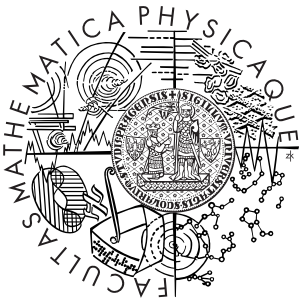
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Pavel Horal

## Reed-Solomonovy kódy a jejich aplikace

Katedra Algebry

Vedoucí bakalářské práce: Doc. RNDr. Aleš Drápal, CSc.
Studijní program: Matematika
Studijní obor: Matematické Metody Informační Bezpečnosti

2006

Chtěl bych poděkovat Doc. RNDr. Alešovi Drápalovi za odborné vedení a čas věnovaný kontrole práce.

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

V Praze dne 30.5. 2006 Pavel Horal

# Obsah

Název práce: Reed-Solomonovy kódy a jejich aplikace
Autor: Pavel Horal
Katedra (ústav): Katedra Algebry
Vedoucí bakalářské práce: Doc. RNDr. Aleš Drápal, CSc.
e-mail vedoucího: drapal@karlin.mff.cuni.cz

Abstrakt: Práce podává ucelenou definici klasických Reed-Solomonových kódů, včetně potřebných základů teorie kódů. Je dokázána cykličnost RS kódů délky $q-1$. Na cykličnosti jsou pak založeny tři prezentované dekódovací algoritmy (Petersonův, Berlekamp-Masseyův a Euklidův dekódovací algoritmus), včetně důkazů existence řešení. V poslední kapitole uvádím několik aplikací RS kódů, včetně nejznámějšího standardu CIRC používaného na hudebních CD.

Klíčová slova: samoopravný kód, ECC, Reed-Solomon, Peterson, Berlekamp-Massey, Euklid, CIRC

Title: Reed-Solomon codes and applications
Author: Pavel Horal
Department: Department of Algebra
Supervisor: Doc. RNDr. Aleš Drápal, CSc.
Supervisor's e-mail address: drapal@karlin.mff.cuni.cz

Abstract: This work presents compact definition of classic Reed-Solomon codes with necessary elements of coding theory. The cyclicity of RS codes of length $q-1$ is prooved and there are comletely described three decoding algorithms (Peterson's, Berlekamp-Massey and Euclid decoding algorithm) based on RS cyclicity. I also introduce a few RS codes applications in the last chapter, mainly the most popular CIRC standard, which is used on audio CDs.

Keywords: Error-Correcting Code, ECC, Reed-Solomon, Peterson, Berlekamp-Massey, Euclid, CIRC

# Kapitola 1

# Úvod

V druhé polovině dvacátého století přinesla digitalizace s sebou informační lavinu. Data je potřeba přenášet i uchovávat co nejefektivněji a nejrychleji, avšak chybná data svou cenu ztrácejí. Je proto třeba přejít k mechanismům, které jsou schopny chyby nejen odhalit, ale také odstranit. Takovými mechanismy jsou samoopravné kódy.

V roce 1944 publikoval Claude Shannon v rámci teorie informace *Shannonovu větu* ([2], str. 22), díky čemuž teorii kódů postavil na pevný základ, neboť dokázal existenci libovolně spolehlivých kódů. S příchodem informačních technologií se pak obor samoopravných kódů stal velmi populární a prožívá svůj rozvoj dodnes. Jako jeden z mála je úspěšnou aplikací nových objevů algebraických teorií v praxi.

Jedním z dnes nejrozšířenějších kódů jsou *Reed-Solomonovy kódy*. Reed-Solomonovy kódy byly poprvé publikovány I.S. Reedem a G. Solomonem v prosinci roku 1958 ve zprávě[1] *„Polynomiální kódy nad určitými konečnými tělesy.“* Dodnes se těší díky svým vlastnostem a „jednoduché“ definici velké oblibě. S objevem generujících polynomů pro cyklické kódy pak byly nalezeny nové dekódovací metody.

Tato práce prezentuje definici klasických Reed-Solomonových kódů a tři různé metody dekódování, založené na cykličnosti RS kódů. V poslední kapitole je popsána nejznámější aplikace RS kódů, konkrétně standard *CIRC*, používaný především na hudebních CD.

Česká terminologie je ustálena jen částečně, došel jsem proto k názoru, že práce tohoto typu by neměla aspirovat na zavedení českých ekvivalentů tam, kde se v odborné literatuře ještě nevyskytly. Proto některá anglická označení používám v češtině na pozici adjektiv. Naopak u většiny českých označení se snažím uvádět i anglické ekvivalenty.

[1] I.S. Reed, G. Solomon: *Polynomial codes over certain finite fields*, M.I.T. Lincoln Laboratory Group Report 47.23, 1958

# Kapitola 2

# Samoopravné kódy

## 2.1 Základní pojmy

Samoopravné kódy (dále jen SO kódy) jsou mechanismy pro ošetření chyb při přenosu informací. Abychom mohli kódy definovat a zkoumat jejich vlastnosti, je nejprve třeba přijmout model komunikace. Jak vypadá přenášená zpráva?

Informace je nejčastěji realizována jako řetězec *symbolů*. V počítačovém světě jsou to například jedničky a nuly (za symbol můžeme považovat také celých 8 bitů, tedy jeden byte - např. `ASCII` tabulka).

Při přenosu máme *zdroj zpráv* (*message source*) a *příjemce* (*recievер*). Přenos zprávy mezi nimi se děje přes *kanál*. Jako kanál můžeme uvažovat například radiové spojení nebo čtení hudebního CD laserem (kde zdroj je CD a příjemce zesilovač). Každému kanálu můžeme přiřadit jeho chybovost, t.j. pravděpodobnost s jakou nastane při přenosu jednoho symbolu chyba (přijatý symbol je různý od odeslaného). V tomto případě bereme přenosy symbolů jako vzájemně nezávislé jevy.

Přijímat data různá od odeslaných je určitě nežádoucí, a proto přicházejí na řadu SO kódy. Naším cílem je data zakódovat (nějakým způsobem přepsat) do formy, která by určitému počtu chyb odolala a na straně příjemce mohla být správně dekódována. Umožní nám to přidání nadbytečné informace (redundance). Tedy za cenu prodloužení informace jsme schopni nějak s chybami pracovat. Slovo „cena“ je použito schválně, neboť v reálu znamená více dat delší přenos a více peněz. U SO kódu nás zajímá o kolik data prodlouží (*rate*) a kolik chyb je schopen detekovat a opravit (*error correction capability*). V dnešní době musíme přidat další kritérium, a tím je efektivita a rychlost kódování/dekódování (snímky z Marsu se mohou dekódovat hodinu, u digitální televize by to pak patrně vadilo).

> **Příklad**: Jednoduchým SO kódem je binární opakovací kód. Každý bit několikrát zopakujeme, například třikrát: $0 \mapsto 000, 1 \mapsto 111$. Nastane-li jedna chyba, jsme schopni správně určit původní bit ($011 \mapsto 1$). Pokud však nastanou dvě chyby, budeme dekódovat špatně.

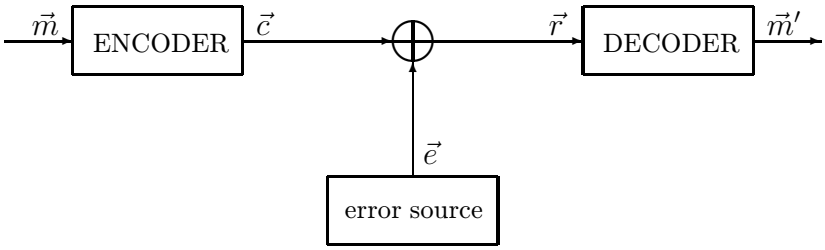
Zprávu, kterou chceme odeslat můžeme rozdělit na bloky o délce $k$. Každý jednotlivý blok pak před odesláním zakódujeme, čímž se prodlouží na $n$ symbolů. Čísla $k$ a

$n$ jsou plně určena použitým kódem. Kód, jehož kódové slovo závisí pouze na odpovídajících $k$ informačních symbolech se nazývá *blokový* (naproti tomu u *konvolučních* kódů závisí i na předešlých blocích/symbolech).

Blok informace délky $k$, kterou chceme předat, se obvykle značí $\vec{m}$. Blok kódované informace délky $n$ je $\vec{c}$. Přijaté slovo (může se lišit od odeslaného $\vec{c}$) označíme $\vec{r}$. Po dekódování přijatého slova $\vec{r}$ obdržíme kandidáta na slovo odeslané $\vec{m}'$. Pravděpodobnost $P[\vec{m} = \vec{m}']$ se nazývá spolehlivost kódu (a právě o ní dokázal C.E. Shannon, že existují libovolně spolehlivé kódy).

Pokud na symbolech lze zavést operace $+, -, \cdot, /$ tak, aby množina všech symbolů tvořila těleso, můžeme na kódová slova pohlížet jako na prvky vektorového prostoru. V dalším textu budeme pracovat se symboly pouze jako s prvky nějakého konečného tělesa $\mathbb{F}_q$.

Definujeme chybu (*error pattern*) jako $\vec{e} = \vec{r} - \vec{c}$. Celé popsané schéma je vidět na obrázku 1.




**Obr. 1:** Schéma přenosu informace s použitím kódování

Tvar chybového vektoru $\vec{e}$ plně závisí na vlastnostech kanálu. Důležitou informací je pro nás, kolik má nenulových symbolů nebo též, v kolika symbolech se liší odeslané kódové slovo od přijatého.

**Definice 2.1.1** *Hammingova vzdálenost (dále jen vzdálenost) $d(\vec{x}, \vec{y})$ dvou vektorů $\vec{x}$ a $\vec{y}$ je rovna počtu souřadnic, ve kterých se liší.*

**Definice 2.1.2** *Hammingova váha (dále jen váha) $w(\vec{x})$ vektoru $\vec{x}$ je rovna počtu nenulových souřadnic.*

**Příklad**: $d(1000111, 1010110) = 2$, $w(1011010) = 4$

Pokud tedy obdržíme slovo $\vec{r}$, je úkolem dekodéru nalézt takové kódové slovo $\vec{c}$, aby vzdálenost $d(\vec{r}, \vec{c})$ byla nejmenší možná. Dekódování na nejbližší kódové slovo se nazývá *Maximum Likehood Decoding*. Chyba dekódování nastane tehdy, má-li přijaté slovo blíže k jinému než odeslanému kódovému slovu. Dobrou charakteristiku kódu proto uvádí následující definice.

**Definice 2.1.3** *Minimální vzdálenost $d_{min}$ kódu $C$ je rovna $d_{min} = \min\{d(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in C\}$ (obvykle se uvádí bez indexu $_{min}$).*

Nechť má kód $C$ minimální vzdálenost $d = 2t+1$. Pokud vzdálenost $d(\vec{c}, \vec{r}) \leq t$ pro nějaké $\vec{c} \in C$, nemůže pro žádné jiné kódové slovo $\vec{c}' \in C$ být vzdálenost $d(\vec{c}', \vec{r}) \leq t$, jinak by totiž muselo platit $d(\vec{c}, \vec{c}') \leq 2t$, a to by byl spor s definicí minimální vzdálenosti.

Vidíme, že pokud při přenosu nastane méně než $t$ chyb (vzdálenost odeslaného a přijatého kódového slova bude menší než $t$), jsme schopni správně dekódovat. O tom, kolik chyb jsme schopni opravit tedy rozhoduje minimální vzdálenost.

**Definice 2.1.4** *O kódu $C$ s délkou bloku kódované informace $k$, délkou kódového slova $n$ a minimální vzdáleností $d$ říkáme, že je to $(n, k, d)$ kód.*

**Definice 2.1.5** *O dvou kódech $C$ a $C'$ říkáme, že jsou navzájem ekvivalentní, pokud se kódová slova liší pouze v pořadí symbolů.*

## 2.2 Lineární kódy

Jak informaci kódovat nebo dekódovat, není vůbec jasné. Použití aparátu lineární algebry takové metody nabízí. Důležitou třídou jsou proto *lineární kódy.*

**Definice 2.2.1** *O kódu $C$ nad tělesem $\mathbb{F}_q$ s parametry $(n, k, d)$ říkáme, že je lineární, pokud $C$ (množina kódových slov) je vektorovým podprostorem prostoru $\mathbb{F}_q^n$.*

Parametry lineárních kódů se obvykle uvádí v hranatých závorkách – je-li řeč o $[n, k, d]$ kódu, jde o kód lineární.

