}X^9,$$

kde je tento polynom shodný s polynomem vypočítaným pomocí maticových operací v příkladu 15.

5.6.3 Oprava chyb

Nechť přijaté slovo $R(X)$ je zatíženo chybovým vzorem $E(X)$ a je dán chybový polynom $\hat{E}(X)$, jenž byl vypočten pomocí principů popsaných výše. Opravu přijatého slova lze provést jako

$$\hat{C}(X) = R(X) + \hat{E}(X) = C(X) + E(X) + \hat{E}(X), \tag{5.30}$$

kde $\hat{C}(X)$ je opravené kódové slovo.

Příklad 17
Nechť platí chybový polynom $\hat{E}(X)$ z příkladu 15 a přijaté kódové slovo $R(X)$ z příkladu 12 a pro porovnání původní kódové slovo $C(X)$ z příkladu 10. Opravené kódové slovo $\hat{C}(X)$ lze vypočítat za pomoci rovnice 5.30 následovně

$$\begin{aligned}\hat{C}(X) &= R(X) + \hat{E}(X)\\ &= \alpha^8 X + \alpha^{14}X^2 + \alpha^{11}X^3 + \alpha^{14}X^4 + \alpha^{13}X^5 + \alpha^9 X^6 + X^7 + \alpha X^8 + \alpha^4 X^9 +\\ &\quad \alpha^2 X^{10} + \alpha^8 X^{11} + \alpha^2 X^{12} + \alpha^5 X^{13} + \alpha X^{14}\\ \hat{C}(X) &= C(X).\end{aligned}$$

Jelikož se původní kódové slovo a přijaté slovo po opravě nalezených chyb neliší, tak se úspěšně podařilo dekódovat a opravit všechny chyby, jež se ve slově vyskytovaly. Z opraveného slova lze získat původní zprávu z informací z rovnice 4.17 a z obrázku 3.2 $M(X) = X^7 + \alpha^1 X^8 + \alpha^4 X^9 + \alpha^2 X^{10} + \alpha^8 X^{11} + \alpha^2 X^{12} + \alpha^5 X^{13} + \alpha^1 X^{14}$, respektive ve vektorové reprezentaci $M = \begin{pmatrix}1 & 2 & 3 & 4 & 5 & 4 & 6 & 2\end{pmatrix}$. Aplikací tabulky 4.2 lze získat slovo „pilotovi“. Úspěšně se tedy podařilo přenést a získat celou zprávu.

5.7 Euklidův algoritmus

V této sekci bude popsáno využití Euklidova algoritmu k dekódování, přičemž sekce primárně vychází z literatury [18, 20]. Tento algoritmus je známý jako rekurzivní metoda pro nalezení největšího společného dělitele (NSD) dvou čísel či polynomů [20]. Nechť platí klíčová rovnice (anglicky *key equation*) definovaná v rovnici 5.28

$$\Omega(X) = S(X)L(X) \bmod X^{2t+1}.$$

Z této rovnice jsou prozatím známy pouze hodnoty $S(X)$ a t. Tuto rovnici lze přepsat jako

$$\Omega(X) = (X^{2t+1})\Theta(X) + S(X)L(X) \tag{5.31}$$

pro jakýkoliv polynom $\Theta(X)$. Rozšířený Euklidův algoritmus pro zadaný pár (a, b) vrací pár (s, t) splňující

$$c = as + bt,$$

kde c je NSD a a b. Více o principu Euklidova algoritmu (EA) viz [19]. Nechť platí následující značení $r_j(X) = \Omega(X)$, $a_j(X) = \Theta(X)$ a $b_j(X) = L(X)$. Lokalizační polynom $L(X)$ lze získat aplikací rozšířeného EA na vstupní hodnoty $r_0(X) = X^{2t+1}$ a $r_1(X) = S(X)$. Pokud v j-tém kroku algoritmu

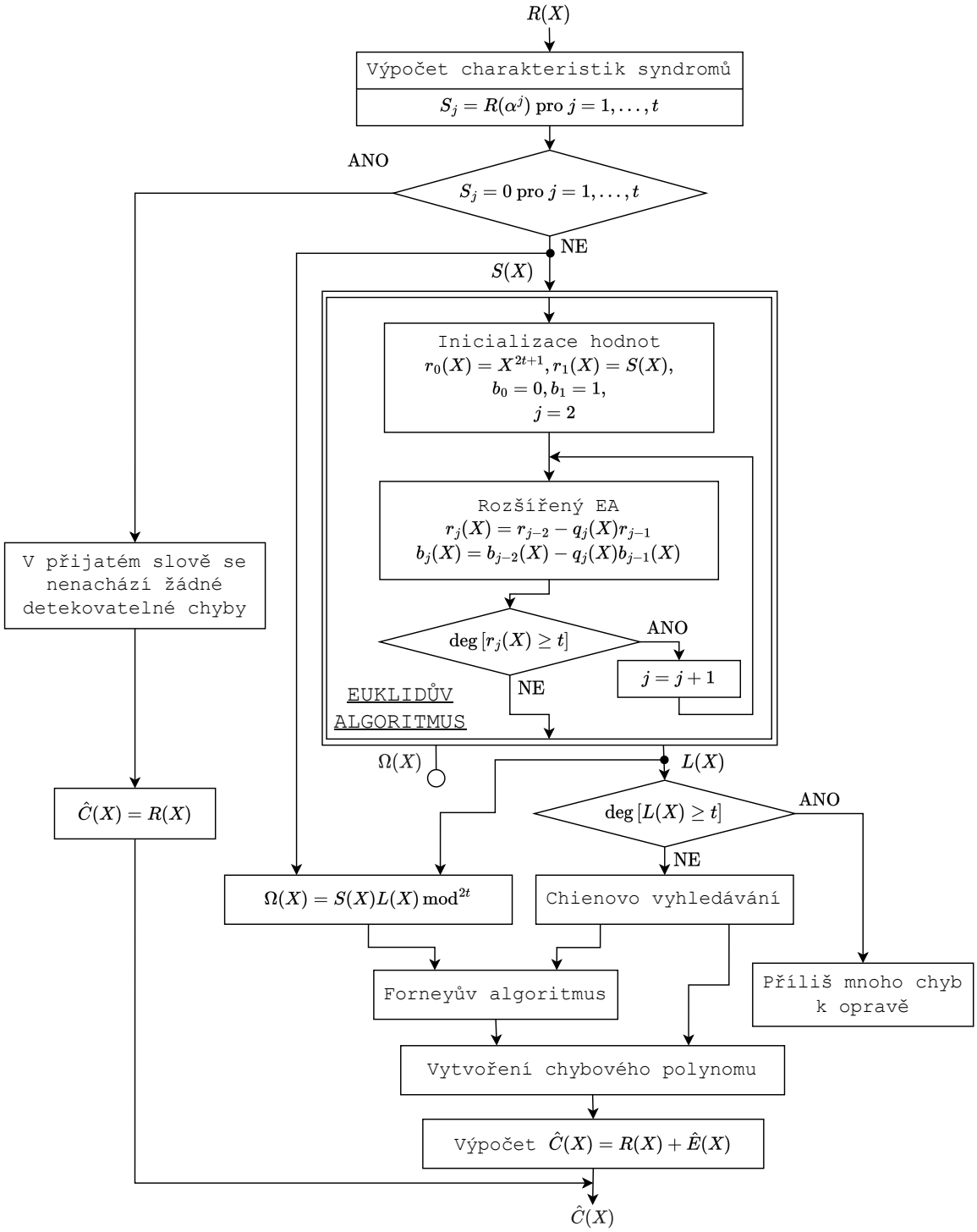
$$r_j(X) = X^{2t+1}a_j(X) + S(X)b_j(X)$$

platí, že $\deg\left[r_j(X)\right] \leq t$, tak $\Omega(X) = r_j(X)$ a $L(X) = b_j(X)$, kde $\deg\left[\ \right]$ je stupeň polynomu. Hodnota $a_j(X)$, respektive $\Theta(X)$ se pro dekódování nevyužije.

5.7.1 Dekódování za použití Euklidova algoritmu

Níže je popsán postup, vyhodnocení Euklidova algoritmu pro dekódování Reed-Solomonova kódů, primárně vycházející z literatury [19, 20]. Tento postup je rovněž znázorněn na diagramu 5.9.

1. Výpočet charakteristik syndromů z přijatého slova $R(X)$.
2. Pokud $S(X) = 0$, tak přijaté slovo neobsahuje žádné detekovatelné chyby. Následuje krok 10.
3. Pokud $S(X) \neq 0$, tak jsou počáteční hodnoty algoritmu nastaveny jako
$$\begin{aligned} r_0(X) &= X^{2t+1} \\ r_1(X) &= S(X) \\ b_0(X) &= 0 \\ b_1(X) &= 1 \\ j &= 2 \end{aligned}$$
4. Rekurzivní parametry jsou definované jako
$$\begin{aligned} r_j(X) &= r_{j-2}(X) - q_j(X) r_{j-1}(X) \\ b_j(X) &= b_{j-2}(X) - q_j(X) b_{j-1}(X) \end{aligned}$$
Přičemž rekurze pokračuje dokud $\deg\left[r_j(X)\right] \geq t$. Když $\deg\left[r_j(X)\right] < t$, tak se rekurze zastaví.
5. *Tento krok je nepovinný. Je možné získat takovou hodnotu $\lambda \in GF(2^m)$, že jejím vynásobením polynomu $b_j(X)$, respektive $\lambda b_j(X)$ vznikne monický polynom. Pak*
$$\begin{aligned} L(X) &= \lambda b_i(X) \\ \Omega(X) &= -\lambda r_i(X) \end{aligned}$$
Literatura [20] například převádí $L(X)$ do tvaru rovnice 5.16.
6. Nalezení kořenů $\frac{1}{\beta_l}$ lokalizačního polynomu Chienovým algoritmem, tím i nalezení pozic chyb β_l. Musí být splněno [20]
$$L(\beta^{-1}) = 0 \qquad \text{kde } \beta^{-1} \in GF(2^m).$$
7. Výpočet chybových hodnot Forneyovým algoritmem
$$E_l = \frac{(\beta_l)^{2-b}\, \Omega(\beta_l^{-1})}{L'(\beta_l^{-1})},$$
kde polynom $\Omega(X)$ lze získat z rovnice 5.28 nebo z posledního kroku EA.
8. Vytvoření chybového polynomu
$$\hat{E}(X) = E_1\beta_1 + E_2\beta_2 + \cdots + E_v\beta_v.$$
9. Oprava poškozeného slova
$$\hat{C}(X) = R(X) + \hat{E}(X).$$
10. Konec algoritmu



Obrázek 5.9: Vývojový diagram Euklidova algoritmu. Výpočet hodnoty $\Omega(X)$ by mohl být vynechán, jelikož samotný pricip Euklidova algoritmu produkuje tento polynom. Tento výpočet je zde ponechán, aby diagram více odpovídal konkrétní implementaci dekodéru, jež bude popsána v další kapitole. Obrázek částečně vychází z diagramu 5.8

Příklad 18
Nechť platí syndromový polynom z příkladu 16

$$S(X) = 1 + \alpha^5 X + \alpha^9 X^2 + \alpha^2 X^3 + \alpha^7 X^4 + \alpha^8 X^5 + \alpha^7 X^6.$$

Aplikací rozšířeného Euklidova algoritmu na jeho vstupní hodnoty $(r_0(X), r_1(X))$ respektive $(X^7, S(X))$, lze zkonstruovat tabulku 5.1, jež znázorňuje kroky tohoto algoritmu.

Tabulka 5.1: Tabulka kroků Euklidova algoritmu [16].

j	$r_j(X)$	$q_j(X)$	$b_j(X)$
0	X^7	—	0
1	$S(X)$	—	1
2	$\alpha^9 + \alpha^6 X + \alpha^8 X^2 + \alpha^9 X^3 + \alpha^8 X^4 + \alpha^8 X^5$	$\alpha^9 + \alpha^8 X$	$\alpha^9 + \alpha^8 X$
3	$\alpha^{11} + \alpha^{14} X + \alpha X^2$	$\alpha^3 + \alpha^{14} X$	$\alpha^{11} + \alpha^7 X + \alpha^7 X^2$

Z této tabulky lze odečíst lokalizační polynom $L(X) = b_3(X) = \alpha^{11} + \alpha^7 X + \alpha^7 X^2$ a vyhodnocovací polynom $\Omega(X) = r_3(X) = \alpha^{11} + \alpha^{14} X + \alpha X^2$. Nyní by bylo možné aplikovat na polynomy krok 7 z podsekce 5.7.1. Zde bude ukázáno, že aplikace tohoto kroku nijak neovlivní celkové výsledky.