**Tvrzení 2.2.2** *Minimální vzdálenost $d_{min}$ lineárního kódu $C$ je rovna minimální váze*

$$d_{min} = w_{min} = min\{w(\vec{c}) \mid c \in C\} \tag{2.1}$$

**Důkaz**: Triviálně dostáváme $d(\vec{c}, \vec{c}') = w(\vec{c} - \vec{c}')$, kde $\vec{c} - \vec{c}' \in C$ □

Dimenze lineárního kódu $C$ je nutně rovna $k$ ($|C| = q^k$). Z $C$, jakožto vektorového podprostoru, můžeme vybrat bázi a pomocí ní sestrojit homomorfismus $\mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$. Obrazem homomorfismu bude právě $C$. Matice takového homomorfismu se značí $G$ a nazývá se *generující matice* kódu $C$.

**Definice 2.2.3** *Matici $G$ tvaru $(k \times n)$, jejíž řádky tvoří nějakou bázi $C$, nazýváme* generující maticí.

Kódování zprávy lineárním kódem je snadné. Stačí vynásobit generující matici $G$ zprávou $\vec{m}$:

$$\vec{c} = \vec{m}G. \tag{2.2}$$

Libovolnou elementární transformací na řádkách matice $G$ dostaneme generující matici $G'$ kódu $C$. Vynásobením řádku $G$ prvkem z $\mathbb{F}_q$ nebo přičtením jednoho řádku k druhému pouze obdržíme jinou bázi $C$.

**Definice 2.2.4** *O $(n, k, d)$ kódu $C$ řekneme, že je systematický, pokud pro $\vec{m} = (m_1, \ldots, m_k)$ je odpovídající kódové slovo tvaru $\vec{c} = (m_1, \ldots, m_k, p_1, \ldots, p_{n-k})$. Prvních $k$ symbolů pak nazýváme informační a zbylých $n - k$ paritní.*

Generující matice lineárního systematického kódu má tvar $G = (IP)$, kde $I$ je jednotková matice $(k \times k)$ a $P$ nějaká matice $(k \times (n-k))$. Systematický kód má tu výhodu, že pokud při přenosu nenastane žádná chyba, jsme ihned bez dekódování schopni vyčíst informaci. Pokud je kód $C$ ekvivalentní nějakému systematickému kódu, bývá lepší užít systematického kódu.

**Tvrzení 2.2.5** *Každý lineární kód je ekvivalentní nějakému systematickému kódů.*

**Důkaz**: Generující matice $G$ kódu $C$ s parametry $[n, k, d]$ obsahuje $k$ lineárně nezávislých vektorů báze podprostoru $C$, tedy její hodnost je $k$. Protože $h(G) = h(G^T) = k$ můžeme z G vybrat $k$ lineárně nezávislých sloupců a přesunout je na začátek matice. Gaussovou eliminací vytvoříme hledanou matici ekvivalentního systematického kódu $C'$ tvaru $G' = (IP)$. □

## 2.3 Dekódování lineárních kódů

U systemaických kódů bychom chtěli mít možnost rychle ověřit, že nenastala žádná chyba (pak můžeme ihned infomaci vyčíst). Potřebujeme nějaké zobrazení na slovech, kde podle obrazu jednoznačně zjistíme, zda je nebo není slovo z kódu $C$. Můžeme zkusit najít takovou lineární kombinaci souřadnic slova, která bude právě pro kódová slova nulová.

$$\sum_{i=1}^{n} a_i v_i = \begin{cases} 0 & \text{pro } \vec{v} \in C \\ \neq 0 & \text{pro } \vec{v} \notin C \end{cases} \tag{2.3}$$

Uvedená lineární kombinace bude přesně určena svými koeficienty $\vec{a} = (a_1, \ldots, a_n)$. Jde o zobrazení $\mathbb{F}_q^n \to \mathbb{F}_q$ a nazývá se *lineární forma*. Abychom nalezli její koeficienty $\vec{a}$, musíme řešit soustavu

$$G\vec{a}^T = 0. \tag{2.4}$$

Máme soustavu (2.4) $k$ lineárně nezávislých rovnic pro $n$ neznámých. Řešením je tedy podprostor[1] $\mathbb{F}_q^n$ dimenze $n - k$. Podprostor se značí $\tilde{C}$ a je to *duální kód*[2] kódu $C$.

Vztah parametrů kódu $C$ k parametrům svého duálního kódu $\tilde{C}$ jsou vcelku hluboké. Pro nás je důležitá především generující matice duálního kódu $\tilde{C}$ (dimenze $n-k$), tedy matice obsahující bázi prostoru všech lineárních forem splňujících (2.4).

**Definice 2.3.1** *Matici $P$, jejíž řádky tvoří nějakou bázi duálního kódu $\tilde{C}$, nazýváme* paritní maticí *kódu $C$.*

[1] kokrétně podprostor *duálního prostoru* $Hom(\mathbb{F}_q^n, \mathbb{F})$, tedy prostoru všech lineárních forem na $\mathbb{F}_q^n$ – viz. [6]

[2] Definice duálních kódů se obvykle uvádí zjednodušeně za pomoci pseudo-skalárního součinu, jako kód $C^\perp$ kolmý na kód $C$ – viz. [5]

Paritní matice má hledanou vlastnost

$$P\vec{v}^T = \begin{cases} 0 & \text{pro } \vec{v} \in C \\ \neq 0 & \text{pro } \vec{v} \notin C \end{cases} \tag{2.5}$$

Pomocí paritní matice jsme schopni rychle určit, zda přijaté kódové slovo je kódovým slovem, tedy zda s velkou pravděpodobností nenastala chyba. Pokud pro přijaté slovo $\vec{r}$ je $P\vec{r} = 0$, platí $\vec{r} = \vec{c}' \in C$. Co ale, když je přijaté slovo tvaru $\vec{r} = \vec{c} + \vec{e}$? Potom dostáváme $P\vec{r}^T = P\vec{c}^T + P\vec{e}^T = P\vec{e}^T = \vec{s}$. Výsledek vynásobení paritní matice přijatým slovem se nazývá *syndrom*. Je vidět, že syndrom závisí pouze na chybě, nikoliv na odesílaném slově.

Syndrom je vektor o $(n-k)$ souřadnicích, tedy pro $[n,k,d]$ kód nad $\mathbb{F}_q$ máme celkem $q^{n-k}$ možných syndromů. Můžeme vytvořit převodní tabulku, kde každému syndromu $\vec{s}$ přiřadíme chybový vektor $\vec{e}$ s nejmenší vahou takový, aby $P\vec{e}^T = \vec{s}$. Máme jednoduchý návod, jak obecně dekódovat lineární kódy (na nejbližší kódové slovo). Velikou nevýhodou je rozsáhlá převodní tabulka obsahující $q^{n-k}$ syndromů a jim odpovídajících chybových vektorů.

## 2.4 Cyklické kódy

Vektor $\vec{v} = (v_0, \ldots, v_{n-1})$ můžeme reprezentovat také pomocí polynomů stupně menšího než $n$, neboť aditivní operace a násobení skalárem jsou v obou reprezentacích identické.

**Definice 2.4.1** *Pro vektor $\vec{v}$ definujme jeho polynomiální reprezentaci $v(x)$:*

$$\vec{v} = (v_0, v_1, \ldots, v_{n-1}) \longleftrightarrow v_0 + v_1 x + \ldots + v_{n-1}x^{n-1} = v(x) \tag{2.6}$$

Nová reprezentace kódových slov umožňuje násobení dvou slov/polynomů, díky kterému budeme schopni popsat novou třídu kódů.

**Definice 2.4.2** *Cyklickým posunem souřadnic vektoru $\vec{v}$ je operace:*

$$\vec{v} = (v_0, \ldots, v_{n-2}, v_{n-1}) \longrightarrow \vec{v}' = (v_{n-1}, v_0, \ldots, v_{n-2})$$

**Definice 2.4.3** *Lineární kód $C$ nazveme cyklickým, pokud spolu s každým kódovým slovem $c \in C$ obsahuje i jeho cyklický posun.*

V polynomiální reprezentaci odpovídá cyklický posun násobení slova $c(x)$ polynomem $x$ modulo $(x^n - 1)$. Tedy

$$\begin{aligned} xc(x) &= x(c_0 + \ldots + c_{n-2}x^{n-2} + c_{n-1}x^{n-1}) \equiv \\ &\equiv c_{n-1} + c_0 x + \ldots + c_{n-2}x^{n-1} \mod (x^n - 1). \end{aligned}$$

Lze snadno nahlédnout, že násobením $x^i$ získáme cyklický posun o $i$ souřadnic. Násobení kódového slova libovolným polynomem vypadá následovně:

$$a(x)c(x) = \Sigma_{i=0}^{deg(a)} a_i x^i c(x).$$

V řeči vektorů tedy nejde o nic jiného, než o lineární kombinaci cyklických posunů kódového slova $\vec{c}$. Protože cyklický kód $C$ obsahuje s kódovým slovem i jeho cyklické posuny, tak $a(x) \cdot c(x) \in C$. Vidíme, že cyklický kód je ideálem v $\mathbb{F}_q[x]/(x^n - 1)$.

Jsme v oboru hlavních ideálů, a proto musí existovat i generující polynom ideálu/kódu $C$. V polynomiální reprezentaci můžeme zvolit monický[3] polynom nejmenšího stupně $g(x) \in C$. Takový polynom musí být určen jednoznačně, protože pokud by existoval jiný polynom $g'(x)$, mohli bychom odečtením polynomu $g'(x)$ od $g(x)$ získat polynom stupně menšího, což by bylo ve sporu s výběrem $g(x)$.

Polynom $g(x)$ z předchozího odstavce také dělí všechny kódové polynomy $c(x) \in C$. Polynom $c(x)$ můžeme vydělit se zbytkem polynomem $g(x)$: $c(x) = q(x)g(x) + r(x)$, kde $deg(r) < deg(g)$. Platí $c(x) - q(x)g(x) \in C \Rightarrow r(x) \in C$, a tak $r(x) = 0$, jinak by to byl spor s výběrem $g(x)$. Můžeme vyslovit tvrzení (již bez důkazu):

**Tvrzení 2.4.4** *Pro cyklický kód $C$ je monický polynom $g(x) \in C$ nejmenšího stupně generující polynom kódu $C$. Takový polynom je určen jednoznačně a pro každé $c(x) \in C$ platí $g(x) \mid c(x)$.*

**Tvrzení 2.4.5** *Generující polynom $g(x)$ cyklického $[n, k, d]$ kódu dělí polynom $x^n - 1$.*

**Důkaz**: $x^n - 1$ můžeme vydělit se zbytkem $x^n - 1 = q(x)g(x) + r(x)$. Tedy $0 - q(x)g(x) \equiv r(x)\ mod(x^n - 1)$. Tedy $r(x) = 0$, jinak dojde ke sporu s minimalitou $g(x)$, protože $q(x)g(x) \in C$. □

**Tvrzení 2.4.6** *Generující polynom $g(x)$ cyklického kódu $C - [n, k, d]$ má stupeň* $\deg(g) = n - k$.

**Důkaz**: Nechť $\deg(g) = r$. Díky předchozímu tvrzení víme, že existuje polynom $h(x)$ stupně $\deg(h) = n - r$, pro který platí $x^n - 1 = h(x)g(x)$. Kódový polynom $c(x)$ je dělitelný $g(x)$. Můžeme psát $c(x) = a(x)g(x)$, kde stupeň $a$ bude nejmenší možný. Pokud bude $\deg(a) > n - r$, mohli bychom dělit se zbytkem $a(x) = q(x)h(x) + r(x)$, kde $\deg(r) < deg(h)$. Dosazením získáme

$$c(x) = a(x)g(x) = (q(x)h(x) + r(x))g(x) = g(x)r(x).$$

Dostali jsme spor s minimálním stupňem $a(x)$, tedy platí $\deg(a) \leq n - r$. Všech možných $a(x)$ je $q^{n-r}$. Protože máme přesně $q^k$ kódových slov, musí být $n - r = k$ a $r = n - k$. □

Tvrzení 2.4.6 dává i bázi kódu $C$, neboť víme, že násobek $a(x)g(x)$ je lineární kombinací cyklicky posunutého $g(x)$. Generující matice cyklického kódu s generujícím polynomem $g(x)$ má tvar:

[3] Koeficient u nejvyšší mocniny je roven jedné.