Následuje aplikace Chienova vyhledávání na lokalizační polynom

$$\begin{aligned} L(\alpha^0) &= \alpha^{11} + \alpha^7 + \alpha^7 = \alpha^{11} \\ &\vdots \\ L(\alpha^6) &= \alpha^{11} + \alpha^{13} + \alpha^4 = 0 \\ &\vdots \\ L(\alpha^{13}) &= \alpha^{11} + \alpha^5 + \alpha^3 = 0 \\ L(\alpha^{14}) &= \alpha^{11} + \alpha^6 + \alpha^5 = \alpha^2 \end{aligned}$$

Kořeny $L(X)$ jsou α^6 a α^{13} a pozice chyb jsou tedy $\beta_1 = \frac{1}{\alpha^6} = \alpha^9$ a $\beta_2 = \frac{1}{\alpha^{13}} = \alpha^2$, což je shodné s výsledky z příkladu 14.

Pomocí Forneyova algoritmu se vypočítají chybové hodnoty

$$E_1(\alpha^9) = \frac{\alpha^9(\alpha^{11} + \alpha^{14}\alpha^6 + \alpha(\alpha^6)^2)}{\alpha^7} = \frac{\alpha^9(\alpha^{11} + \alpha^5 + \alpha^{13})}{\alpha^7} = \alpha^2\alpha^8 = \alpha^{10},$$

$$E_2(\alpha^2) = \frac{\alpha^2(\alpha^{11} + \alpha^{14}\alpha^{13} + \alpha(\alpha^{13})^2)}{\alpha^7} = \frac{\alpha^2(\alpha^{11} + \alpha^{12} + \alpha^{12})}{\alpha^7} = \alpha^{13}\alpha^8 = \alpha^6.$$

Ze zjištěných chybových pozic a hodnot lze sestavit chybový polynom

$$\hat{E}(X) = \alpha^6 X^2 + \alpha^{10} X^9,$$

jenž je shodný s chybovým polynomem z příkladu 16. I přestože byl lokalizační a vyhodnocovací polynom rozdílný od předešlých příkladů, tak se došlo ke stejnému chybovému polynomu.

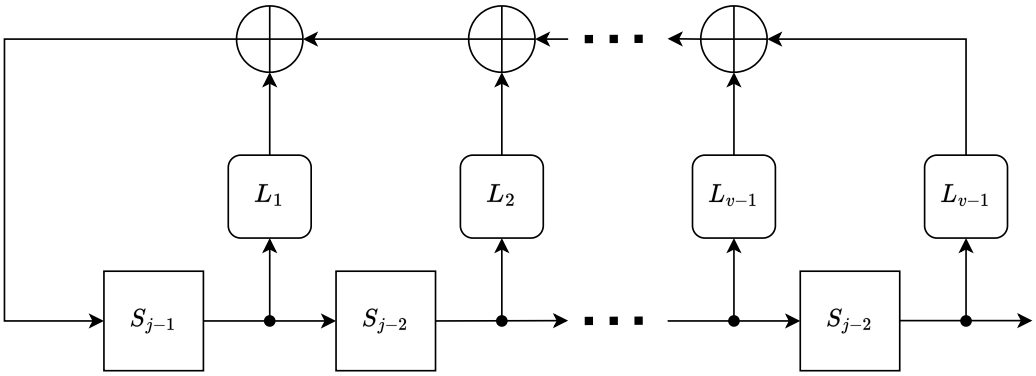
5.8 Berlekamp-Masseyho algoritmus

Berlekamp-Masseho algoritmus (BMA) je iterativní dekódovací algoritmus pro BCH a RS kódy. S prvotní myšlenkou přišel E. Berlekamp, který ji v roce 1967 představil na mezinárodním sympozium v Itálii [2] a o rok později ji zanesl do své knihy [3]. Na jeho práci navázal J. Massey, který ve své práci [17] představil řešení tohoto algoritmu pomocí LFSR, jež poskytuje další pohled na jeho vlastnosti. Princip algoritmu vychází z rovnice 5.9, kterou lze přepsat za předpokladu $j = (v+1), (v+2), \ldots, 2v$ jako

$$S_j = -\sum_{n-1}^{v} L_n S_{j-n},$$

kde tento výraz lze implementovat pomocí LFSR, jež je znázorněn na obrázku 5.10. BM algoritmus je iterativní s první iterací označenou jako $i = 1$. Nechť délka LFSR produkující první charakteristiku syndromu S_1 je $\ell^{(i)} = \ell^{(1)} = 1$ a odpovídající lokalizační polynom je $L^{(i)}(X) = L^{(1)}(X) = 1$. Předpokladem je, že tento polynom také produkuje S_2. S přihlédnutím k rovnici 5.9 a obrázku 5.10 lze odvodit vztah mezi těmito dvěma charakteristikami syndromu jako

$$S_{i+1} = -\sum_{n=1}^{\ell^{(i)}} L_n^{(i)} S_{i+1-n}.$$



Obrázek 5.10: Obvod LFSR se zpětnovazebními vstupy (anglicky *taps*) $L_1, L_2, \ldots, L_v$ a výstupy $S_1, S_2, \ldots, S_{2v}$ za předpokladu $j = v+1, v+2, \ldots, 2v$. Diagram byl převzat z [20].

Ve speciálních případech může být takto vypočítaná charakteristika syndromu aproximována předchozí charakteristikou. Obecně však toto neplatí a aktuální návrh LFSR může být od požadovaného rozdílný. Jak moc dostatečně produkuje současný návrh LFSR další syndrom, je dán odchylkou d, jejíž výpočet je dán rovnicí

$$d^{(i)} = S_{i+1} + \sum_{n=1}^{\ell^{(i)}} L_n^{(i)} S_{i+1-n} = \sum_{n=0}^{\ell^{(i)}} L_n^{(i)} S_{i+1-n}.$$

Jestliže je současný návrh LFSR správný, tak výsledek této rovnice je nulový. Znamená to tedy, že současné nastavení LFSR, respektive proměnné $\ell^{(i)}$ a $L^{(i)}(X)$, jsou správně nastavené a musí tudíž zůstat po zbytek běhu algoritmu neměnné.

Jinými slovy se současný design LFSR nebude měnit. Respektive, pokud $d = 0$, tak nastavení LFSR je

$$\begin{aligned} \ell^{(i+1)} &= \ell^{(i)}, \\ L^{(i+1)}(X) &= L^{(i)}(X), \\ i &= i + 1. \end{aligned}$$

Pokud je design LFSR nesprávný, tak $d^{(i)} \neq 0$ a je tedy potřeba ho upravovat, dokud nebude $d^{(i)} = 0$ nebo dokud se nepřekročí možný počet iterací $2t$. Jedním ze způsobů, jak korektně vygenerovat všechny charakteristiky syndromu, je zapamatování si posledního případu iterace m, kdy se LFSR nepodařilo vyprodukovat následující charakteristiku syndromu S_m. Využije se tedy m-tý design LFSR s asociovanou odchylkou d_m za účelem modifikace současného návrhu LFSR. Toto řešení lze shrnout vztahem

$$L^{(i+1)}(X) = L^{(i)}(X) - X^{i-m}\frac{d^{(i)}}{d^{(m)}}L^{(m)}(X).$$

Detailnější vysvětlení tohoto algoritmu pracujícího na principu LFSR a všech jeho částí je možné nálezt například v literatuře [14, 16], pričemž tato sekce primárně vychází z literatury [14].

5.8.1 Dekódování za použití Berlekamp-Masseyho algoritmu

Níže je popsán postup vyhodnocení Berlekamp-Masseyho algoritmu pro dekódování Reed-Solomonova kódů, kde tento postup vychází z rovnic, jež byly popsány výše. V literatuře [14, 18, 7] je možné nalézt obdobné postupy, jak vyhodnocovat tento algoritmus. Tento postup je rovněž znázorněn na diagramu 5.11.

1. Výpočet charakteristik syndromů z přijatého slova $R(X)$.
2. Pokud $S(X) = 0$, tak přijaté slovo neobsahuje žádné detekovatelné chyby. Následuje krok 19.
3. Pokud $S(X) \neq 0$, tak jsou počáteční hodnoty algoritmu nastaveny jako

$$\begin{aligned} \text{Čítač iterací: } & i = 1 \\ \text{Délka LFSR: } & \ell = 0 \\ \text{Propojovací polynom: } & L(X) = 1 \\ \text{Propojovací pomocný polynom: } & \rho(X) = X. \end{aligned}$$

4. Výpočet odchylky

$$d = S_i + \sum_{j=1}^{\ell} L_j S_{i-j}.$$

5. Pokuď odchylka $d = 0$, tak následuje krok 11.
6. Pokud $d \neq 0$, je možné opravit dočasný propojovací polynom $T(X)$ přidáním pomocného propojovacího polynomu $\rho(X)$ k současnému propojovacímu polynomu $L^{(i)}(X)$ následovně

$$T(X) = L(X) - d\rho(X).$$

7. Kontrola velikosti LFSR. Pokud platí $2\ell \geq i$, tak následuje krok 9.

8. Vzhledem ke kroku 7 musí být LFSR zvětšen. Zvětšení probíhá následovně

$$\ell = i - \ell.$$

9. Normalizace propojovacího polynomu $L(X)$ vydělením odchylkou $d \neq 0$ a následné uložení výsledku do pomocného LFSR $\rho(X)$ následovně

$$\rho(X) = \frac{L(X)}{d}.$$

10. Nyní je možné přepsat $L(X)$, neboť byl normalizován a uložen do $\rho(X)$ během kroku 9. Je tedy možné aktualizovat tento propojovací polynom $L(X)$ následovně

$$L(X) = T(X).$$

11. Posunutí pomocného LFSR o jednu pozici použitím

$$\rho(X) = X\rho(X).$$

12. Pokud $2t = i$, tak následuje skok na krok 14, jinak krok 13.

13. Inkrementace čítače $i = i + 1$ a skok na krok 4.

14. Nalezení kořenů lokalizačního polynomu Chienovým algoritmem, tím i nalezení pozic chyb.

15. Výpočet chybových hodnot Forneyovým algoritmem.

16. Vytvoření chybového polynomu.

17. Oprava poškozeného slova.

18. Konec algoritmu.

Konkrétní aplikace toho algoritmu je znázorněna na příkladu 19. Podsekce primárně vychází z [20].

Příklad 19
Nechť platí přijaté kódové slovo $R(X)$ a z něho vypočítané charakteristiky syndromu z příkladu 12

$$S_1 = \alpha^5 \quad S_2 = \alpha^9 \quad S_3 = \alpha^2 \quad S_4 = \alpha^7 \quad S_5 = \alpha^8 \quad S_6 = \alpha^7.$$

Níže bude sepsán postup vyhodnocení BM algoritmu vůči přijatému slovu dle kroků popsaných v podsekci 5.8.1. Jelikož syndrom je nenulový, a tudíž se v přijatém slově vyskytuje alespoň jedna chyba, tak je potřeba inicializovat počáteční hodnoty z třetího kroku algoritmu.

◦ Inicializace počátečních hodnot:

$$\begin{aligned} i &= 1, \\ \ell &= 0, \\ L(X) &= 1, \\ \rho(X) &= X. \end{aligned}$$

- Iterace $i = 1$ se vstupními hodnotami: $\ell = 0$, $L(X) = 1$ a $\rho(X) = X$.

  {4} Výpočet odchylky: $d = S_1 + \sum_{j=1}^{0} L_j S_{1-j} = \alpha^5$.

  {5} Kontrola zda $d = 0 \Leftrightarrow \alpha^5 = 0 \rightarrow$ NE.

  {6} Oprava dočasného propojovacího polynomu: $T(X) = L(X) - d\rho(X) = 1 - \alpha^5 X$.

  {7} Kontrola velikosti LFSR: $2\ell \geq i \Leftrightarrow 2 \cdot 0 \geq 1 \rightarrow$ NE.

  {8} Zvětšení LFSR: $\ell = i - \ell = 1 - 0 = 1$.