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & \cdots & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} \end{pmatrix} \tag{2.7}$$

Generující polynom dělí každý kódový polynom. To znamená, že kořeny generujícího polynomu jsou i kořeny každého kódového slova. Pokud prvky $\alpha_1, \ldots, \alpha_{n-k}$ z nějakého rozšíření $\mathbb{F}_q$ jsou kořeny $g(x)$, tak jejich dosazením do přijatého polynomu $r(x)$ zjistíme, zda je kódovým polynomem. Dosazení do polynomu můžeme zapsat maticově, a tak sestrojit paritní matici cyklického kódu[4]:

$$P = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \alpha_1^3 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^3 & \cdots & \alpha_2^{n-1} \\ & \vdots & & & \vdots & \\ 1 & \alpha_{n-k} & \alpha_{n-k}^2 & \alpha_{n-k}^3 & \cdots & \alpha_{n-k}^{n-1} \end{pmatrix} \tag{2.8}$$

Jak ukáži později, tato matice povede k novým dekódovacím možnostem.

## 2.5 Součinové kódy

Máme-li dva samoopravné kódy $C_1 - (n_1, k_1, d_1)$ a $C_2 - (n_2, k_2, d_2)$ naskýtá se otázka, zda bychom je mohli nějakým způsobem spojit a třeba i dostat jisté lepší vlastnosti. Jednou možností, jak spojení provést, jsou *součinové kódy* (*product codes*).

Součinové kódy se obvykle popisují ve dvou rozměrech (maticově). Vždy kódujeme najednou $k = k_1 k_2$ symbolů. Zprávu uspořádáme po řádcích do matice $(k_2 \times k_1)$:

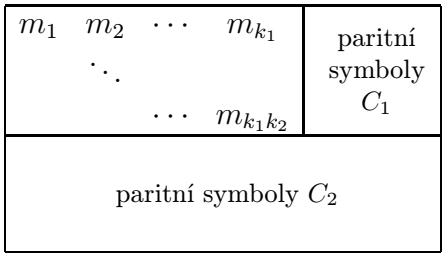
$$\begin{pmatrix} m_1 & m_2 & \cdots & m_{k_1} \\ m_{k_1+1} & m_{k_1+2} & & m_{2k_1} \\ & \ddots & & \\ & & \cdots & m_{k_2 k_1} \end{pmatrix}$$

Jako první zakódujeme jednotlivé řádky matice kódem $C_1$. Dostaneme $k_2$ kódových slov délky $n_1$. Vše necháme uspořádané v matici, nyní již tvaru $(k_2 \times n_1)$:

$$\begin{pmatrix} c_1^{(1)} & c_2^{(1)} & \cdots & c_{n_1}^{(1)} \\ c_1^{(2)} & c_2^{(2)} & & c_{n_1}^{(2)} \\ & \ddots & & \\ c_1^{(k_2)} & c_2^{(k_2)} & \cdots & c_{n_1}^{(k_2)} \end{pmatrix} \tag{2.9}$$

Nakonec postupně zakódujeme sloupce vzniklé matice kódem $C_2$. Výledkem je matice tvaru $(n_2 \times n_1)$, kterou můžeme považovat za kódové slovo délky $n = n_1 n_2$. Obrázek 2 ukazuje výslednou matici při použití dvou systematických kódů.

[4]Polynom $g(x)$ se nemusí vždy rozkládat v $\mathbb{F}_q$ na kořenové činitele, ale až v nějakém jeho algebraickém rozšíření. Jde proto o paritní matici v poněkud jiném smyslu, než bylo uvedeno výše.




**Obr. 2** Schéma součinového kódu pro systematické kódy $C_1$ a $C_2$

**Tvrzení 2.5.1** *Součinový kód, tvořený $(n_1, k_1, d_1)$ kódem $C_1$ a $(n_2, k_2, d_2)$ kódem $C_2$, je $(n_1n_2, k_1k_2, d_1d_2)$ kód.*

**Důkaz**: Délka kódového i zdrojového slova jasně plyne z popisu součinového kódu. Minimální vzdálenost budeme muset dokázat.

Výchozím bodem je určení minimální vzdálenosti matice (2.9), vytvořené po kódování kódem $C_1$. Pokud zaměníme ve zprávě jeden symbol, změní se odpovídající řádek alespoň v $d_1$ symbolech. Minimální vzdálenost matic z (2.9), odpovídajících různým zprávám, je proto $d_1$. Minimální vzdálenost $d_1$ nastává alespoň v jednom řádku, a tak při kódování sloupců kódem $C_2$, se pro dvě různé zprávy bude lišit nejméně v $d_1$ sloupcích. Minimální vzdálenost dvou kódových slov kódu $C_2$ je $d_2$, a protože se vždy liší alespoň $d_1$ slov, dostáváme minimální vzdálenost součinového kódu $d_{min} = d_1d_2$. □

Součinové kódy jsou odolné vůči některým chybovým vzorům. Pokud umí kód $C_1$, resp. $C_2$, opravit $t_1 = (d_1 - 1)/2$, resp. $t_2 = (d_2 - 1)/2$, chyb, jsme schopni v součinovém kódu zprávu správně dekódovat i pokud obdržíme $t_1$ sloupců a $t_2$ řádků zcela chybných. To je o mnoho více než dostáváme z minimální vzdálenosti $d_1d_2$. Získali jsme vysokou odolnost proti *dávkovým chybám*[5]. Zároveň máme složitost dekódování stejnou jako u relativně krátkých kódů $C_1$ a $C_2$.

## 2.6 MDS kódy

**Tvrzení 2.6.1** (Singletonův odhad) *Pro blokový kód $C$ s parametry $(n, k, d)$ platí*

$$d \leq n - k + 1 \tag{2.10}$$

**Důkaz**: Odstraníme-li v kódu $d-1$ souřadnic, tak se nezmění jeho velikost $k$ (vzdálenost slov je nejhůře rovna jedné). Tedy po odstranění $d - 1$ souřadnic musí zbýt alespoň $k$ jiných souřadnic.

$$n - (d - 1) \geq k \implies d \leq n - k + 1$$

□

[5]Jedná se o chyby přicházející v dávkách (*burst error*), tedy souvislé chyby na několika symbolech za sebou – pro většinu reálných kanálů jsou velmi pravděpodobné.

Kódy splňující v (2.10) rovnost se nazývají *MDS* (*Maximum Distance Separable*) kódy.

# Kapitola 3

# Reed-Solomonovy kódy

## 3.1 Konstrukce RS kódů

*Reed-Solomonovy kódy* jsou konstruovány nad obecným konečným tělesem $\mathbb{F}_q$. Počet prvků takového tělesa je $q = p^m$, pro nějaké prvočíslo $p$. Důležitou vlastností konečného tělesa je cykličnost jeho multiplikativní podgrupy. Jinými slovy existuje *primitivní prvek* $\alpha \in \mathbb{F}_q$, jehož mocniny $\alpha^i$, pro $i = 0, \ldots, q-2$, generují všechny nenulové prvky $\mathbb{F}_q$.

Pokud vezmeme informaci $\vec{m}$ délky $k \leq q-1$, můžeme z ní sestrojit polynom stupně menšího než $k$, tvaru

$$m(x) = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1}. \tag{3.1}$$

Kódové slovo délky $n = q-1$ je pak tvořeno valuacemi polynomu $m(x)$ ve všech nenulových bodech tělesa $\mathbb{F}_q$. Klasická definice RS kódů vypadá následovně:

**Definice 3.1.1** *Nechť je $\alpha \in \mathbb{F}_q$ nějaký primitivní prvek. Reed-Solomonův*[1] *($RS_{q,k}$) kód je tvořen slovy tvaru:*

$$(f(\alpha^0 = 1), f(\alpha^1 = \alpha), f(\alpha^2), \ldots, f(\alpha^{q-2})), \tag{3.2}$$

*kde $f(x)$ probíhá všechny polynomy stupně menšího než $k$, $0 < k \leq q-1$.*

Jedná se o kód lineární, neboť nulový vektor získáme valuací nulového polynomu a součet dvou valuací je opět valuace: $f(\alpha^i) + \hat{f}(\alpha^i) = (f + \hat{f})(\alpha^i)$.

Libovolný polynom $f(x)$, kde $\deg(f) \leq k-1$, můžeme obdržet jako lineární kombinaci polynomů $1, x, x^2, \ldots, x^{k-1}$. Valuace těchto polynomů zcela jistě generují celý $RS_{q,k}$. Protože polynom $f(x)$, stupně nejvýše $k-1$, je jednoznačně určen svými $k \leq q-1$ funkčními hodnotami, žádné kódové slovo délky $q-1$ nemůže být valuací dvou různých polynomů, stupně nejvýše $k-1$. Dimenze $RS_{q,k}$ kódu je proto $k$ a bázi

[1] Takto definované RS kódy délky $q-1$ se označují jako *klasické*.

tak tvoří valuace polynomů $1, x, x^2, \ldots, x^{k-1}$. Generující matice má tedy tvar:

$$G = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(q-2)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \cdots & \alpha^{(k-1)(q-2)} \end{pmatrix} \tag{3.3}$$

Polynom $f(x)$, stupně nejvýše $k-1$, může mít nejvýše $k-1$ kořenů. Každé kódové slovo proto obsahuje nejvýše $k-1$ souřadnic rovných nule. Minimální váha, a tedy i vzdálenost, musí splňovat $w = d \geq n-k+1$. Díky Singletonově odhadu (2.10) však platí rovnost. Minimální vzdálenost kódu je $d = n-k+1$, jedná se proto o MDS kód (oddíl 2.6). Je dobré nahlédnout, že pokud ubereme kódu některé souřadnice (zkrátíme ho), tak MDS vlastnost zůstane[2].

**Tvrzení 3.1.2** *Reed-Solomonův kód definovaný v 3.1.1 je cyklický.*

**Důkaz**: Stačí ukázat, že pro kódové slovo $(f(\alpha^0), f(\alpha^1), \ldots, f(\alpha^{q-2}))$ je kódovým slovem i jeho cyklický posun $(f(\alpha^1), \ldots, f(\alpha^{q-2}), f(\alpha^0 = \alpha^{q-1}))$. Valuace polynomu $f(x)$ v bodě $\alpha^l$ je rovna $f(\alpha^l) = \Sigma_{i=0}^{k-1} f_i(\alpha^l)^i$. Cyklický posun můžeme proto realizovat jako valuace polynomu $\hat{f}(x) = \Sigma_{i=0}^{k-1} \frac{f_i}{\alpha^i} x^i$. Pro nový polynom platí $\hat{f}(\alpha^{l+1}) = \Sigma_{i=0}^{k-1} \frac{f_i}{\alpha^i} (\alpha^{l+1})^i = \Sigma_{i=0}^{k-1} f_i (\alpha^l)^i = f(\alpha^l)$. □

Uvedená vlastnost dovoluje použít znalostí a postupů z cyklických kódů. RS kód je hlavním ideálem v okruhu $\mathbb{F}_q[x]/(x^n - 1)$. Jak vypadá jeho generující polynom $g(x)$?

Generující polynom $g(x)$ dělí $x^n - 1 = x^{q-1} - 1$. Polynom $x^{q-1} - 1$ má právě $q-1$ kořenů a tím jsou nenulové prvky $\mathbb{F}_q$. Polynom $x^{q-1} - 1$ se tedy (stejně jako $g(x)$) rozkládá v $\mathbb{F}_q$ na lineární činitele. Nemusíme se proto starat o žádné algebraické rozšíření a stále pracujeme jen v $\mathbb{F}_q$. Generující polynom můžeme sestrojit jako součin $g(x) = \prod_{i=1}^{n-k} (x - \alpha_i)$ pro nějaké navzájem různé prvky $\alpha_1, \ldots, \alpha_{n-k}$. Otázkou zůstává, zda námi sestrojený $g(x)$ generuje právě $RS_{q,k}$. Ve skutečnosti generuje MDS kód se stejnými parametry.