  {9} Normalizace propojovacího polynomu a uložení do $\rho(X)$: $\rho(X) = \frac{1}{\alpha^5} = \alpha^{10}$.

  {10} Přepsaní $L(X)$, jelikož již bylo zpracováno: $L(X) = T(X) = 1 - \alpha^5$.

  {11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha^{10} X$.

  {12} Kontrola zda $2t = i \Leftrightarrow 6 = 1 \rightarrow$ NE.

  {13} Inkrementace čítače: $i = i + 1 = 1 + 1 = 2$.

- Iterace $i = 2$ se vstupními hodnotami: $\ell = 1$, $L(X) = 1 + \alpha^5$ a $\rho(X) = \alpha^{10} X$.

  {4} Výpočet odchylky: $d = S_2 + \sum_{j=1}^{1} L_j S_{2-j} = \alpha^9 + \alpha^{10} = \alpha^{13}$.

  {5} Kontrola zda $d = 0 \Leftrightarrow \alpha^{13} = 0 \rightarrow$ NE.

  {6} Oprava dočasného propojovacího polynomu: $T(X) = L(X) - d\rho(X) = 1 - \alpha^4 X$.

  {7} Kontrola velikosti LFSR: $2\ell \geq i \Leftrightarrow 2 \cdot 1 \geq 2 \rightarrow$ ANO.

  {9} Normalizace propojovacího polynomu: $\rho(X) = \frac{1-\alpha^5 X}{\alpha^{13}} = \alpha^2 - \alpha^7 X$.

  {10} Přepsaní $L(X)$, jelikož již bylo zpracováno: $L(X) = T(X) = 1 - \alpha^4 X$.

  {11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha^2 X - \alpha^7 X^2$.

  {12} Kontrola zda $2t = i \Leftrightarrow 6 = 2 \rightarrow$ NE.

  {13} Inkrementace čítače: $i = i + 1 = 2 + 1 = 3$.

- Iterace $i = 3$ se vstupními hodnotami: $\ell = 1$, $L(X) = 1 + \alpha^4 X$ a $\rho(X) = \alpha^2 X - \alpha^7 X^2$.

  {4} Výpočet odchylky: $d = S_3 + \sum_{j=1}^{1} L_j S_{3-j} = \alpha^2 + \alpha^{13} = \alpha^{14}$.

  {5} Kontrola zda $d = 0 \Leftrightarrow \alpha^{14} = 0 \rightarrow$ NE.

  {6} Oprava dočasného propojovacího polynomu: $T(X) = L(X) - d\rho(X) = 1 - X - \alpha^6 X$.

  {7} Kontrola velikosti LFSR: $2\ell \geq i \Leftrightarrow 2 \cdot 1 \geq 3 \rightarrow$ NE.

  {8} Zvětšení LFSR: $\ell = i - \ell = 3 - 1 = 2$.

{9} Normalizace propojovacího polynomu: $\rho(X) = \frac{1-\alpha^4 X}{\alpha^{14}} = \alpha - \alpha^5 X$.

{10} Přepsaní $L(X)$, jelikož již bylo zpracováno: $L(X) = T(X) = 1 - X - \alpha^6 X$.

{11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha X - \alpha^5 X^2$.

{12} Kontrola zda $2t = i \Leftrightarrow 6 = 3 \rightarrow$ NE.

{13} Inkrementace čítače: $i = i + 1 = 3 + 1 = 4$.

- Iterace $i = 4$ se vstupními hodnotami: $\ell = 2$, $L(X) = 1 - X - \alpha^6 X^2$ a $\rho(X) = \alpha X - \alpha^5 X^2$.

  {4} Výpočet odchylky: $d = S_4 + \sum_{j=1}^{2} L_j S_{4-j} = \alpha^7 + \alpha^2 + \alpha^0 = \alpha^{11}$.

  {5} Kontrola zda $d = 0 \Leftrightarrow \alpha^{11} = 0 \rightarrow$ NE.

  {6} Oprava dočasného propojovacího polynomu: $T(X) = L(X) - d\rho(X) = 1 - \alpha^{11} X - \alpha^{11} X^2$.

  {7} Kontrola velikosti LFSR: $2\ell \geq i \Leftrightarrow 2 \cdot 2 \geq 4 \rightarrow$ ANO.

  {9} Normalizace propojovacího polynomu: $\rho(X) = \frac{1-X-\alpha^6 X^2}{\alpha^{11}} = \alpha^4 - \alpha^4 X - \alpha^{10} X^2$.

  {10} Přepsaní $L(X)$, jelikož již bylo zpracováno: $L(X) = T(X) = 1 - \alpha^{11} X - \alpha^{11} X^2$.

  {11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha^4 X - \alpha^4 X^2 - \alpha^{10} X^3$.

  {12} Kontrola zda $2t = i \Leftrightarrow 6 = 4 \rightarrow$ NE.

  {13} Inkrementace čítače: $i = i + 1 = 4 + 1 = 5$.

- Iterace $i = 5$ se vstupními hodnotami: $\ell = 2$, $L(X) = 1 - \alpha^{11} X - \alpha^{11} X^2$ a $\rho(X) = \alpha^4 X - \alpha^4 X^2 - \alpha^{10} X^3$.

  {4} Výpočet odchylky: $d = S_5 + \sum_{j=1}^{2} L_j S_{5-j} = \alpha^8 + \alpha^3 + \alpha^{13} = 0$.

  {5} Kontrola zda $d = 0 \Leftrightarrow 0 = 0 \rightarrow$ ANO.

  {11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha^4 X^2 - \alpha^4 X^3 - \alpha^{10} X^4$.

  {12} Kontrola zda $2t = i \Leftrightarrow 6 = 5 \rightarrow$ NE.

  {13} Inkrementace čítače: $i = i + 1 = 5 + 1 = 6$.

- Iterace $i = 6$ se vstupními hodnotami: $\ell = 2$, $L(X) = 1 - \alpha^{11} X - \alpha^{11} X^2$ a $\rho(X) = \alpha^4 X^2 - \alpha^4 X^3 - \alpha^{10} X^4$.

  {4} Výpočet odchylky: $d = S_6 + \sum_{j=1}^{2} L_j S_{6-j} = \alpha^7 + \alpha^4 + \alpha^3 = 0$.

  {5} Kontrola zda $d = 0 \Leftrightarrow 0 = 0 \rightarrow$ ANO.

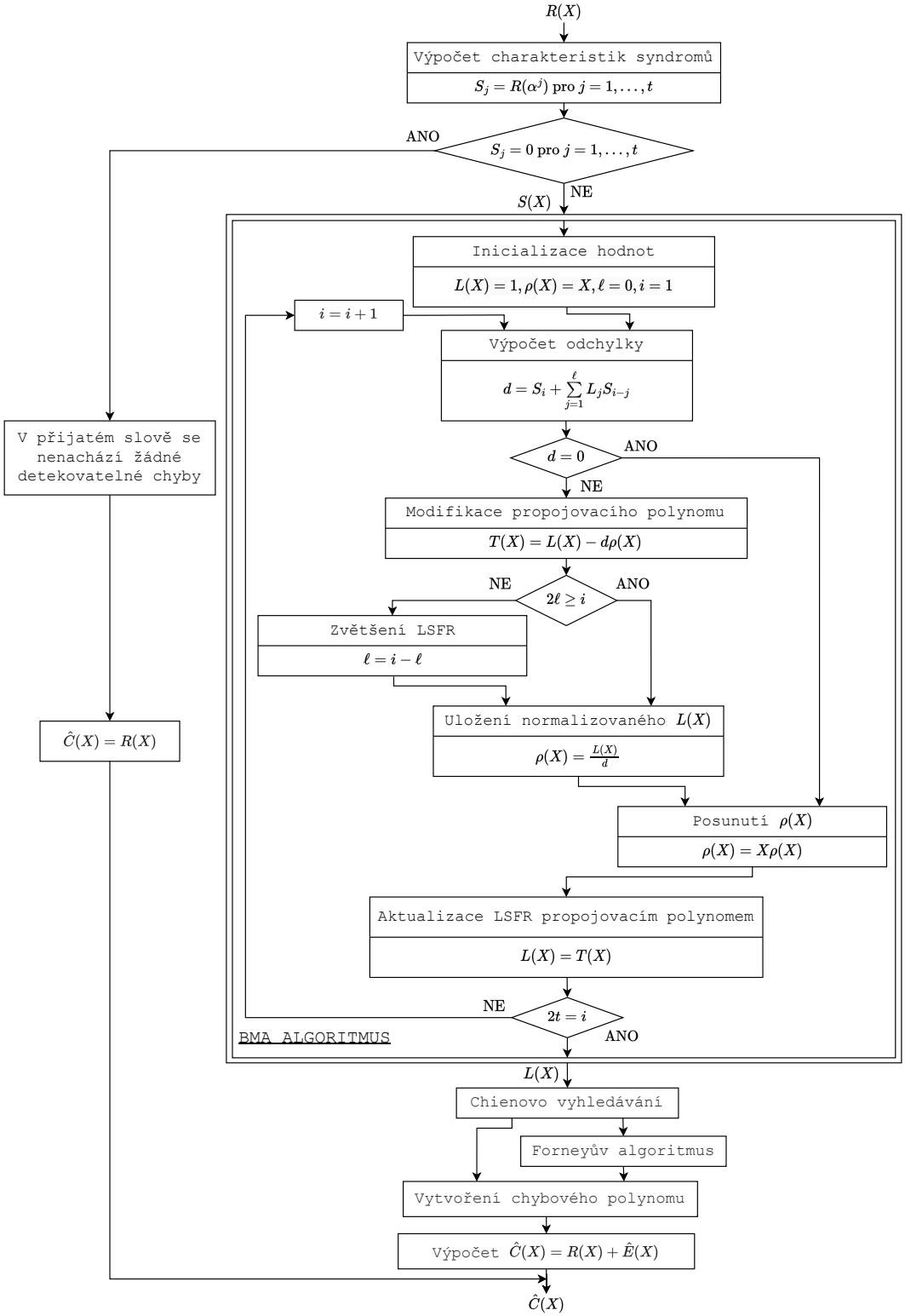
  {11} Posunutí pomocného LFSR: $\rho(X) = X\rho(X) = \alpha^4 X^3 - \alpha^4 X^4 - \alpha^{10} X^5$.

  {12} Kontrola zda $2t = i \Leftrightarrow 6 = 6 \rightarrow$ ANO. Konec iterací BM algoritmu.

- Chienův algoritmus se vstupním lokalizačním polynomem $L(X) = 1 - \alpha^{11} X - \alpha^{11} X^2$, respektive $L(X) = 1 + \alpha^{11} X + \alpha^{11} X^2$.

  ...

Výstupním polynomem tohoto algoritmu, jakožto u Euklidova a PGZ algoritmu, je opět $L(X) = 1 + \alpha^{11}X + \alpha^{11}X^2$ s jehož pomocí by se opět našly lokace chyb pomocí Chienova vyhledávání (příklad 14) a následně i jejich hodnoty pomocí Forneyova algoritmu (příklad 16). Následně by se ze získaných chybových hodnot sestavil chybový polynom (příklad 17), který by se přičetl k přijaté zprávě a výsledkem by bylo opravené kódové slovo $\hat{C}(X)$.



Obrázek 5.11: Vývojový diagram Berlekamp-Masseyho algoritmu. Obrázek byl převzat s úpravami z [14].

Kapitola 6

Implementace

Tato kapitola bude pojednávat o konkrétní implementaci kodéru a dekodéru Reed-Solomonova kódu včetně jejich vedlejších částí jako například generování prvků Galoisových těles nebo generování generujícího polynomu. Následně je popsán způsob implementace měření výpočetní náročnosti se zavedenými kódovými scénáři. Na diagramu 6.1 je znázorněna zjednodušená implementace programu. Konkrétní příklady spouštění programu společně s popisem všech přepínačů lze nalézt v příloze B.

6.1 Úvod do implementace

Vytvořený program je ve formě konzolové aplikace, která nese název `ReedSolomon`. Aplikace má dva způsoby spouštění, respektive dva způsoby její parametrizace. Parametry jsou zadávány

1. pomocí přepínačů před spuštěním programu nebo
2. v průběhu běhu programu, kdy je uživateli řečeno, jaký parametr se má zrovna zadat.