Pro námi definovaný RS kód je generující polynom tvaru:

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) = \sum_{i=0}^{n-k} g_i x^i. \tag{3.4}$$

Každý kódový polynom (polynom odpovídající kódovému slovu) je násobkem generujícího polynomu. Kořeny $g(x)$ musejí být tedy i kořeny kódového polynomu. Proto lze snadno zjistit, zda je přijaté slovo kódovým slovem ohodnocením odpoví-

[2] Ve své původní konstrukci byly RS kódy valuacemi ve všech bodech tělesa $\mathbb{F}_q$ (tedy navíc v nule). Naopak zkrácené kódy jsou oblíbené pro menší nároky na dekodér.

dajícího polynomu. Můžeme rovněž sestrojit paritní matici $RS_{q,k}$ kódu:

$$P = \begin{pmatrix} 1 & \alpha^1 & \alpha^2 & \alpha^3 & \cdots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \cdots & \alpha^{2(q-2)} \\ \vdots & & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \alpha^{3(n-k)} & \cdots & \alpha^{(n-k)(q-2)} \end{pmatrix} \tag{3.5}$$

Vynásobení paritní matice $P$ přijatým slovem $\vec{r}$ odpovídá valuaci přijatého polynomu ve všech $n-k$ kořenech generujícího polynomu $\alpha, \ldots, \alpha^{n-k}$. Pokud je vše v pořádku, měl by být výsledkem nulový vektor délky $n-k$. Jinak víme, že nastala chyba ($r(x)$ není násobkem $g(x)$).

## 3.2 Kódování RS kódů

Z definice můžeme rovnou sestrojit generující matici, tedy zdálo by se, že proces kódování je daný. Již jsem se zmínil o efektivnosti systematického kódu. Pokud při přenosu nenastane chyba, jsme schopni informaci ihned vyčíst.

Důležitou vlastností pro nás bude opět cykličnost RS kódů, zvláště pak existence generujícího polynomu $g(x)$. Uvedený postup je obecný pro cyklické kódy a ukazuje kódování do systematického tvaru.

Máme dán cyklický kód $C$ s parametry $[n, k, d]$ a generující polynom $g(x)$ stupně $n-k$. Informaci délky $k$, kterou chceme kódovat, budeme reprezentovat polynomem $m(x)$ stupně nejvýše $k-1$.

Nejprve spočteme *paritní* polynom $p(x)$:

$$x^{n-k} \cdot m(x) \equiv p(x) \ (mod\ g(x)).$$

Stupeň $p(x)$ je menší než stupeň $g(x)$, tedy $deg(p) < n-k = deg(g)$. Polynom odpovídající kódovému slovu v již systematické formě je:

$$c(x) = x^{n-k} \cdot m(x) - p(x).$$

Přepsáno do vektoru vypadá kódové slovo takto:

$$\vec{c} = (-p_0, \ldots, -p_{n-k-1}, m_0, \ldots, m_k).$$

## 3.3 Dekódování RS kódů

Nejpoužívanější metody pro dekódování RS kódů jsou založeny na jeho cykličnosti[3].

Výsledek součinu paritní matice $P$ s přijatým slovem $\vec{r}$ nazveme syndromem a označíme ho $\vec{S}$:

$$\vec{S} = P \cdot \vec{r}^T. \tag{3.6}$$

[3]S mírnými úpravami jsou aplikovatelné například i na BCH kódy.

Tvar přijatého slova je $\vec{r} = \vec{c} + \vec{e}$, kde $\vec{e}$ je slovo určující chybu. Součin paritní matice a kódového slova je z definice roven nule a tak rovnost 3.6 můžeme pro RS kódy přepsat jako:

$$\vec{S} = \begin{pmatrix} 1 & \alpha^1 & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \cdots & \alpha^{2(n-1)} \\ \vdots & & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{(n-k)2} & \alpha^{(n-k)3} & \cdots & \alpha^{(n-k)(n-1)} \end{pmatrix} \cdot \vec{e}^T \tag{3.7}$$

Pro cyklické kódy je výpočet syndromů ekvivalentní valuaci přijatého polynomu

$$S_j = r(\alpha^j) = e(\alpha^j) = \sum_{i=0}^{n-1} e_i (\alpha^j)^i \qquad \text{pro } j = 1, \ldots, n-k. \tag{3.8}$$

Můžeme předpokládat, že nastalo $v$ chyb. Chybový polynom $e(x)$ bude mít právě $v$ nenulových koeficientů. Jejich pozice označím $i_1, \ldots, i_v$. Jednotlivé syndromy $S_1, \ldots, S_{n-k}$ určují soustavu $n-k$ nelineárních rovnic tvaru:

$$S_j = \sum_{w=1}^{v} e_{i_w} (\alpha^j)^{i_w} \qquad \text{pro } j = 1, \ldots, n-k. \tag{3.9}$$

Abychom se neztratili v záplavě mocnin a indexů, zavedeme nové označení:

$$\begin{aligned} X_w &= \alpha^{i_w} \\ Y_w &= e_{i_w} \qquad \text{pro } w = 1, \ldots, v. \end{aligned} \tag{3.10}$$

Výpočet syndromu pak vypadá následovně:

$$S_j = \sum_{w=1}^{v} Y_w X_w^j \qquad \text{pro } j = 1, \ldots, n-k. \tag{3.11}$$

Hodnoty $Y_w$ se nazývají *velikosti chyb* (*error magnitudes*) a $X_w$ *chybové pozice* (*error locations*). Důvod užitých označení je zřejmý.

Zůstává tedy soustava nelineárních (mocninných) rovnic. Metody na její řešení využívají tzv. *error-locator polynom* $\Lambda(x)$. Je to polynom, jehož kořeny jsou inverzy k hledaným chybovým pozicím:

$$\Lambda(x) = \prod_{l=1}^{v} (1 - xX_l). \tag{3.12}$$

Dekódování můžeme rozdělit na několik kroků. Nejprve musíme nalézt error-locator polynom. Pokud se podaří určit koeficienty error-locator polynomu, můžeme nalézt jeho kořeny $X_w^{-1}$, tedy hledané chybové pozice. Pak už stačí jen spočítat velikosti chyb $Y_w$ a tím nakonec určit chybu $e(x)$.

Dekódování RS kódů (a i jiných cyklických kódů) pomocí error-locator polynomu můžeme zapsat následovně:

**Algoritmus** *Dekódování cyklických kódů pomocí error-locator polynomu*
VSTUP: přijaté slovo $r(x)$
VÝSTUP: dekódované slovo[4] $c'(x)$

1. Spočti syndromy $\vec{S}$. Pokud $\vec{S} = 0$ je $c'(x) = r(x)$ a ukonči dekódování.
2. Spočti error-locator polynom $\Lambda(x)$
3. Najdi kořeny $X_i$ polynomu $\Lambda(x)$ – *Chienovo prohledávání*
4. Spočti velikosti chyb $Y_i$ – *Forneyho vzorec*
5. Pomocí získaných $X_i$ a $Y_i$ urči chybové slovo $e'(x)$
6. $c'(x) = r(x) - e'(x)$

Jak se změní uvedené vzorce pro jiné cyklické kódy? Funkce pro $S_j$ z (3.11) se nazývá mocninný součet symetrické funkce $\Lambda(x)$. Pro dekódování pomocí error-locator polynomu potřebujeme $n-k$ po sobě jdoucích $S_j$ (pro $j = b+1, \ldots, b+n-k$, kde $b \geq 0$). Aby byl možný přepis (3.11), dostáváme nutnou i postačující podmínku na kořeny generujícího polynomu:

$$g(x) = \prod_{j=1}^{n-k}(x - \alpha^{b+Ej}) \qquad \text{pro } b \geq 0 \text{ a } E > 0$$

Malou újmou na obecnosti, ale v zájmu přehlednosti vzorců zůstaneme u našeho RS kódu. Pro jiné kódy nejde o nic jiného, než úpravu indexů ve vzorcích.

Dále se seznámíme se třemi algoritmy pro nalezení error-locator polynomu, metodou pro nalezení jeho kořenů a vzorcem pro výpočet chybových hodnot.

### 3.3.1 Petersonův algoritmus

Petersonův algoritmus slouží k nalezení koeficientů error-locator polynomu. V dalším textu budeme předpokládat, že při přenosu nastalo $v \in [0, \ldots, \lfloor \frac{d-1}{2} \rfloor]$ chyb.

Počet chyb určuje stupeň error-locator polynomu $\Lambda(x)$. Vztah mezi polynomem $\Lambda(x)$ a syndromy $S_j$ ukazuje následující tvrzení:

**Tvrzení 3.3.1** (zobecněné Newtonovy identity) *Pro syndromy* $S_j = \sum_{i=1}^{v} Y_i X_i^j$ *a error-locator polynom* $\Lambda(x) = \sum_{i=0}^{v} \Lambda_i x^i$ *platí následující rekurentní vztah:*

$$S_j = -\sum_{i=1}^{v} \Lambda_i S_{j-i} \qquad \text{pro } j = v+1, \ldots, d-1 \tag{3.13}$$

[4]Dekódované slovo $c'(x)$ se při velkém počtu chyb může lišit od odeslaného slova $c(x)$ – nestává *chyba dekódování*

**Důkaz**: Pro error-locator polynom

$$\prod_{j=1}^{v}(1 - xX_j) = \sum_{j=0}^{v} \Lambda_j x^j$$

dosazením $x = 1/X_i$ a vynásobením $Y_i X_i^{l+v}$ dostaneme

$$0 = \sum_{j=0}^{v} \Lambda_j Y_i X_i^{l+v-j}$$

Součtem přes všechna $i$:

$$0 = \sum_{i=1}^{v}\sum_{j=0}^{v} \Lambda_j Y_i X_i^{l+v-j} = \sum_{j=0}^{v} \Lambda_j \left( \sum_{i=1}^{v} Y_i X_i^{l+v-j} \right) = \sum_{j=0}^{v} \Lambda_j S_{l+v-j}$$

Víme, že $\Lambda_0 = 1$, a tak dostáváme hledaný tvar:

$$S_{l+v} = -\sum_{j=1}^{v} \Lambda_j S_{l+v-j} \qquad \text{pro } l = 1, \ldots, d-1-v$$

. □

Díky tvrzení víme, že pro chybu váhy $v$ závisí $j$-tý syndrom na $v$ předchozích. Uvedená rekurence vytváří soustavu lineárních rovnic, kde neznámé jsou právě hledané koeficienty $\Lambda_i$, pro $i = 1, \ldots, v$. Zapsáno maticově:

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ \vdots & & \ddots & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{pmatrix} \begin{pmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{pmatrix} = \begin{pmatrix} -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v} \end{pmatrix} \tag{3.14}$$

Řešení takové soustavy již není problém, avšak musíme si uvědomit, že neznáme počet chyb $v$ a tedy ani tvar soustavy rovnic. Pomůže následující tvrzení.

**Tvrzení 3.3.2** *Nechť při přenosu nastane právě $v$ chyb. Potom matice syndromů chybového vektoru $\vec{e}$*

$$S = \begin{pmatrix} S_1 & S_2 & \cdots & S_w \\ S_2 & S_3 & \cdots & S_{w+1} \\ \vdots & & \ddots & \vdots \\ S_w & S_{w+1} & \cdots & S_{2w-1} \end{pmatrix}$$

*je regulární pokud $w = v$ a singulární pokud $w > v$.*

**Důkaz**: Pokud $w > v$, tak skutečný error-locator polynom dle tvrzení 3.3.1 určuje koeficienty lineární kombinace prvních $v$ sloupců matice, jejíž výsledkem bude sloupec $v+1$. Přičtení lineární kombinace sloupců k jinému sloupci determinant neovlivní a je jasné, že matice s jedním nulovým sloupcem bude singulární.