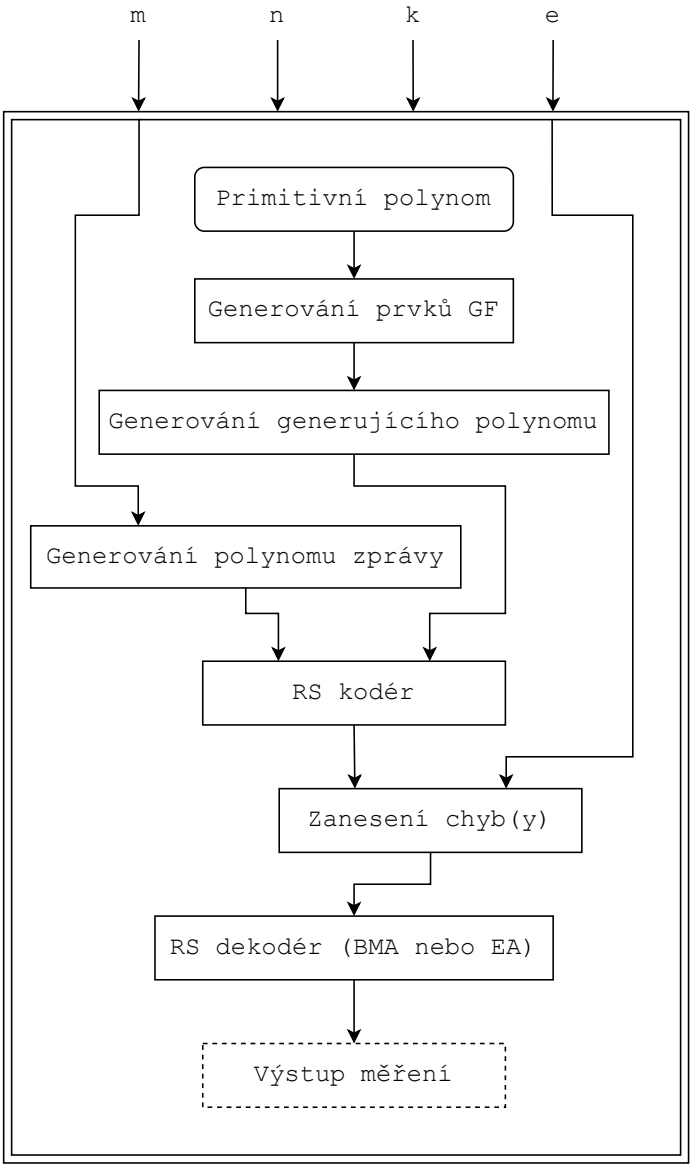
Více o možnostech spouštění je uvedeno v přiloženém `README` anebo v nápovědě programu, kterou lze vyvolat přepínačem `-h`. Přepínače, které musí být uživatelem specifikovány, aby aplikace korektně fungovala jsou

1. `-n` – délka kódového slova,
2. `-k` – délka informační části,
3. `-m` – informace (zpráva), která má být zakódována a následně dekódována,
4. `-e` – chyby, které budou zaneseny na určité místo v kódovém slově.

Vstupní parametry n a k nejsou v diagramu 6.1 nikam navedeny, jelikož se tyto parametry vyskytují globálně napříč celým programem, a proto nejsou v diagramu pro přehlednost zakresleny.

Na standardní výstup je při běhu aplikace tištěn její postup, který reflektuje postupy kódování a dekódování Reed-Solomonovými kódy zmíněné v předchozích kapitolách. Jedná se například o tabulku konečných těles 2.1, operační tabulky sčítání 2.2 a násobení 2.3 nebo postupy dekódování Berlekamp-Masseyovým či Euklidovým algoritmem.

V následujících sekcích budou popsány podstatné dílčí části implementace.



Obrázek 6.1: Diagram přenosového kanálu. Diagram je převzatý z [9]

6.1.1 Seznam použitých technologií

V následujícím seznamu jsou sepsány technologie, jež byly použity při tvorbě implementace

- Programovací jazyk C++ ve verzi 17,
- GNU překladač g++ (použitá verze 11.3.0),
- Python (použitá verze 3.10.3),
- GNU Make (použitá verze 4.3).

Podstatnou částí je využití verze 3.10 nebo vyšší u jazyka Python, jelikož měřicí skript v implementaci využívá řídicí příkaz `match` (ekvivalentem v jazyku C je `switch`), který se v dřívějších verzích nenachází.

6.2 Generování klíčových částí Reed-Solomonova kódu

V této sekci budou popsány tři základní části, jež jsou potřeba vytvořit před samostatnou funkcí kodéru a dekodéru Reed-Solomonova kódu, které jsou

1. vygenerování prvků konečného tělesa,
2. vytvoření generujícího polynomu,
3. vytvoření informačního polynomu.

Generování prvků konečného tělesa vychází z [4]. Algoritmus konkrétně vychází z metody využívající primitivní polynom, kdy je každý jeden prvek tělesa vyjádřen jako zbytek po dělení obecné mocniny X^n pro $n = 0, \ldots, 2^m - 2$ primitivním polynomem. Algoritmus je realizovaný ve funkci `generateGFElements()`. Prvky jsou po vygenerování uloženy do struktury `mapa` pojmenované jako `map`, která má globální působení v programu. Mapa má jako klíč hodnotu exponentu v exponenciálním zápisu. Klíč odkazuje na hodnotu vektorového binárního zápisu převedeného do dekadické podoby, viz příklad 1.

Generující polynom a jeho generování vychází ze sekce 4.2, kdy funkce `generateGX()` realizuje rovnici 4.10 s přednastavenou hodnotou $b = 1$.

Tvorba informačního polynomu, respektive polynomu zprávy vychází z příkladu 7, kdy je převedena posloupnost dekadických čísel do polynomiálního tvaru za využití vygenerovaného konečného tělesa. Je potřeba dodržet maximální možnou délku zprávy danou hodnotou n. Obdobně se toto týká i zadaných dílčích částí zprávy, které musí být v intervalu $< 0, n >$. Funkce realizující tento postup je `generateMX()`.

Nyní jsou vygenerovány všechny části potřebné pro správnou funkci kodéru a dekodéru Reed-Solomonova kódu.

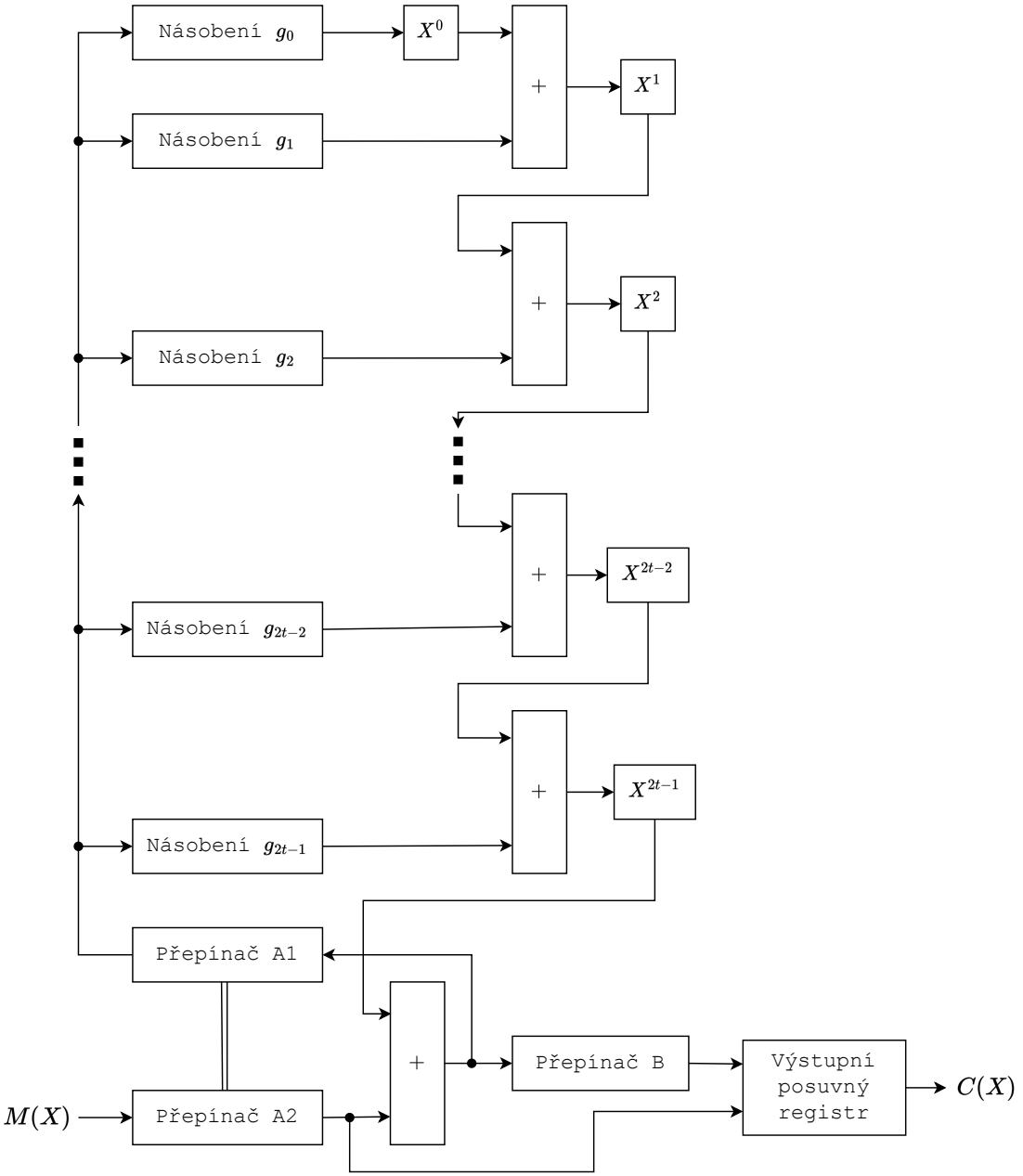
6.3 Kodér Reed-Solomonova kódu

Implementace kodéru Reed-Solomonova kódu vychází z popisu obvodu kodéru v [9]. Funkci obvodu lze popsat procedurou, kde obvod, z kterého procedura vychází, je zobrazen na diagramu 6.2. Postup procedury lze popsat následovně:

1. Vyčištění zpětnovazebního posuvného registru ($X^0 = X^1 = \cdots = X^{2t-1} = 0$).
2. Povolení přepínače A1, A2 a zakázání přepínače B.
3. Postupné nahrání informačních symbolů (celkem k symbolů).
4. Zakázání přepínače A1, A2 a povolení přepínače B.
5. Aplikace posuvu v obvodu celkem $n - k$ krát za účelem získání kontrolních symbolů.
6. Zakódované slovo se nachází ve výstupním registru.
7. Skok na krok 1 pro zakódování další zprávy.

Sepsané kroky procedury společně s diagramem byly inspirací pro implementaci funkce `RS_encode()` implementace. Vstupními parametry jsou generující polynom společně se zprávou a výstupem je kódové slovo v systematickém tvaru.

V diagramu se za bloky označené jako "+" skrývá bloková sčítačka prvků konečného tělesa a za bloky označené jako "Násobení g_j" pro $j = 0, 1, \ldots, 2t-2, 2t-1$, se skrývá bloková násobička prvků konečného tělesa. Bloky jsou v pořadí implementovány pomocí funkcí `block_add()` a `block_mul()`. Vstupem funkcí jsou dva prvky konečného tělesa, respektive jejich exponenciální reprezentace (klíče v mapě), na které má být daná operace aplikována. Operace sčítání aplikuje operaci XOR na dekadickou reprezentaci vstupů, tedy na hodnoty, které odkazuje klíč v mapě `map`. Operace násobení pracuje pouze s exponenciální reprezentací, kde je výsledek dán jako zbytek po dělení součtu obou vstupů hodnotou n, která odpovídá délce kódového slova.



Obrázek 6.2: Diagram kodéru Reed-Solomonova kódu využívající posuvný registr. Diagram je převzatý s úpravami z [9]

6.4 Dekodér Reed-Solomonova kódu

Dekodér je realizován pomocí dvou algoritmů a to

- Berlekamp-Masseyho (BM) algoritmu,
- Euklidova algoritmu (EA).