Těžší je případ, kdy $w = v$. Předpokládejme, že je matice singulární, a tedy existuje pro soustavu (3.14) řešení $\Lambda^*(x) \neq \Lambda(x)$. Pokusíme se zopakovat postup z důkazu tvrzení 3.3.1. Nechť

$$\Lambda^*(\frac{1}{X_i}) = \beta_i \qquad \text{pro } i = 1, \ldots, v \tag{3.15}$$

Dosadíme-li do $\Lambda^*(x)$ za $x$ hodnotu $1/X_i$ a vynásobíme $Y_i X_i^{l+v}$, dostáváme:

$$\beta_i Y_i X_i^{l+v} = \sum_{j=0}^{v} \Lambda_j^* Y_i X_i^{l+v-j} \tag{3.16}$$

Sečteme-li rovnice přes všechna $i$:

$$\begin{aligned} \sum_{i=1}^{v} \beta_i Y_i X_i^{l+v} &= \sum_{i=1}^{v} \sum_{j=0}^{v} \Lambda_j^* Y_i X_i^{l+v-j} \\ \sum_{i=1}^{v} \beta_i Y_i X_i^{l+v} &= \sum_{j=0}^{v} \Lambda_j^* S_{l+v-j} \end{aligned} \tag{3.17}$$

Protože $\Lambda^*(x)$ je řešením soustavy, tak pro $l = 1, \ldots, v$ je pravá strana rovnice (3.17) rovna nule.

$$\sum_{i=1}^{v} (\beta_i Y_i) X_i^{l+v} = 0 \qquad \text{pro } l = 1, \ldots, v \tag{3.18}$$

Víme, že $Y_i \neq 0$ pro $i = 1, \ldots, v$ a pokud $\beta_r \neq 0$ pro nějaké $r$ dostáváme, že matice

$$\begin{pmatrix} X_1^{v+1} & X_1^{v+2} & \cdots & X_1^{2v} \\ X_2^{v+1} & X_2^{v+2} & & X_2^{2v} \\ \vdots & & \ddots & \vdots \\ X_v^{v+1} & X_v^{v+2} & \cdots & X_v^{2v} \end{pmatrix} \tag{3.19}$$

je singulární (lineární závislost sloupců určuje právě (3.18)). To je však *Vandermondova matice*, jejíž determinant je nenulový.

$$\begin{vmatrix} X_1^{v+1} & X_1^{v+2} & \cdots & X_1^{2v} \\ X_2^{v+1} & X_2^{v+2} & & X_2^{2v} \\ \vdots & & \ddots & \vdots \\ X_v^{v+1} & X_v^{v+2} & \cdots & X_v^{2v} \end{vmatrix} = \prod_{i=1}^{v} X_i^{v+1} \begin{vmatrix} 1 & X_1 & \cdots & X_1^{v-1} \\ 1 & X_2 & & X_2^{v-1} \\ \vdots & & \ddots & \vdots \\ 1 & X_v & \cdots & X_v^{v-1} \end{vmatrix} =$$

$$= \prod_{i=1}^{v} X_i^{v+1} \prod_{j<l} (X_l - X_j) \neq 0$$

Musí proto být $\beta_i = 0$ pro $i = 1, \ldots, v$. Dva polynomy stupně nejvýše $v$, se stejnými $v$ kořeny, jejichž absolutní členy se rovnají $\Lambda_0 = \Lambda_0^*$ musí být nutně totožné. Dokázali jsme, že $\Lambda(x) = \Lambda^*(x)$, a tedy i celé tvrzení. □

Nyní již máme dostatek materiálu pro zformování funkčního algoritmu na hledání error-locator polynomu.

**Algoritmus** *Petersonův algoritmus*
VSTUP: posloupnost syndromů $\{S_i\}_{i=1}^{d-1}$
VÝSTUP: error-locator polynom $\Lambda(x)$

1. Nastav $w = \frac{d-1}{2}$
2. Zjisti zda je matice
$$S = \begin{pmatrix} S_1 & S_2 & \cdots & S_w \\ S_2 & S_3 & \cdots & S_{w+1} \\ \vdots & & \ddots & \vdots \\ S_w & S_{w+1} & \cdots & S_{2w-1} \end{pmatrix}$$
regulární. Pokud ano, vypočítej soustavu (3.14) a jdi na krok 3.
Pokud je $w = 1$, ohlaš *selhání dekodéru* a skonči.
Pokud je matice singulární, sniž odhad $w = w - 1$ a proveď znovu krok 2.
3. Vrať $\Lambda(x)$

Jiný způsob využití rekurentního vztahu 3.3.1 pro hledání error-locator polynomu uvádím v následující části.

### 3.3.2 Berlekamp-Masseyho algoritmus

Berlekamp-Masseyho algoritmus opět pomůže určit koeficienty error-locator polynomu. Jednotlivé členy lineární rekurentní rovnice, jaká je v tvrzení 3.3.1, můžeme vytvořit *lineárním posuvným registrem se zpětnou vazbou* (*LFSR*[5]).

**LFSR** je teoretický obvod[6]. Skládá se z lineárně uspořádaných *buněk*, schopných uchovat jednu hodnotu. Buňka může být napojena *zpětnou vazbou*, která vytváří lineární kombinaci hodnot v buňkách.

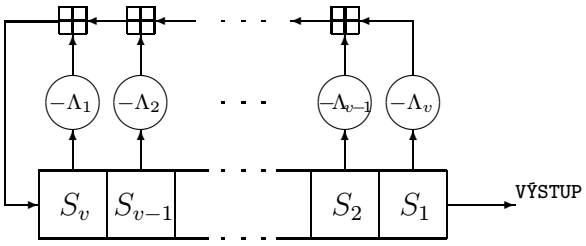
V každém kroku je lineární kombinací uložených hodnot vytvořena hodnota nová. Všechny uložené hodnoty se následně posunou o jednu buňku dopředu. První hodnota, která se nemá kam posunout, je výstup. Nově vypočtená hodnota je přidána na konec.

Aby mohl LFSR začít pracovat, je potřeba ho nejdříve naplnit *počátečními hodnotami*. V každém kroku pak na výstupu nalezneme jednu hodnotu. O takto vytvořené posloupnosti hodnot řekneme, že ji daný LFSR *generuje*.

Každý LFSR je úplně popsán svojí *délkou* (počtem buněk) a koeficienty lineární kombinace. Koeficienty se obvykle zapisují ve formě polynomu. Je snadné nahlédnout, že generovaná posloupnost je lineární rekurence.

[5]Linear Feedback Shift Register
[6]LFSR je hojně používán i v praxi, neboť je elektronicky velmi jednoduše realizovatelný.

Obrázek 3 ukazuje LFSR odpovídající rekurenci 3.3.1. Vazební polynom je $\Lambda(x)$ a délka registru $L = v$. LFSR je uveden v počátečním stavu, naplněný prvky posloupnosti $S_1, \cdots, S_v$.




**Obr. 3:** Posuvný registr (v počátečním nastavení) generující syndromy $S_1, \ldots, S_{2t+1}$

Celý registr je úplně popsán zpětnými vazbami $\Lambda(x)$, délkou a počátečním stavem. Úloha hledání koeficientů error-locator polynomu se transformuje na hledání nejkratšího posuvného registru generujícího předem známou posloupnost $S_1, \ldots, S_{d-1}$. Existence i jednoznačnost řešení takové úlohy máme již zajištěnou tvrzením (3.3.2).

Délka posuvného registru je rovna stupni error-locator polynomu $\Lambda(x)$. Při hledání nejkratšího registru generujícího celou posloupnost $d-1$ syndromů budeme postupovat iterativně. Pro $i$-tý krok budeme hledat nejkratší LFSR délky $L_i$, generující posloupnost $S_1, \ldots, S_i$. Každý takto vytvořený LFSR bude určovat svými zpětnými vazbami nějaký polynom $\Lambda^{(i)}(x)$, tedy v každém kroku hledáme dvojici $(\Lambda^{(i)}(x), L_i)$. Stupeň polynomu se přitom nemusí rovnat délce registru, protože vpravo umístěné buňky nemusejí mít zpětnou vazbu. Uvedená dvojice pak dostatečně popisuje nalezený LFSR.

**Tvrzení 3.3.3** *Pro posloupnost $\{S_i\}_{i=1}^{d-1}$, nejkratší lineární posuvný registr $(\Lambda^{(i+1)}(x), L_{i+1})$ generující prvních $i+1$ členů a nejkratší registr $(\Lambda^{(i)}(x), L_i)$ generující prvních $i$ členů, ne však $i$-tý, platí nerovnost:*

$$L_{i+1} \geq \max\{L_i, i + 1 - L_i\} \tag{3.20}$$

**Důkaz**: Zcela zřejmě platí

$$L_{i+1} \geq L_i, \tag{3.21}$$

jinak by totiž byl $(\Lambda^{(i+1)}(x), L_{i+1})$ kratším registrem generujícím $i$ členů posloupnosti. Pro důkaz sporem předpokládejme, že

$$L_{i+1} < i - L_i. \tag{3.22}$$

Vyjádříme-li pro $(\Lambda^{(i+1)}(x), L_{i+1})$ člen $S_{i+1}$

$$S_{i+1} = -\sum_{j=1}^{L_{i+1}} \Lambda_j^{(i+1)} S_{i+1-j} \tag{3.23}$$

prvky $S_{i+1-j}$ v součtu můžeme počítat pomocí registru ($\Lambda^{(i)}(x)$,$L_i$). Nahrazení $S_{i+1-j}$ smíme provést, pokud se při tom nebudeme odkazovat na neexistující členy $S_m$, pro $m \leq 0$. Nahrazovaným prvkem s nejnižším indexem je $S_{i+1-L_{i+1}}$, a proto

$$\begin{aligned} i+1-L_{i+1}-L_i &> 0 \\ L_{i+1} &< i+1-L_i. \end{aligned} \tag{3.24}$$

To je splněno díky předpokladu (3.22), a tak můžeme provést substituci

$$\begin{aligned} S_{i+1} &= -\sum_{j=1}^{L_{i+1}} \Lambda_j^{(i+1)} \left( -\sum_{k=1}^{L_i} \Lambda_k^{(i)} S_{i+1-j-k} \right) \\ S_{i+1} &= \sum_{k=1}^{L_i} \sum_{j=1}^{L_{i+1}} \Lambda_j^{(i+1)} \Lambda_k^{(i)} S_{i+1-j-k} \\ S_{i+1} &= \sum_{k=1}^{L_i} \Lambda_k^{(i)} \left( \sum_{j=1}^{L_{i+1}} \Lambda_j^{(i+1)} S_{i+1-j-k} \right) \\ S_{i+1} &= \sum_{k=1}^{L_i} \Lambda_k^{(i)} S_{i+1-k} \end{aligned} \tag{3.25}$$

Dostáváme, že registr ($\Lambda^{(i)}(x)$,$L_i$) generuje člen $S_{i+1}$. To je však ve sporu s předpokladem tvrzení. Podmínka (3.22) tedy neplatí a dostáváme $L_{i+1} \geq i+1-L_i$ □

Ve skutečnosti platí v uvedeném tvrzení rovnost, to však uvidíme dále. Pro pochopení algoritmu je potřeba další tvrzení o na první pohled zvláštním LFSR.

**Tvrzení 3.3.4** *Nejkratší LFSR generující posloupnost $i$ nul a jako $(i+1)$-ní prvek nějaké nenulové číslo má délku $L = i+1$.*

**Důkaz**: Pokud by platilo $L < i+1$, musel by být registr na začátku naplněn pouze nulami, a proto by dále produkoval znovu samé nuly. □

Zpátky k našemu algoritmu, předpokládejme, že již máme nalezeny nejkratší registry $(\Lambda^{(0)}(x) = 1, L_0 = 0)$, $(\Lambda^{(1)}(x), L_1)$, …, $(\Lambda^{(i)}(x), L_i)$. Chtěli bychom za pomoci těchto registrů nelézt registr $(\Lambda^{(i+1)}(x), L_{i+1})$. Již známe nejkratší LFSR $(\Lambda^{(i)}(x), L_i)$ produkující posloupnost délky $i$. Můžeme spočítat jaký bude $(i+1)$-ní prvek:

$$\hat{S}_{i+1} = -\sum_{j=1}^{L_i} \Lambda_j^{(i)} S_{i-j} \tag{3.26}$$

To, co nás ve skutečnosti zajímá, je odchylka od námi hledaného LFSR:

$$\Delta_{i+1} = S_{i+1} - \hat{S}_{i+1} = \sum_{j=0}^{L_i} \Lambda_j^{(i)} S_{i-j} \tag{3.27}$$

Pokud je $\Delta_{i+1} = 0$ jsme hotovi, a $(\Lambda^{(i)}(x), L_i)$ je zároveň nejkratším registrem generujícím $i+1$ prvků posloupnosti. Avšak pokud $\Delta_{i+1} > 0$ musíme modifikovat zpětné vazby, tedy polynom $\Lambda^{(i)}(x)$:

$$\Lambda^{(i+1)}(x) = \Lambda^{(i)}(x) + A(x) \tag{3.28}$$

pro nějaký polynom $A(x)$.