Kód ke zmíněným algoritmům byl převzat z literatury [20], kde byl přiložen jako podpůrný materiál pro studium, který lze nalézt na internetové stránce `http://the-art-of-ecc.com/index.html`. Moje implementace, skládající se z částí popsaných v předchozích sekcích a z této převzaté implementace, jež byla za účelem kompatibility obou programů modifikována tvoří, společně jeden funkční celek realizující aplikaci umožňující kódování a dekódování Reed-Solomonových kódů.

Přibližný postup, jakým funkce pro dekódování pracují, je znázorněn na diagramu 5.9 pro Euklidův algoritmus a na diagramu 5.11 pro BM algoritmus.

6.5 Schopnost implementace

Implementace v aktuálním stavu umožňuje práci s maximální délkou kódových slov 255 symbolů, v prostudované literatuře jsem nenašel, že by se v praxi používala delší kódová slova, jelikož Reed-Solomonovy kódy se zmíněnou maximální délkou mají symboly, které lze zaznamenat pomocí osmi bitů, tedy jednoho bajtu. Je však možné modifikací funkce `set_primitive_polynomial()` v souboru `GF_arithmetic.cpp` do programu zanést další primitivní polynomy, jež by umožňovaly práci s delšími kódovými slovy.

Převzatá část kódu je vytvořena v programovacím jazyku C a nachází se zde tedy struktura `pole`, která program limituje rovněž z hlediska maximální délky kódových slov. Velikosti polí byly z převzaté části upraveny, tak aby umožňovaly pracovat s délkou kódových slov 1023 symbolů. Má část implementace je vytvořena v jazyku C++ a využívám struktur `std::vector`, které nemají pevně danou délku.

Limitace délky kódového slova není teoreticky žádná (za předpokladu dostatečného zvětšení velikosti polí). Je zde však limitace ze strany velikosti operační paměti nebo systému. Navíc by byla latence tak vysoká, že by v praktickém využití z největší pravděpodobností znemožňovala zpracovávat informace v dostatečně vysoké rychlosti [9].

Poznámka: Aktuální velikost polí je předimenzována tak, aby dokázala zpracovat kódové slovo o maximální délce 1023 symbolů. Ve funkci `set_primitive_polynomial()` *jsou předpřipravené „zakomentované“ řádky s primitivními polynomy za účelem případného vyzkoušení.*

6.6 Měření výpočetní náročnosti

Měření výpočetní náročnosti probíhá rovněž v souborech programu `ReedSolomon` pomocí čítačů iterací a pomocí struktury `std::chrono::high_resolution_clock`, jež je použita pro měření časové náročnosti. Měření se specializuje na iterační a časovou náročnost. Každé z těchto měření bude provedeno dvakrát, a to pro Berlekamp-Masseyův a Euklidův dekódovací algoritmus. Pro výsledky z každého měření je v následující kapitole vymezená sekce.

Samotné spouštění a analýzu experimentů realizuje skript v jazyku Python `measure.py`. Skript spouští program `ReedSolomon`, kterému nastavuje určité přepínače dle typu experi-

mentu a následně zpracovává výsledky, jež program ukládá do souboru. Přepínače, nastavené skriptem do programu `ReedSolomon`, které jsou potřebné pro realizaci měření, jsou

- `-r` – povolení přepínačů (jinak by byl program parametrizován ze standardního vstupu),
- `-n` – délka kódového slova,
- `-k` – maximální délka informační části,
- `-m` – informační část,
- `-e` – počet chyb a jejich následné lokace a hodnoty,
- `-f` – jméno souboru do kterého budou mezivýsledky dočasně ukládány,
- `-i` – specifikuje měření iterační náročnosti,
- `-t` – specifikuje měření časové náročnosti,
- `-c` – specifikuje, kolik měření bude provedeno (primárně využitelné s přepínačem `-t`).

Nechť parametry přepínačů `-n` a `-k` jsou v pořadí n a k. Jejich hodnoty v experimentech jsou uloženy v souboru `RS.in`, kde jsou zapsány ve formátu (n, k). Těmito dvojicemi jsou určeny kódové scénáře, jimiž je primárně parametrizován Reed-Solomonovův kód, a se kterými probíhají experimenty. Volba kódových scénářů vychází z [8], které byly doplněny dalšími kódovými scénáři, aby byl analyzován celý rozsah délek informačních částí kódových slov. Kódové scénáře jsou dány tabulkou 6.1.

Tabulka 6.1: Výčet kódových scénářů, které budou podrobeny měření.

n	k
63	55
63	47
63	39
63	31
63	23
63	15

n	k
255	239
255	225
255	205
255	191
255	183
255	175

n	k
255	165
255	135
255	117
255	98
255	76
255	65

n	k
255	51
255	38
255	28
255	17

Za účelem měření dekódovacího procesu jsou do zakódované zprávy zaneseny chyby. Před zanesením vygenerované chyby na určitou pozici v kódovém slově je nejdříve zkontrolováno, zda se na dané pozici již tato hodnota w nenachází. Pokud se zde vyskytuje, tak je nahrazena hodnotou $(w+1) \bmod n$. Například by mohla při experimentu nastat situace, kdy mají být zaneseny do kódového slova celkem tři chyby, ale vlivem shodnosti původní hodnoty a hodnoty chyby budou zaneseny pouze dvě chyby. Byly by tak ovlivněny výsledky měření. Při experimentech s časovou náročností je nastaveno, aby se kódovaní a dekódovaní provedlo celkem stokrát, kdy jsou jednotlivé mezivýsledky sčítány a následně je na výstup tištěn výsledek vydělený počtem provedení. Částečně se tímto odladí odchylka mezi výsledky způsobená proměnlivou vytížeností CPU. Pro ještě více odladěné mezivýsledky byla frekvence CPU při měření nastaven na hodnotu $1\,GHz$ programem `cpupower-frequency-set`[1]

Výstupem skriptu se spuštěným experimentem je tabulka vytištěná na standardní výstup zobrazující výsledky měření.

[1] Manuálové stránky programu https://manpages.ubuntu.com/manpages/trusty/man1/cpupower-frequency-set.1.html

Kapitola 7

Srovnávací studie Reed-Solomonových kódů

V této kapitole budou prezentovány výsledky měření výpočetní náročnosti určených kódových scénářů dané tabulkou 6.1. Výpočetní náročnost se zaměřuje na dva aspekty, a to konkrétně:

- Iterační náročnost – Měření se zaobírá tím, kolik jednotlivé algoritmy provázející kódovací nebo dekódovací proces musely vykonat iterací. Například Chienův vyhledávací algoritmus, při délce kódového slova n symbolů, musí provést n iterací, aby zjistil, zda jsou všechny určité prvky konečného tělesa kořenem lokalizačního polynomu či nikoli. Dalším příkladem je kontrola všech charakteristik syndromů, která bude trvat $2 * t$ iterací, aby bylo zkontrolováno, zda se v přijatém slově vyskytuje detekovatelná chyba.
- Časová náročnost – Měření se vztahuje na to kolik času dohromady zaberou jednotlivé algoritmy provázející kódovací nebo dekódovací proces.

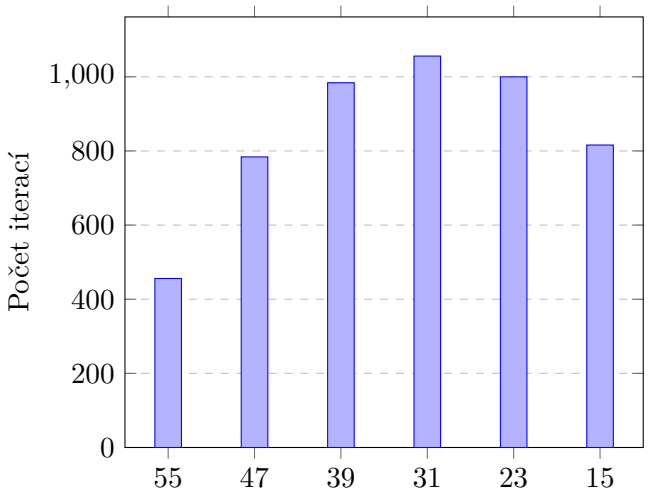
Poznámka: Výstupem implementace jsou také statistiky o iterační čí časové náročnosti jednotlivých algoritmů kódovacího a dekódovacího procesu.
Na následujícím výčtu jsou vypsány konkrétní experimenty a jejich konkrétní zaměření:

- Experiment 1 – iterační náročnost za použití dekodéru využívajícího BM algoritmus.
- Experiment 2 – iterační náročnost za použití dekodéru využívajícího EA.
- Experiment 3 – časová náročnost za použití dekodéru využívajícího BM algoritmus.
- Experiment 4 – časová náročnost za použití dekodéru využívajícího EA.

Pro přehlednost bude iterační a časová náročnost kódování Reed-Solomonva kódu zobrazena pouze jednou, jelikož pro každý dekódovací proces je takřka shodná. Výsledky měření budou zobrazeny ve formě sloupcového nebo čárového grafu. Při měření časové výpočetní náročnosti nejsou výsledky na rozdíl od iterační konzistentní. Z tohoto důvodu bude na grafy zobrazující výsledky těchto měření aplikována lineární regrese. Grafy tedy budou proloženy aproximační přímkou, ale pouze v případě, kdy nebude narušovat čitelnost grafu. Časová výpočetní náročnost je měřena v mikrosekundách, a je tak zaznamenána i v daných grafech, avšak pro lepší čitelnost jsou hodnoty na osách znázorňující čas zaneseny tak, aby bylo nekomplikované převést hodnoty na milisekundy. V poslední sekci budou diskutovány výsledky měření výpočetní náročnosti.

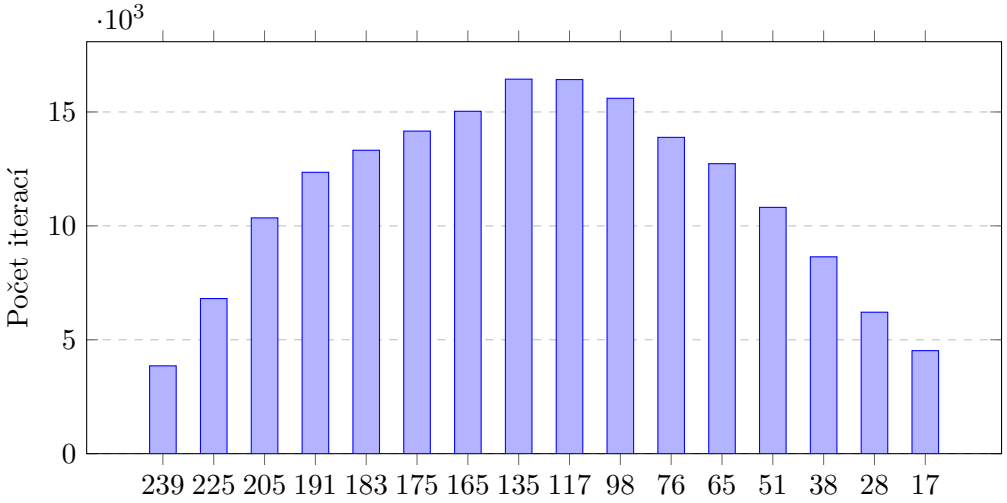
7.1 Experiment 1

Tento experiment se zaměřuje na iterační výpočetní náročnost kódování a dekódovaní Reed-Solomonova kódu, kdy dekodér využívá Berklamp-Masseyho algoritmus. Graf 7.1 zobrazuje výsledky měření kódování při délce kódového slova $n = 63$ symbolů a při délce kódového slova $n = 255$ symbolů jsou výsledky vyobrazeny na grafu 7.2. Zatímco výsledky pro dekódování jsou pro $n = 63$ symbolů prezentovány na grafu 7.3 a pro $n = 255$ symbolů na grafu 7.4.