Abychom byli schopni určit polynom $A(x)$, předpokládejme, že někdy v kroku $m$, pro $0 \leq m < i$, jsme narazili na stejný problém a došlo k prodloužení registru

$$\begin{aligned} L_m &< L_i \\ L_{m+1} &= L_i \end{aligned} \tag{3.29}$$

Pokusíme se využít polynom $\Lambda^{(m)}(x)$ pro sestavení polynomu $A(x)$. Po polynomu $A(x)$ chceme, aby jako vazební polynom produkoval samé nuly a pouze při $(i+1)$-ním prvku vytvořil opačnou odchylku $-\Delta_{i+1}$. Takovou volbou je například posunutý a mírně upravný polynom $\Lambda^{(m)}(x)$

$$A(x) = -\Delta_{i+1}\Delta_m^{-1}x^{i-m}\Lambda^{(m)}(x). \tag{3.30}$$

Hledaný polynom je

$$\Lambda^{(i+1)}(x) = \Lambda^{(i)}(x) - \Delta_{i+1}\Delta_m^{-1}x^{i-m}\Lambda^{(m)}(x). \tag{3.31}$$

Stupeň takto vytvořeného polynomu je

$$\deg(\Lambda^{(i+1)}) \leq \max\{L_i, i - m + L_m\}. \tag{3.32}$$

Předpokládejme, že v tvrzení 3.3.3 platí rovnost.

$$L_{m+1} = L_i = \max\{L_m, m + 1 - L_m\} \tag{3.33}$$

Speciálně díky (3.29) dostáváme

$$L_i = m + 1 - L_m. \tag{3.34}$$

Proto v (3.32) dostáváme

$$\deg(\Lambda^{(i+1)}) \leq \max\{L_i, i + 1 - L_i\} \tag{3.35}$$

a tedy díky předpokládané rovnosti v 3.3.3 pak můžeme určit

$$L_{i+1} = \max\{L_i, i + 1 - L_i\}.$$

Znamená to, že pokud jsme schopni pro počáteční hodnoty algoritmu zachovat rovnost v tvrzení 3.3.3, po zbytek algoritmu bude platit také. Počáteční podmínky

$$(\Lambda^{(0)} = 1, L_0 = 0)(\Lambda^{(1)} = 1, L_1 = 1) \tag{3.36}$$

rovnost splňují a dokázali jsme tedy, že v tvrzení 3.3.3 platí rovnost i nalezli algoritmus.

**Algoritmus** Berlekamp-Massey
VSTUP: posloupnost syndromů $\{S_i\}_{i=1}^{d-1}$
VÝSTUP: error-locator polynom $\Lambda(x)$

1. Počáteční podmínky algoritmu jsou

$$(\Lambda^{(0)}(x) = 1, L_0 = 0)$$
$$(\Lambda^{(1)}(x) = 1, L_1 = 1)$$

2. $i$-tý krok: Spočete pro $(\Lambda^{(i)}(x), L_i)$ odchylku na $(i+1)$-ním prvku $\Delta_{i+1}$

$$\Delta_{i+1} = \sum_{j=0}^{L_i} \Lambda_j^{(i)} S_{i-j} \tag{3.37}$$

Pokud je $\Delta_{i+1} = 0$, tak $(\Lambda^{(i+1)}(x), L_{i+1}) = (\Lambda^{(i)}(x), L_i)$.
Pokud $\Delta_{i+1} \neq 0$, tak za pomoci nejbližšího registru $(\Lambda^{(r)}(x), L_r)$, pro $r < i$ kde $\Delta_{r+1} \neq 0$ spočteme

$$\Lambda^{(i+1)}(x) = \Lambda^{(i)} + \Delta_{i+1}\Delta_{r+1}^{-1} x^{i-r} \Lambda^{(r)}(x) \tag{3.38}$$
$$L_{i+1} = \max\{L_i, i + 1 - L_i\} \tag{3.39}$$

3. Pro krok $i = (d-2)$ vrať $\Lambda^{(d-1)}(x)$.

### 3.3.3 Euklidův algoritmus

V praxi se často stane, že je možné ještě před samotným dekódování označit některé přijaté symboly za chybné nebo ztracené. Taková situace nastane například tehdy, pokud je příchozí signál příliš slabý nebo dokonce žádný[7]. Symboly, o kterých víme, že jsou chybné, můžeme prohlásit za vymazané. Skutečná síla cyklických kódů tak přichází s možností přidat do dekódovacího procesu *výmazy* (*erasures*).

Zatímco u chyb neznáme jejich pozici ani počet, u výmazů víme oboje. Nejvíce dokážeme opravit $v$ chyb, kde $2v \leq d-1$. Naproti tomu výmazů dokážeme opravit $s$, kde $s \leq d-1$, neboť $n-(d-1)$ symbolů ještě jednoznačně určuje kódové slovo (vzdálenost dvou slov je alespoň 1). Spojeno dohromady dostáváme odhad, kolik chyb dokážeme při $s$ výmazech opravit:

$$2v + s \leq d - 1. \tag{3.40}$$

Přijaté slovo má po zavedení výmazů tvar $r(x) = c(x) + \hat{e}(x) + e^*(x) = c(x) + e(x)$. Polynom $\hat{e}(x)$ je chybový (error) polynom a $e^*(x)$ je polynom určující smazané (erasure) symboly. Celkovou chybu pak určuje *errata-polynom* $e(x)$. Syndromy pro $v$ chyb a $s$ výmazů jsou tvaru

$$S_i = r(\alpha^i) = e(\alpha^i) = \sum_{j=0}^{n-1} e_j \alpha^{ij} = \sum_{j=1}^{v+s} Y_j X_j^i, \tag{3.41}$$

[7]U CD přehrávačů, kde jednotlivé symboly znamenají různý fázový posun odraženého laseru, nemusí vůbec k odrazu dojít

kde $Y_j$ a $X_j$ jsou jak velikosti a pozice chyb, tak i výmazů. Pomocí syndromů dále definujeme *syndrom polynom*

$$S(x) = \sum_{i=1}^{2t} S_i x^{i-1}. \tag{3.42}$$

Abychom mohli začít pracovat s nově definovaným errata polynomem je nutné rozšířit rodinu locator polynomů:
*error-locator*

$$\lambda(x) = \prod_{j=1}^{s}(1 - X_j x) \tag{3.43}$$

*erasure-locator*

$$\tau(x) = \prod_{j=1}^{s}(1 - X_j x) \tag{3.44}$$

*errata-locator*

$$\Lambda(x) = \tau(x)\lambda(x) = \prod_{j=1}^{v+s}(1 - X_j x) \tag{3.45}$$

Nakonec další důležitý polynom je *errata-evaluator*

$$A(x) = \sum_{j=1}^{v+s} Y_j X_j \left( \prod_{i \neq j}(1 - X_i x) \right). \tag{3.46}$$

S errata-locator polynomem, jehož stupeň se může blížit hodnotě $2t$, nemůžeme využít předchozích algoritmů využívajících rekurenci z tvrzení 3.3.1. Musíme se tedy poohlédnout po jiných vztazích.

**Tvrzení 3.3.5** *Pro syndrom polynom a locator polynomy platí následující vztah:*

$$S(x) \equiv \frac{A(x)}{\lambda(x)\tau(x)} \mod x^{2t} \tag{3.47}$$

**Důkaz**: Dosazením 3.41 do 3.42 a následnou úpravou dostáváme

$$S(x) = \sum_{i=1}^{2t} \sum_{j=1}^{v+s} Y_j X_j^i x^{i-1} = \sum_{j=1}^{v+s} Y_j \sum_{i=1}^{2t} X_j^i x^{i-1} =$$

$$= \sum_{j=1}^{v+s} Y_j X_j \sum_{i=0}^{2t-1} (X_j x)^i = \sum_{j=1}^{v+s} Y_j X_j \left( \frac{1 - (X_j x)^{2t}}{(1 - X_j x)} \right) =$$

$$= \sum_{j=1}^{v+s} \frac{Y_j X_j}{(1 - X_j x)} - \sum_{j=1}^{v+s} \frac{Y_j X_j^{2t+1} x^{2t}}{(1 - X_j x)}$$

Vynásobením errata-locator polynomem $\Lambda(x)$ obdržíme rovnici

$$S(x)\Lambda(x) = A(x) + x^{2t}\left(\sum_{j=1}^{v+s} Y_j X_j^{2t+1}\left(\prod_{i\neq j}(1 - X_i x)\right)\right)$$

Protože absolutní člen $\Lambda(x) = \tau(x)\lambda(x)$ je nenulový a tedy

$$\Lambda(x) \bmod x^{2t} \neq 0$$

, již snadno dostáváme dokazovanou rovnost. □

Co v kongruenci z uvedeného tvrzení známe a co chceme zjistit? Jistě umíme spočítat syndrom polynom. Protože pracujeme s výmazy (známe jejich pozici), máme k dispozici i erasure-locator polynom $\tau(x)$. Je proto výhodné kongruenci ještě mírně uspořádat, a tak definujeme Forney-syndrom polynom $T(x)$

$$T(x) \equiv S(x)\tau(x) \bmod x^{2t} \tag{3.48}$$

s jehož pomocí pak kongruence nabývá tvaru, který si zaslouží (alespoň symbolicky) své tvrzení (již bez důkazu):

**Tvrzení 3.3.6** (Berlekamp-Forney key equation)

$$A(x) \equiv T(x)\lambda(x) \bmod x^{2t} \tag{3.49}$$

Rovnice na první pohled vypadá, že pro neznámé polynomy $A(x)$ a $\lambda(x)$ bude mít nekonečně mnoho řešení. Ve skutečnosti jsou naše vyhlídky na jejich nalezení mnohem lepší. Pomohou v tom odhady stupňů obou polynomů.

Pro $v$ chyb a $s$ výmazů platí $2v + s \leq d - 1$. Protože $s$ známe, můžeme přepsat nerovnost na $v \leq \frac{d-1-s}{2}$. Odhady stupňů tedy jsou:

$$\deg(\lambda) \leq \frac{d-1-s}{2} \tag{3.50}$$

$$\deg(A) \leq \frac{d-3+s}{2} \tag{3.51}$$

$$\deg(\lambda) + \deg(A) \leq \frac{d-1-s}{2} + \frac{d-3+s}{2} = d - 2 < d - 1 \tag{3.52}$$

Dalším důležitým faktem, vyplývajícím z definice obou polynomů, je jejich nesoudělnost. Za těchto podmínek má Berlekamp-Forneyova rovnice dokonce právě jediné řešení pro neznámé $\lambda(x)$ a $A(x)$. Aby bylo jasné, proč tomu tak je, musíme se nejprve podívat na některé vlastnosti *rozšířeného Euklidova algoritmu* (EEA) pro polynomy[8]. Na chvíli tedy opustíme zavedenou symboliku, abychom se na problém podívali v obecné rovině.

[8]Postup počítání EEA pro polynomy je totožný s EEA pro celá čísla

**Rozšířený Euklidův algoritmus** je algoritmus na nalezení největšího společného dělitele dvou polynomů $\gcd(a(x), b(x)) = c(x)$ a zároveň koeficientů $f(x)$ a $g(x)$ splňujících rovnost

$$a(x)f(x) + b(x)g(x) = c(x). \tag{3.53}$$

Zajímavější než samotný výsledek algoritmu pro nás bude postup, kdy v $i$-tém kroku máme nalezeny polynomy $f_i(x)$, $g_i(x)$ a $c_i(x)$ splňující

$$a(x)f_i(x) + b(x)g_i(x) = c_i(x). \tag{3.54}$$

Počítání největšího společného dělitele $c(x)$ (klasický Euklidův algoritmus) vyjadřuje rekurentní formulka

$$c_i(x) = c_{i-2}(x) - q_i(x)c_{i-1}(x), \tag{3.55}$$

kde $\deg(c_i) < \deg(c_{i-1})$. Dosazením do (3.54) okamžitě vidíme rovnice pro zbylé dva polynomy

$$f_i(x) = f_{i-2}(x) - q_i(x)f_{i-1}(x) \tag{3.56}$$

$$g_i(x) = g_{i-2}(x) - q_i(x)g_{i-1}(x). \tag{3.57}$$

Počáteční podmínky vypadají následovně:

$$\begin{array}{lll} c_{-1}(x) = a(x) & f_{-1}(x) = 1 & g_{-1}(x) = 0 \\ c_0(x) = b(x) & f_0(x) = 0 & g_0(x) = 1 \end{array}$$

Krok algoritmu můžeme zapsat také maticově. V následujících rovnicích budu kvůli přehlednosti psát polynomy bez $(x)$ - tedy např. místo $q_i(x)$ jen $q_i$:

$$\begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & q_i \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \end{pmatrix} \tag{3.58}$$

Maticový přístup umožní jednoduše odvodit některé netriviální vztahy. Z (3.58) dostáváme

$$\begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix} = \prod_{j=1}^{i} \begin{pmatrix} 0 & 1 \\ 1 & q_j \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f_{i-1} & g_{i-1} \\ f_i & g_i \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \tag{3.59}$$