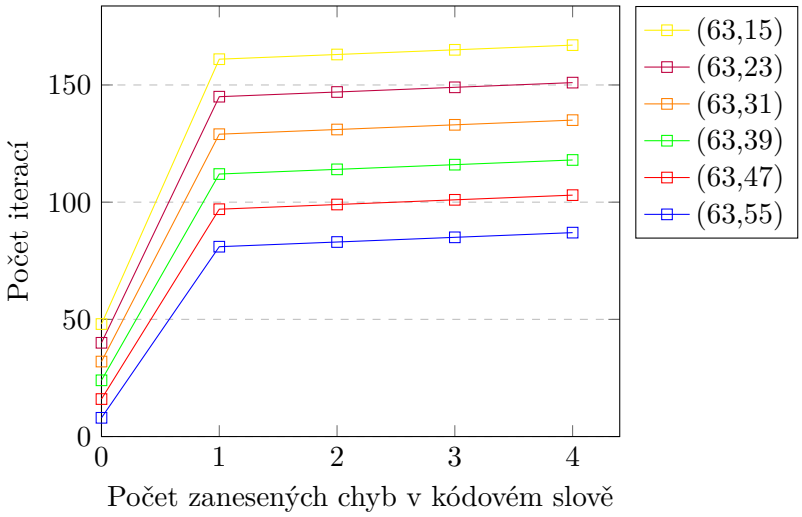
Počet symbolů k v informační části RS kódů při délce kódového slova $n = 63$ symbolů

Obrázek 7.1: Iterační výpočetní náročnost kódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů.

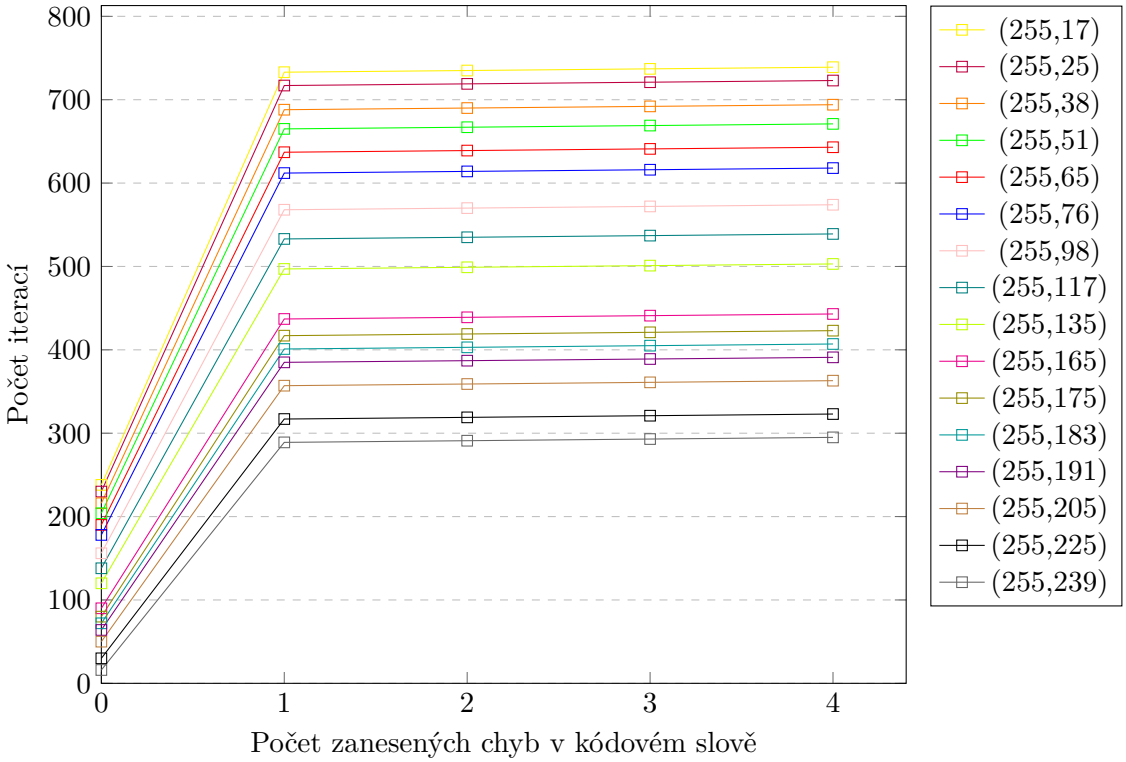


Počet symbolů k v informační části RS kódů při délce kódového slova $n = 255$ symbolů

Obrázek 7.2: Iterační výpočetní náročnost kódování RS kódů o délce kódového slova $n = 255$ symbolů z určených kódových scénářů.



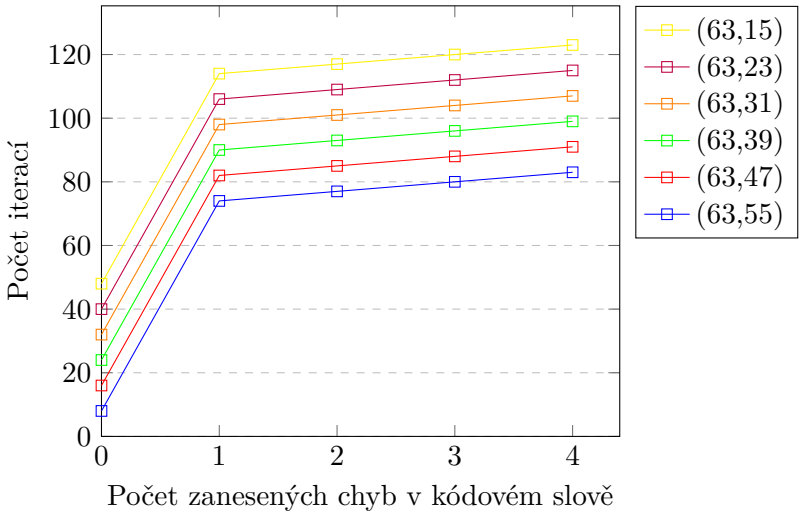
Obrázek 7.3: Iterační výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů při použití BM algoritmu.



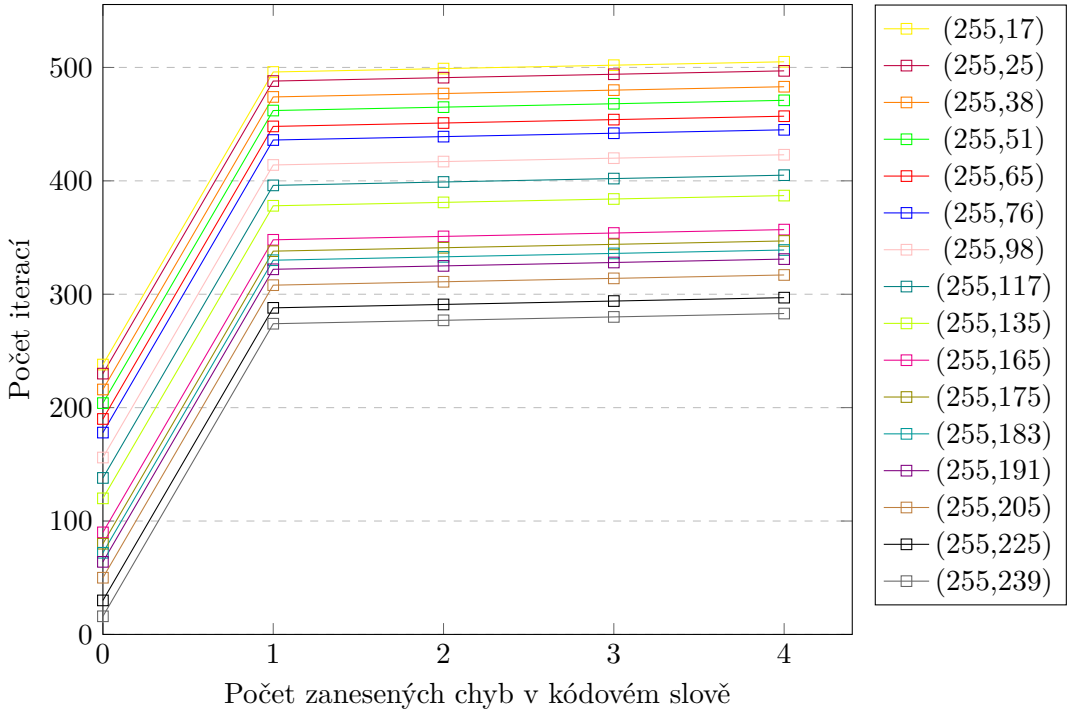
Obrázek 7.4: Iterační výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 255$ symbolů z určených kódových scénářů při použití BM algoritmu.

7.2 Experiment 2

Oproti prvnímu experimentu, tak zde se uvažuje dekodér Reed-Solomonova kódu pracující na Euklidově algoritmu, přičemž je zde opět měřena iterační výpočetní náročnost. Výsledky měření jsou pro délku kódového slova $n = 63$ symbolů zaneseny na grafu 7.5 a pro $n = 255$ symbolů vyobrazeny na grafu 7.6



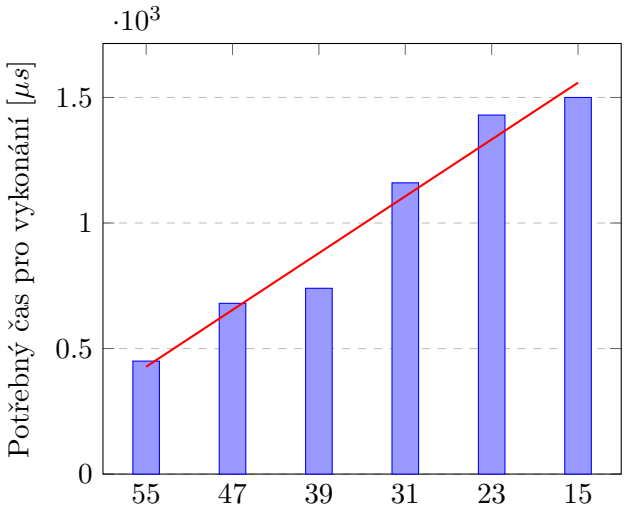
Obrázek 7.5: Iterační výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů při použití Euklidova algoritmu.



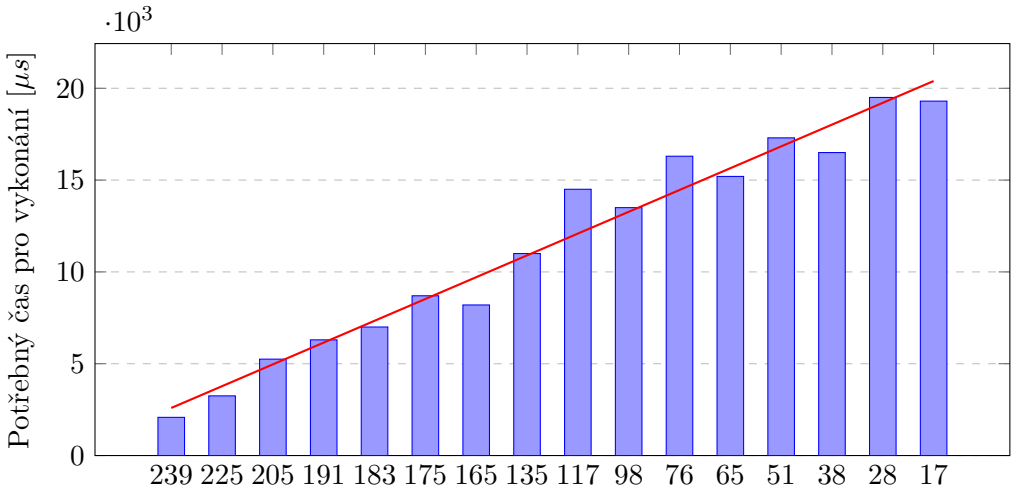
Obrázek 7.6: Iterační výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 255$ symbolů z určených kódových scénářů při použití Euklidova algoritmu.

7.3 Experiment 3

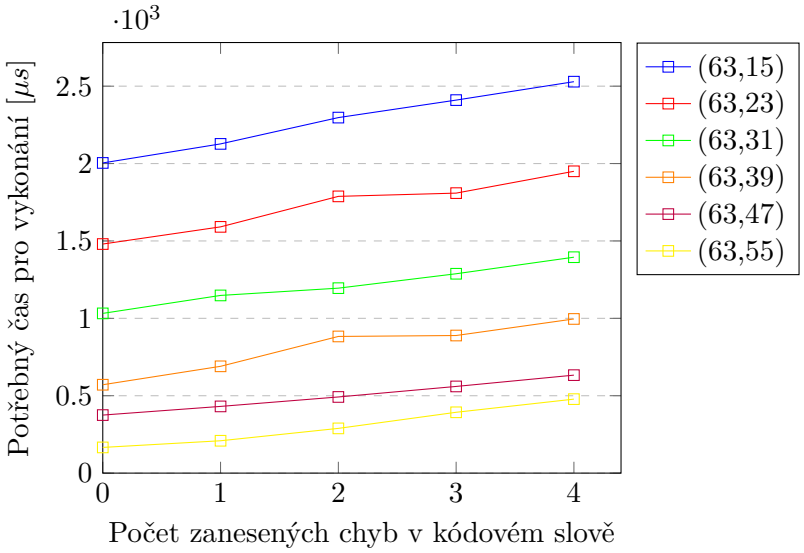
V aktuálním a v následujícím experimentu se měření výpočetní náročnosti bude zaměřovat na čas, respektive na to, jak dlouho trvá kódování a dekódování, přičemž se tento experiment týká dekódování pomocí BM algoritmu. Graf 7.7 zobrazuje časovou závislost kódování mezi různými délkami informačních částí a délkou kódového slova $n = 63$ symbolů. Zatímco graf 7.8 se zaměřuje na délku kódových slov $n = 255$ symbolů. Červená přímka na zmíněných grafech představuje regresní přímku. Je zde zobrazena pro přibližnou představu, jak rychle roste časová náročnost kódování se snižující se informační částí kódových slov. Pro časovou náročnost dekódování je vyhrazen graf 7.9 pro $n = 63$ a graf 7.10 pro $n = 255$ symbolů.