Důležitý vztah dává determinant matice $\begin{pmatrix} f_{i-1} & g_{i-1} \\ f_i & g_i \end{pmatrix}$, který jak je vidět z nerozznásobeného tvaru se rovná $(-1)^i$. Dostáváme tedy

$$f_{i-1}g_i - f_i g_{i-1} = (-1)^i. \tag{3.60}$$

Vyjádříme-li z (3.54) $f_i = \frac{c_i - bg_i}{a}$, dosadíme do (3.60) a obdobně i pro $g_i$, dostaneme další dva vztahy

$$c_{i-1}g_i - c_i g_{i-1} = (-1)^i a \tag{3.61}$$

$$f_{i-1}c_i - f_i c_{i-1} = (-1)^i b. \tag{3.62}$$

Nyní přijdou na řadu odhady stupňů, neboť právě ty sehrají nejdůležitější roli. První je z (3.55), kde $c_i$ je zbytek po dělení $c_{i-1}$, a tedy

$$\deg(c_i) < \deg(c_{i-1}). \tag{3.63}$$

Z rovnic (3.56) a (3.57) můžeme nahlédnout, že stupně $f_i$ i $g_i$ jsou neklesající, a tak dostáváme

$$\deg(f_i) > \deg(f_{i-1}) \tag{3.64}$$

$$\deg(g_i) > \deg(g_{i-1}). \tag{3.65}$$

Tyto odhady spojené s (3.61) a (3.62) dávají nakonec

$$\deg(g_i) + \deg(c_{i-1}) = \deg(a) \tag{3.66}$$

$$\deg(f_i) + \deg(c_{i-1}) = \deg(b). \tag{3.67}$$

Tímto jsme dokončili rozbor rozšířeného Euklidův algoritmu a na řadu přichází důležité tvrzení.

**Tvrzení 3.3.7** *Pro polynomy $a(x)$, $b(x)$ a neznámé polynomy $f(x)$ a $c(x)$ splňující odhady $\deg(f) \leq \mu$ a $\deg(c) \leq \nu$, kde $\mu + \nu = \deg(b) - 1$ má rovnice*

$$a(x)f(x) \equiv c(x) \bmod b(x) \tag{3.68}$$

*za podmínky $\gcd(f(x), c(x)) = 1$ právě jedniné řešení (až na přenásobení konstantou). Takové řešení můžeme navíc nalézt rozšířeným euklidovým algoritmem.*

**Důkaz**: Opět vynechám pro přehlednost v označení polynomů $(x)$. Rovnost (3.68) můžeme přepsat jako

$$af + bg = c, \tag{3.69}$$

pro nějaký blíže nespecifikovaný polynom $g$.

Rozšířený Euklidův algoritmus pro dvojici $(a, b)$ postupně vytváří trojice $(f_i, g_i, c_i)$ splňující

$$af_i + bg_i = c_i. \tag{3.70}$$

Stupeň $c_i$ je ostře klesající, a tak můžeme nalézt první index $i'$, pro který platí

$$\begin{aligned} \deg(c_{i'-1}) &> \nu \\ \deg(c_{i'}) &\leq \nu. \end{aligned} \tag{3.71}$$

Rovnost s hledanými polynomy a rovnost s polynomy z $i'$-tého kroku algoritmu po řadě vynásobené $f_{i'}$ a $f$ jsou pak

$$\begin{aligned} af f_{i'} + bg f_{i'} &= c f_{i'} \\ af_{i'} f + bg_{i'} f &= c_{i'} f. \end{aligned} \tag{3.72}$$

Odečteme-li druhou rovnici od první, dostáváme

$$bg f_{i'} - bg_{i'} f = c f_{i'} - c_{i'} f$$

$$c f_{i'} \equiv c_{i'} f \bmod b.$$

Z podmínek v tvrzení je $\deg(f) \leq \mu$, což spolu s (3.71) dává $\deg(c_{i'} f) \leq \mu + \nu < \deg(b)$. Z (3.71) také víme, že $\deg(c) \leq \nu < \deg(c_{i'-1})$ a spojeno s rovností (3.67)

$\deg(f_{i'}) = \deg(b) - \deg(c_{i'-1}) = \mu + \nu + 1 - \deg(c_{i'-1}) \leq \mu$. Obdobně dostáváme tedy $\deg(cf_{i'}) \leq \mu + \nu < \deg(b)$. Proto dostáváme rovnost

$$cf_{i'} = c_{i'} f \tag{3.73}$$

Spojíme-li tento fakt s (3.72), vyjde též

$$gf_{i'} = g_{i'} f. \tag{3.74}$$

Podmínka (3.60) zaručuje nesoudělnost $f_{i'}$ a $g_{i'}$ a poslední rovnost tedy znamená, že pro nějaký polynom $q$ platí

$$\begin{aligned} f &= qf_{i'} \\ g &= qg_{i'}. \end{aligned} \tag{3.75}$$

Dosadíme-li do (3.69), získáme:

$$aqg + bqf = c.$$

Dostáváme tedy $q|c$. Jedna z podmínek tvrzení však byla nesoudělnost $f$ a $c$. Proto musí být $\deg(q) = 0$ a $q$ je pouze číslo. Z (3.75) potom dostáváme pro nějaké číslo $\Delta$

$$\begin{aligned} f(x) &= \Delta f_{i'}(x) \\ g(x) &= \Delta g_{i'}(x) \end{aligned}$$

a dále samozřejmě také

$$c(x) = \Delta c_{i'}(x).$$

Tím jsme dokončili konstrukční důkaz ukazující, že řešení nalezneme v $i'$-tém kroku, kde $c_{i'}$ je první polynom z $\{c_i\}$, jehož stupeň klesne pod odhad $\nu$. □

Použití tvrzení 3.3.7 na řešení Berlekamp-Forneyovy klíčové rovnice je celkem zřejmé. Již známe odhady stupňů (3.50) až (3.52), které podmínky tvrzení přesně splňují, a není tedy nic dalšího, co bychom potřebovali.

**Algoritmus** *Euklidův (dekódovací) algoritmus*
VSTUP: syndromy $S_i$, erasure-locator $\tau(x)$
VÝSTUP: errata-locator $\Lambda(x)$, errata-evaluator $A(x)$

1. Vypočítejte syndrom polynom $S(x)$ dle (3.42).
2. Vypočítejte Forney-syndrom polynom $T(x)$ dle (3.48).
3. Pro výpočet $\Lambda(x)$ a $A(x)$ aplikujte rozšířený Euklidův algoritmus na polynomy $(T(x), x^{d-1})$ s počátečními podmínkami: $\Lambda_0(x) = \tau(x)$, $\Lambda_{-1} = 0$, $R_{-1}(x) = x^{d-1}$, $R_0(x) = T(x)$.

   $$R_i(x) = R_{i-2}(x) - Q_i(x) R_{i-1}(x)$$

   $$\Lambda_i(x) = \Lambda_{i-2}(x) - Q_i(x) \Lambda_{i-1}(x)$$

   kde $\deg(R_i) < \deg(R_{i-1})$. Pro první index $i'$ splňující $\deg(R_{i'}) < \frac{d-3+s}{2}$ spočítejte $\Delta = \Lambda_{i'}(0)$ a pokračujte krokem 4.

4. $A(x) = \frac{A_{i'}(x)}{\Delta}$ a $\Lambda(x) = \frac{\Lambda_{i'}(x)}{\Delta}$

Uvedený algoritmus zaslouží ještě krátký komentář. Předně bije do očí erratalocator polynom $\Lambda(x)$, který se v Berlekamp-Forneyově klíčové rovnici nevyskytuje. Může za něj upravená počáteční podmínka $\Lambda_0(x) = \tau(x)$. Napíšeme-li vztah pro $\Lambda_1(x)$: $\Lambda_1(x) = \Lambda_{-1}(x) - Q_1(x)\Lambda_0(x) = Q_1(x)\tau(x)$, tak pro posloupnost $\Lambda'_i(x)$ s neupravenou počáteční podmínkou $\Lambda'_0(x) = 1$ jistě platí: $\Lambda_{i'}(x) = \tau(x)(\Lambda'_{i'-2}(x) - Q_{i'}(x)\Lambda'_{i'-1}(x)) = \tau(x)\Lambda'_{i'}(x)$. Tedy upravením počáteční podmínky jsme nijak chod algoritmu neovlivnili (zastavující podmínka se vztahuje k $A_i(x)$). Navíc rovnou obdržíme error-locator polynom a ušetřili jedno závěrečné násobení.

Určení parametru $\Delta$ také není nijak záhadné, neboť víme, že absolutní člen erratalocator polynomu $\Lambda(x)$ musí být jedna, tedy $\Delta = \Lambda_{i'}(0)$.

Je jasné, že uvedený algoritmus můžeme použít i pro samostatné dekódování chyb pouhým nastavením počtu výmazů $s = 0$. Zároveň je potřeba poukázat na tvrzení (3.3.7), které je obecně velmi užitečným a mocným nástrojem. Proto se můžeme setkat i s klíčovými rovnicemi jiného tvaru.

Nakonec musím podotknout, že i Berlekamp-Masseyho algoritmus lze upravit pro dekódování chyb a výmazů.

### 3.3.4 Chienovo prohledávání

Chienovo prohledávání je metoda pro nalezení kořenů polynomů nad konečnými tělesy. Používá brutální síly, tedy zkouší všechny možné prvky tělesa. Každé konečné těleso $\mathbb{F}_q$ má svůj primitivní prvek $\alpha$, jehož mocniny generují celé těleso, kromě nuly. Dosazení mocniny $\alpha^i$ do polynomu $\Lambda(x)$ vypadá následovně

$$\Lambda(\alpha^i) = 1 + \Lambda_1\alpha^i + \Lambda_2\alpha^{2i} + \ldots + \Lambda_v\alpha^{vi}$$

Při dosazení následující mocniny $\alpha^{i+1}$ můžeme využít předchozího výpočtu:

$$\Lambda(\alpha^{i+1}) = 1 + (\Lambda_1\alpha^i)\alpha^i + (\Lambda_2\alpha^{2i})\alpha^2 + \ldots + (\Lambda_v\alpha^{vi})\alpha^v$$

**Algoritmus** *Chienovo prohledávání*[9]
VSTUP: error-locator polynom $\Lambda(x) \in \mathbb{F}_q[x]$
VÝSTUP: chybové pozice $\{i_w\}$, kde $X_w = \alpha^{i_w}$

1. Počáteční podmínky $\Lambda_j^{(0)} = \Lambda_j$, pro $j = 1, \ldots, v$
2. Pro $l = 1, \ldots, q-1$ proveď:

   $$\Lambda_j^{(l)} = \Lambda_j^{(l-1)}\alpha^j$$

   Pokud $\sum_{j=1}^{v} \Lambda_j^{(l)} = -1$ pak $q - l$ je chybová pozice.

Postup hledání kořenů je přímočarý, pro $\Lambda_j^{(l)} = \Lambda_j(\alpha^l)^j$. Chybová pozice je $j_w = q - l$ díky vlastnosti tělesa $\mathbb{F}_q$: $\alpha^{-l} = \alpha^{q-l}$.

[9] Chienovo prohledávání je obecná metoda pro hledání kořenů polynomů nad konečným tělesem… tady je mírně upravena kvůli požadovanému výstupu

### 3.3.5 Forneyho vzorec

V momentě, kdy určíme chybové pozice $X_i$, nejsme hotovi. Stále nás čeká spočítat velikosti chyb $Y_i$. Jednu z možností dává samotná definice syndromů:

$$S_i = Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \qquad \text{pro } i = 1, \ldots, d-1 \tag{3.76}$$

Protože $v \leq d-1$ ($v$ je počet chyb v případě opravování chyb nebo počet chyb a výmazů, pokud počítáme i s výmazy), máme dostatečnou soustavu lineárních rovnic pro $v$ neznámých $Y_i$.