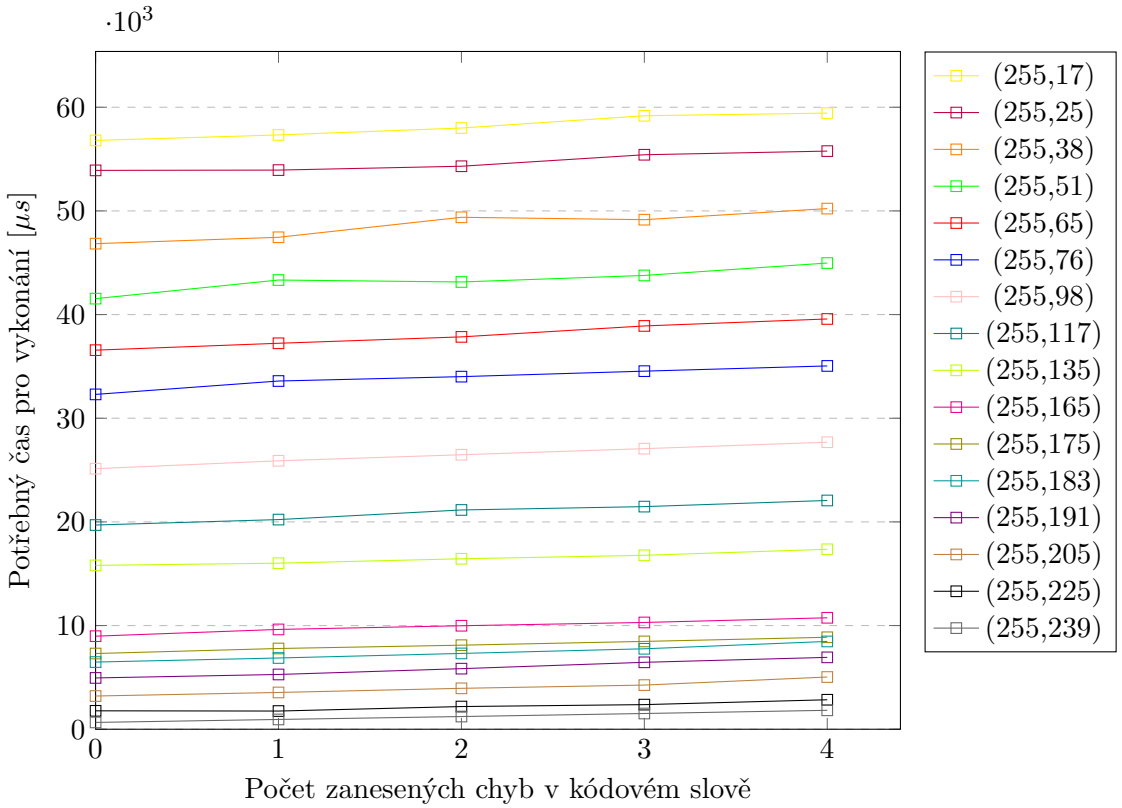
Obrázek 7.7: Časová výpočetní náročnost kódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů.



Obrázek 7.8: Časová výpočetní náročnost kódování RS kódů o délce kódového slova $n = 255$ symbolů z určených kódových scénářů.



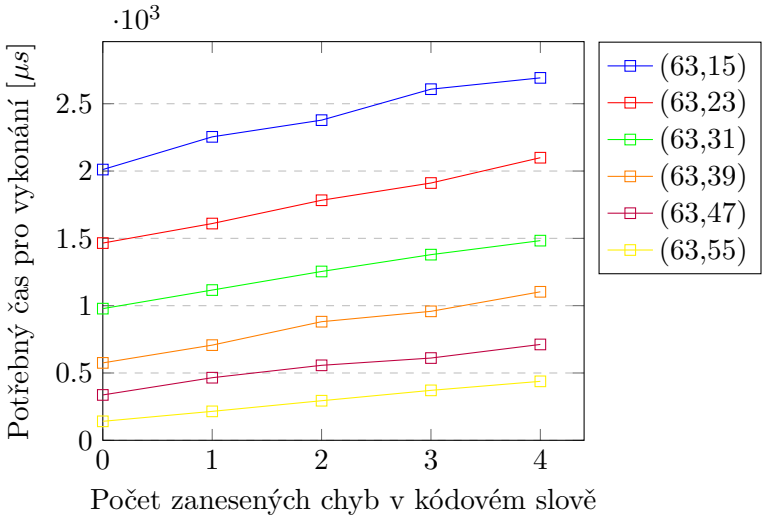
Obrázek 7.9: Časová výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů při použití BM algoritmu.



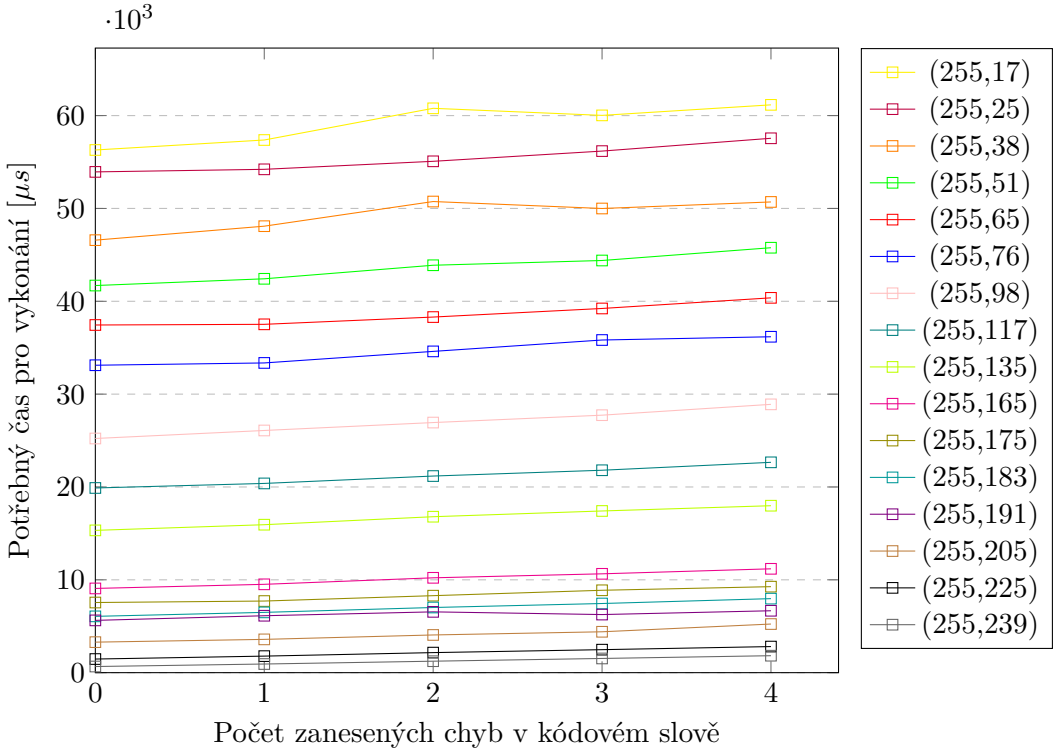
Obrázek 7.10: Časová výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 255$ symbolů z určených kódových scénářů při použití BM algoritmu.

7.4 Experiment 4

Poslední z experimentů se zaměřuje na dekódovací proces, jež využívá Euklidova algoritmu. Graf výsledků měření pro délku kódového slova $n = 63$ symbolů je 7.11 a pro $n = 255$ symbolů je 7.12.



Obrázek 7.11: Časová výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 63$ symbolů z určených kódových scénářů při použití Euklidova algoritmu.



Obrázek 7.12: Časová výpočetní náročnost dekódování RS kódů o délce kódového slova $n = 255$ z určených kódových scénářů při použití Euklidova algoritmu.

7.5 Diskuze výsledků měření

Z prvního experimentu je na grafech 7.1 a 7.2 nejvíce patrný jejich průběh. Ten je způsoben tím, že implementace vychází z popisu kodéru realizovaného pomocí hardwaru, viz obrázek 6.2. Takovýto dekodér by dokázal zakódovat zprávu v n taktech tedy v délce kódového slova. V softwarové realizaci je možné počet iterací části počítající paritní informaci vypočítat jako $2 * k * t$ (dle počtu iterací `for` cyklů). Ze znalostí získaných z teorie platí, že se zvětšující se délkou informační části klesá korekční schopnost kódu. Respektive korekční schopnost je závislá na délce kódového slova a délce informační části. Chování kodéru z grafu lze ukázat na následujícím příkladu

- RS(63,55) – $2 * k * t = 2 * 55 * \frac{63-55}{2} = 440$ iterací
- RS(63,31) – $2 * k * t = 2 * 31 * \frac{63-31}{2} = 992$ iterací
- RS(63,15) – $2 * k * t = 2 * 15 * \frac{63-15}{2} = 720$ iterací

Ovšem z časové náročnosti kódování ve třetím experimentu na grafech 7.7 a 7.8 je patrné, že se snižující se délkou informační části roste čas potřebný pro zakódování zprávy, i když se vykoná méně iterací, jak bylo popsáno výše. Je to způsobeno tím, že při malé délce informační části je potřeba vytvořit delší zabezpečovací část. Lze tedy říci, že důsledkem pomalejšího zpracování je práce s delším vektorem zabezpečovací části, kde je tato část tvořena jednotlivými symboly, jež byly spočteny kódovacím algoritmem a následně přidány do tohoto vektoru.

Co se týče porovnání dekódovacích algoritmů, tak lze dekódovat přijaté slovo pomocí Euklidova algoritmu na obecně nižším počtu iterací než pomocí BM algoritmu. Přičemž počet iterací vychází z principu dekódovaní popsaných v sekcích 5.7 a 5.8. Z časové náročnosti vychází i přes občasné výkyvy rychleji BM algoritmus. Jakožto u kódování, tak i u dekódování platí to, že se snižující se délkou kódových slov jsou oba procesy časově náročnější. V dekódovacím procesu je to primárně způsobeno vysokou mírou schopnosti korekce, tím i větším počtem charakteristik syndromů, se kterými se musí pracovat.

Kapitola 8

Závěr

Tato bakalářská práce se zabývala problematikou samoopravných kódů. Cílem práce byla implementace a analýza výpočetní náročnosti vybraného samoopravného kódu (zde Reed-Solomonova kódu), která byla splněna. Po obecném úvodu problematiky byly vypsány poznatky vysvětlující potřebné znalosti pro další práci. V pořadí se jednalo o popis Galoisových těles následovaný úvodem do samoopravných kódů. Posléze byla vysvětlena teorie Reed-Solomonových kódů a princip jejich kódování a dekódovaní. Jednotlivé kapitoly byly proloženy názornými příklady, které čtenáři pomohly lépe pochopit popisovanou teorii, přičemž od kapitoly vysvětlující Reed-Solomonovy kódy příklady na sebe navazovaly. Pokud by byly sepsány v souvislém celku, tak by popisovaly celý princip kódování a dekódování Reed-Solomonových kódů. Kapitola pojednávající o implementaci popisovala, jak jsem realizoval kodér s dekóderem a s případným zanesením chyb do kódového slova. S implementací byly v následující kapitole následně prováděny experimenty zaměřující se konkrétně na iterační a časovou náročnost. Výsledky měření byly zaznamenány ve formě grafů, po kterých následovala diskuze nad výsledky měření.

V práci se mi podařilo provést vlastní implementaci výpočtu prvků Galoisových těles společně s realizací operací nad nimi. Následně jsem implementoval dle popisu z literatury kodér Reed-Solomonova kódů, jehož výstupem je kódové slovo v systematickém tvaru a převzatý dekodér Reed-Solomonova kódu jsem upravil tak, aby vyhovoval výstupu mého kodéru a zároveň dokázal počítat s dříve vygenerovanými prvky konečného tělesa. Celková implementace dokáže v aktuálním stavu pracovat s maximální délkou kódových slov až 255 symbolů. Je teoreticky možné experimentovat s delšími kódovými slovy při určité modifikaci kódu.

Experimenty se zaměřovaly na Reed-Solomonovy kódy s délkou kódových slov 63 a 255 symbolů s různou délkou jejich informační části. Konkrétně se měření zaobíralo jejich iterační a časovou náročností. Z výsledků měření vyplynulo, že z hlediska, jak iterační, tak časové vychází lépe Berlekamp-Masseyův algoritmus. Byla také vysvětlena anomálie způsobující průběh grafu kodéru při měření iterační náročnosti. Z měření časové náročnosti bylo zjištěno, že s přibývajícími počty chyb v kódovém slově roste čas potřebný pro jejich opravu lineárně a také bylo zjištěno, že se snižující se délkou informační části roste potřebný čas pro zakódování a následné dekódování zprávy.

Při studiu teorie a zpracovávání bakalářské práce jsem se naučil lépe chápat principy samoopravných kódů, zejména Reed-Solomonových kódů. Dle mého názoru by mohl být text práce společně s mojí implementace přínosný v předmětu Systémy odolné proti poruchám – SPP, kdy by dychtivým studentům mohl pomoct k pochopení problematiky Reed-Solomonových kódů.

Na tuto bakalářskou práci bych mohl v budoucnu navázat implementací dalších dekódovacích algoritmů, respektive různých variant Berlekamp-Masseyho a Euklidova algoritmu, kde bych tuto implementaci rovněž podrobit měření a následnému porovnání jednotlivých dekódovacích procesů.

Literatura

[1] Adámek, J. *Kódování*. 1. vyd. Praha: Státní nakladatelství technické literatury, 1989. ISBN 04-005-89.

[2] Berlekamp, E. Nonbinary BCH decoding (Abstr.). *IEEE Transactions on Information Theory* [online]. . 1968, sv. 14, č. 2, s. 242–242, [cit. 2023-26-04]. DOI: 10.1109/TIT.1968.1054109. Dostupné z: https://doi.org/10.1109/TIT.1968.1054109.

[3] Berlekamp, E. R. *Algebraic Coding Theory – Revised Edition*. . USA: World Scientific Publishing Co., Inc., 2015. ISBN 978-981-4635-89-9.

[4] Bidlo, M. *Systémy odolné proti poruchám: Galoisova tělesa*. FIT VUT v Brně, 2023 [cit. 2023-04-09].

[5] Bidlo, M. *Systémy odolné proti poruchám: BCH kódy a Reed-Solomonovy kódy*. FIT VUT v Brně, 2023 [cit. 2023-04-10].

[6] Bose, R. a Ray Chaudhuri, D. On a class of error correcting binary group codes. *Information and Control*. . 1960, sv. 3, č. 1, s. 68–79. DOI: 10.1016/S0019-9958(60)90287-4. ISSN 0019-9958. Dostupné z: https://doi.org/10.1016/S0019-9958(60)90287-4.

[7] Dumas, J.-G., Roch, J.-L., Tannier, E. a Varrette, S. *Foundations of Coding: Compression, Encryption, Error Correction*. 1. vyd. Hoboken: Wiley Publishing, ©2015. ISBN 978-1-118-88144-6.

[8] Elghayyaty, M., Mouhib, O., Wahbi, A., Idrissi, A. E. H. E., Hlou, L. et al. Performance Comparison of New Designs of Chien Search and Syndrome Blocks for Bch and Reed Solomon Codes. *International Journal of Communication Networks and Information Security* [online]. . 2020, sv. 12, č. 2, s. 235–241, [cit. 2023-27-04]. DOI: 10.17762/ijcnis.v12i2.4562. Dostupné z: https://doi.org/10.17762/ijcnis.v12i2.4562.

[9] Geisel, W. *Tutorial on Reed-Solomon Error Correction Coding* [online]. Srpen 1990 [cit. 2023-04-04]. Dostupné z: https://ntrs.nasa.gov/citations/19900019023.

[10] Gustave Solomon, I. S. R. a. Polynomial Codes Over Certain Finite Fields. *Journal of The Society for Industrial and Applied Mathematics*. . 1960, sv. 8, , s. 300–304.

[11] Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*. . 1950, sv. 29, č. 2, s. 147–160, [cit. 2023-04-04]. DOI: 10.1002/j.1538-7305.1950.tb00463.x. Dostupné z: https://doi.org/10.1002/j.1538-7305.1950.tb00463.x.

[12] Hlavičková, I. *Lineární algebra: Cramerovo pravidlo, inverzní matice* [online]. FIT VUT v Brně, 2023 [cit. 2023-04-19]. Dostupné z: https://sites.google.com/vutbr.cz/ilgprednaskyhlavickova.

[13] Hocquenghem, A. Codes correcteurs d'erreurs. *Chiffres.* . Září 1959, sv. 2, č. 2, s. 147–156.

[14] Lajos Hanzo, T. H. L. a Yeap, B. L. *Turbo Coding, Turbo Equalisation and Space-Time Coding: for Transmission over Fading Channels.* 1. vyd. Chichester: Wiley Publishing, ©2002. ISBN 0-470-84726-3.

[15] Lin, S. a Costello, D. *Error Control Coding: Fundamentals and Applications.* 1. vyd. Englewood Cliffs: Prentice-Hall, ©1983. ISBN 00-13-283796-X.

[16] Lin, S. a Costello, D. *Error Control Coding: Fundamentals and Applications.* 2. vyd. Upper Saddle River: Pearson/Prentice-Hall, 2004. ISBN 0-13-017973-6.

[17] Massey, J. L. Shift-Register Synthesis and BCH Decoding. *IEEE TRANSACTIONS ON INFORMATION THEORY.* . 1969, IT-15, č. 1, s. 122–127, [cit. 2023-26-04]. DOI: 10.1109/TIT.1969.1054260. Dostupné z: https://doi.org/10.1109/TIT.1969.1054260.

[18] Moon, T. K. *Error correction coding: Mathematical Methods and Algorithms.* 1. vyd. Hoboken: Wiley Publishing, ©2005. ISBN 0-471-64800-0.

[19] Moreira, J. C. a Farrell, P. G. *Essentials of Error-Control Coding.* 2. vyd. Chichester: Wiley Publishing, ©2006. ISBN 978-0-470-02920-6.

[20] Morelos Zaragoza, R. H. *The Art of Error Correcting Coding.* 2. vyd. Chichester: Wiley Publishing, ©2006. ISBN 978-0-470-01558-2.

[21] Rurik, W. a Mazumdar, A. Hamming codes as error-reducing codes. In: . *2016 IEEE Information Theory Workshop (ITW).* 2016, s. 404–408 [cit. 2023-04-04]. DOI: 10.1109/ITW.2016.7606865. ISBN 978-1-5090-1090-5. Dostupné z: https://doi.org/10.1109/ITW.2016.7606865.

[22] Sanvicente, E. *Understanding Error Control Coding.* 1. vyd. Cham: Springer, 2019. ISBN 978-3-030-05839-5.

[23] Sklar, B. *Digital Communications: Fundamentals and Applications.* 2. vyd. New Jersey: Prentice Hall, 2001. ISBN 0-13-084788-7.

Příloha A

Obsah přiloženého paměťového média

- `bp/` ... Složka se soubory pro překlad bakalářské práce
- `src/` ... Složka se zdrojovými kódy
  - `Makefile` ... Makefile pro sestavení projektu
  - `CMakeLists.txt` ... Konfigurační soubor pro nástroj CMake
  - `ReedSolomon.hpp` ... Hlavičkový soubor obsahující deklarace funkcí
  - `*.cpp` ... Zdrojové soubory s implementací
  - `measurement/` ... Složka obsahující soubory realizující měření
    - `measure.py` ... Python skript pro měření výkonu
    - `RS.in` ... Vstupní soubor pro skript obsahující signatury RS kódů
- `README.md` ... Soubor popisující základní práci s programem
- `thesis.pdf` ... Bakalářská práce ve formátu PDF

Příloha B

Manuál

Zadáním následujícího příkazu do terminálu se spustí překlad celého programu

```
$ make
```

Výstupem je spustitelný soubor `ReedSolomon`, který lze spustit pomocí

```
$ make run
```

nebo

```
$ ./ReedSolomon
```

V obou zmíněných způsobech spouštění bude po uživateli požadováno, aby na vstup zadal hodnoty, jimiž bude parametrizovat aplikaci. Lze také parametrizovat aplikaci pomocí přepínačů, viz nápověda programu

```
$ ./ReedSolomon -h
```

Konkrétní příklad funkce programu lze ukázat na

```
$ make exampleBMA
```

nebo

```
$ make exampleEA
```

Na výstupu obou těchto příkazů bude vypsáno řešení příkladu kódování a dekódování Reed-Solomonova kódu, jenž jsou počítány v textu bakalářské práce.

Co se týká experimentů tak ty lze spustit pomocí příkazu

```
$ make exp{1-4}
```

kde za `exp` se nachází jedna číslice odpovídající konkrétnímu experimentu z kapitoly 7 bakalářské práce.

Přeložené binární soubory lze odstranit příkazem

```
$ make clean
```

Další nezmíněné detaily je možné nalézt v přiloženém souboru `README.md` nebo v nápovědě jednotlivých programů.

Seznam přepínačů programu ReedSolomon

- `-h` – Zobrazí nápovědu programu.
- `-r` – Povolí přepínače. Pokud tento přepínač není nastaven, tak budou vstupy programu čteny ze standardního vstupu.
- `-n` – Nastavuje celkovou délku kódového slova.
- `-k` – Nastavuje maximální délku zprávy, jež má být zakódována a přenesena.
- `-f` – Nastavuje cestu a název souboru do kterého budou vypsány výsledky měření.
- `-t` – Nastaví, aby se měřila časová náročnost programu.
- `-i` – Nastaví, aby se měřila iterační náročnost programu.
- `-b` – Dekódování bude probíhat pomocí BM algoritmu.
- `-a` – Dekódování bude probíhat pomocí Euklidova algoritmu.
- `-c` – Nastaví počet kolikrát se bude opakovat proces kódování a dekódování. Využitelné zejména při měření s přepínačem `-t`, jenž výsledky měření touto hodnotou zprůměruje.
- `-m` – Hodnoty za přepínačem oddělené mezerou určují jednotlivé části zprávy.
- `-e` – První hodnota určuje počet chyb, jež budou zaneseny do zakódovaného kódového slova. Za touto hodnotou následuje určitý počet dvojic (<pozice>, <chyba>), kde první hodnota udává pozici chyby ve slově a druhá hodnota udává hodnoty, jež bude na tuto pozici dosazena.

Seznam přepínačů měřicího skriptu

- `-h` – Vypíše nápovědu měřícího skriptu.
- `-e` – Slouží k nastavení maximálního počtu chyb, které budou zaneseny do kódového slova v programu.
- `-a` – Slouží k nastavení, aby program ReedSolomon dekódoval pomocí Euklidova algoritmu.
- `-b` – Slouží k nastavení, aby program ReedSolomon dekódoval pomocí BM algoritmu.
- `-t` – Slouží k nastavení, aby program ReedSolomon produkoval do souboru statistiku o časové výpočetní náročnosti použitých algoritmů.
- `-i` – Slouží k nastavení, aby program ReedSolomon produkoval do souboru statistiku o iterační výpočetní náročnosti použitých algoritmů.
- `-f` – Nastavuje cestu k vstupnímu souboru s Reed-Solomonovy kódy, respektive s parametry Reed-Solomonova kódu ve tvaru "RS(n,k)".
- `-o` – Nastavuje cestu k výstupnímu souboru, do kterého program ReedSolomon bude produkovat statistiky o výpočetní náročnosti.