Jednodušší výpočet ukazuje Forneyho vzorec. Budeme potřebovat *error-evaluator polynom*[10] $A(x)$.

$$A(x) = \sum_{j=1}^{v} Y_jX_j \left( \prod_{i \neq j} (1 - X_ix) \right) \tag{3.77}$$

Vztah polynomu $A(x)$ a již známých polynomů udává Berlekamp-Forneyova klíčová rovnice (3.3.6)

$$A(x) \equiv S(x)\Lambda(x) \bmod x^{2t} \tag{3.78}$$

kde $S(x)$ je polynom syndromů stejný jako (3.42), tedy

$$S(x) = \sum_{i=1}^{2t} S_ix^{i-1} \tag{3.79}$$

Dále potřebujeme pro error-locator polynom $\Lambda(x)$ jeho derivaci vzhledem k $x$:

$$\Lambda'(x) = \Lambda_1 + 2\Lambda_2x + 3\Lambda_3x^2 + \ldots + v\Lambda_vx^{v-1} \tag{3.80}$$

**Tvrzení 3.3.8** (Forneyho vzorec)

$$Y_j = -\frac{A(X_j^{-1})}{\Lambda'(X_j^{-1})} \qquad \text{pro } j = 1, \ldots, v \tag{3.81}$$

**Důkaz**: Dosazením $X_j^{-1}$ do $A(x)$ dostáváme:

$$A(X_j^{-1}) = Y_jX_j \left( \prod_{i \neq j} (1 - X_iX_j^{-1}) \right) \tag{3.82}$$

Vyjádříme derivaci $\Lambda(x)$ dle vzorce pro derivaci součinu ($(fg)' = f'g + fg'$):

$$\Lambda'(x) = (-X_j) \left( \prod_{i \neq j} (1 - X_ix) \right) + (1 - X_j) \left( \prod_{i \neq j} (1 - X_ix) \right)'$$

a po dosazení $X_j^{-1}$ dostáváme:

$$\Lambda'(X_j^{-1}) = (-X_j) \left( \prod_{i \neq j} (1 - X_iX_j^{-1}) \right).$$

□

Získali jsme tedy vzorec pro snadnější výpočet velikostí chyb $Y_j$. Pokud známe dvojice $(X_j, Y_j)$, nic nebrání ve finálním určení chybového slova $e(x)$.

[10] V Euklidově algoritmu je to již spočtený errata-evaluator polynom

### 3.3.6 Krátce o složitostech

Ne vždy je nejefektivnější algoritmus ten nejpoužívanější. V praxi záleží na potřebě rychle dekódovat proti které však stojí složitost, velikost a i spotřeba energie dekodéru. Záleží především na konkrétní aplikaci. O některých pak bude řeč v následující kapitole.

V této části stručně shrnuji složitosti uvedených algoritmů bez hlubšího dokazování. Složitosti jsou uváděny ve vztahu k délce kódového slova $n$, délce informace $k$ a minimální vzdálenosti $d$.

**Kódování**

Zdaleka nejsložitější operací je dělení polynomu se zbytkem. Jeho složitost je $O(dn)$.

**Výpočet syndromů**

Výpočet syndromů znamená pouze $(d-1)$ krát dosadit do přijatého polynomu $r(x)$. Použijeme-li *Hornerovo schéma*, složitost bude $(d-1)O(n) \approx O(dn)$.

**Petersonův algoritmus**

Pro určení koeficientů error-locator polynomu je potřeba řešit soustavu rovnic. Pro nejhorší případ, tedy nastane-li jedna chyba, musíme zjistit regularitu matice $S(w \times w)$, pro $w = (d-1)/2, \ldots, 1$. Budeme-li počítat za pomoci *Gaussovy eliminace*, jejíž složitost je $O(w^3)$, dostáváme složitost zhruba $O(d^4)$. Bohužel to, že nastane pouze jedna chyba, je obvykle nejpravděpodobnější.

**Berlekamp-Masseyho algoritmus**

Berlekamp-Masseyho algoritmus má oproti Petersonovu tu výhodu, že nejdříve se pokouší nalézt řešení pro jednu chybu, pak pro dvě, atd. Jeho použití je tedy výhodnější. Pokud nastane méně chyb, bude dekódování rychlé (ne jako u Petersonova algoritmu). Algoritmus má celkem $d-1$ kroků. V každém musíme počítat odchylku (pro $i$-tý krok nejhůře $O(i)$). Přepočítání locator polynomu pak provedeme nejhůře v $(d-1)/2$ krocích (opět zhruba $O(i)$). Celkově tedy dostáváme $O(d^2)$.

**Euklidův algoritmus**

Abychom mohli porovnat Euklidův algoritmus s předchozími, předpokládejme, že dekódujeme pouze chyby bez výmazů ($s = 0$). Tím jsme se mimo jiné vyhli počítání Forney-syndrom polynomu. Složitost Euklidova algoritmu je v nejhorším případě $O(d^3)$. Avšak oproti předchozím je součástí výstupu i error-evaluator polynom $A(x)$.

**Chienovo prohledávání**

Při hledání kořenů error-locator polynomu zkoušíme dosadit všechny možné prvky tělesa $\mathbb{F}_q$. Dosazujeme do polynomu stupně nejvýše $(d-1)/2$. Protože délka kódového slova je shodná s počtem nenulových prvků tělesa, dostáváme složitost $O(dn)$.

**Forneyho vzorec**

Zde potřebujeme derivaci error-locator polynomu, což není z výpočetního hlediska pro polynomy žádný problém. Výpočet error-evaluatoru $A(x)$ zabere $O(d^2)$. Dále pro nejvýše $(d-1)/2$ chyb dosazujeme do polynomů stupně nejvýše $(d-1)/2$. Proto je složitost $O(d^2)$.

Všechny složitosti byly odvozeny velmi hrubě a vše kromě určujících členů bylo zanedbáno. To dává obrázek pouze o tom jak se chovají algoritmy asymptoticky. V

praxi pracujeme s kódy délky v řádu maximálně stovek symbolů. Při tak „malých“ délkách jsou to ale právě zanedbané členy a konstanty, které určují rychlost. Dalším zanedbáním jsou operace v konečném tělese (zpracované jako konstantní), kde není dělení úplně triviální záležitostí. V neposlední řadě velmi záleží na implementaci. Tohle vše je také důvod, proč nemá cenu zkoumat složitost algoritmů do hloubky, neboť bychom museli až na „dno“.

Uvedené metody dekódování rozhodně netvoří kompletní seznam. Mezi vynechanými je například hledání error-locator polynomu za pomoci *Groebnerových bází* nebo přímé využití definice RS kódů s dekódováním přes aproximační polynomy (*Goa algoritmus*).

# Kapitola 4

# Aplikace RS kódů

Samoopravné kódy jsou dnes používany skoro všude, kde se pracuje s daty. Každá jejich aplikace se liší dle daných požadavků. V této kapitole prezentuji nejznámější aplikace RS kódů.

## 4.1 CCSDS standard

V roce 1984 vydalo sdružení vesmírných agentur *CCSDS* (Consultative Committee for Space Data Systems) oficiální doporučení na komunikační kódovací standard pro kosmická plavidla.

Doporučení obsahuje RS kód $(255, 233)$. Kód je nad konečným tělesem $\mathbb{F}_{2^8}$ generovaným polynomem $p(x) = x^8 + x^7 + x^2 + x + 1$. Generující polynom $(255, 233)$ RS kódu je tvaru

$$g(x) = \prod_{j=112}^{143} (x - \alpha^{11j}), \tag{4.1}$$

kde $\alpha$ je primitivní prvek $\mathbb{F}_{2^8}$. Tento zvláštní tvar generujícího polynomu (je symetrický) dovoluje zjednodušit proces kódování.

CCSDS standard byl použit například v misích Mars Observer (1992), Pathfinder (1996), Cassini (1997), Ulysses (1990).

## 4.2 CIRC

Asi nejznámější aplikací RS kódů najdeme u kompaktních audio disků. Skrývá se pod názvem *CIRC* (Cross-Interleaved Reed-Solomon Code). Je ukázkovým příkladem několika triků pro zlepšení vlastností kanálu. Příčin chyb u CD je hned několik, od rozptýlení paprsku, přes bublinky vzduchu v plastu po otisky prstů a škrábance. Je jasné, že nejčastějšími chybami budou dávkové (*burst*) chyby. Je proto výhodné zacházet s 8-bitovým řetězcem jako s jedním symbolem (bytem). Pak například 8 chybných bitů za sebou znamená chybu maximálně dvou symbolů. Používá se také metoda prokládání (interleaving), abychom dávkové chyby více rozptýlili.

**Příklad**: Pokud máme řetězec `PROKLÁDÁNÍ JE DOBRÉ` a nastane dávková chyba na čtyřech symbolech dostaneme `PXXXXÁDÁNÍ JE DOBRÉ`. S obtížemi pak poznáváme co obsahovala původní zpráva. Pokud však data před odesláním zamícháme, například `PÁJRRDEÉOÁ DK NOLÍB`, potom se stejnou chybou `PXXXXDEÉOÁ DK NOLÍB` po opětovném přerovnání symbolů dostáváme `PXOKLXDÁNÍ XE DOBXÉ`. Díky rozptýlení chyby dokážeme zprávu luštit lehčeji.

CIRC používá dvou zkrácených Reed-Solomonových kódů $C_1$ a $C_2$ pracujících s 8-bitovými symboly, tedy nad $\mathbb{F}_{2^8}$. Kódy nemají přirozenou délku $q-1 = 2^8-1 = 255$, ale aby nebyl příliš složitý dekodér, jsou zkráceny na délky $C_1-(28,24)$ a $C_2-(32,28)$ (čísla uvedená v závorce jsou (n,k) – v bytech). Oba tyto kódy mají nadbytečnost 4 a minimální vzdálenost 5 bytů (symbolů). Poměr prodloužení zprávy je 4/3.

Použité 8-bitové symboly dobře korespondují s velikostí jednoho hudebního *vzorku* (*sampl*) 16 bitů. Na popis jednoho vzorku tedy potřebujeme dva symboly. Vzorků je za sekundu zaznamenáno 44100. Pro dva kanály (levý a pravý) je požadovaná datová propustnost $1,4112$ kbps.

Pro popis kódování označím posloupnost vzorků pravého kanálu $\{R_j\}$ a levého kanálu $\{L_j\}$. Nejprve dvanáct vzorků (šest pravého a šest levého kanálu) uspořádáme do *rámce* (*frame*) – příslušnost vzorku k $i$-tému kanálu označím horním indexem:

$$F_i = L_1^i R_1^i L_2^i R_2^i L_3^i R_3^i L_4^i R_4^i L_5^i R_5^i L_6^i R_6^i.$$

Jeden vzorek zabere 16-bitů. Naše kódy pracují nad 8-bitovými symboly. Každý vzorek proto rozdělíme na dvě poloviny:

$$F_i = L_1^i L_{1'}^i R_1^i R_{1'}^i L_2^i L_{2'}^i R_2^i R_{2'}^i L_3^i L_{3'}^i R_3^i R_{3'}^i L_4^i L_{4'}^i R_4^i R_{4'}^i L_5^i L_{5'}^i R_5^i R_{5'}^i L_6^i L_{6'}^i R_6^i R_{6'}^i.$$

Velikost jednoho rámce odpovídá délce vstupního slova pro kód $C_1$. Na řadu nejprve přichází první prokládání. Půlka z každého vzorku (každý sudý symbol) je zpožděna o dva rámce. To znamená, že rámec obsahuje polovinu symbolů rámce kódovaného dva kroky zpět.

$$F_i' = L_1^i L_{1'}^{i-2} R_1^i R_{1'}^{i-2} L_2^i L_{2'}^{i-2} R_2^i R_{2'}^{i-2} L_3^i L_{3'}^{i-2} R_3^i R_{3'}^{i-2} L_4^i L_{4'}^{i-2} R_4^i R_{4'}^{i-2} \\ L_5^i L_{5'}^{i-2} R_5^i R_{5'}^{i-2} L_6^i L_{6'}^{i-2} R_6^i R_{6'}^{i-2}$$

V dalším kroku budeme prokládat zakódovaný rámec délky 28 bytů. V rámci zpozdíme $j$-tý symbol o $4j$ rámců. Jeden rámec se nám proto rozprostře do $4 \times 28 = 112$ rámců. Označím-li symboly jednoho rámce jako $\{C_j\}$ a příslušnost k $i$-tému rámci horním indexem symbolu, vypadá zpoždění následovně:

$$F_i'' = C_1^{i-4} C_2^{i-8} C_3^{i-12} \dots C_{28}^{i-112}$$

Takto zpožděné rámce kódujeme kódem $C_2$. Rámec se tedy dostává na konečnou délku 32 bytů.

Dalším prokládáním jsou opět zpožděny sudé symboly, tentokrát jen o jeden rámec. Následují další „kosmetické“ úpravy na zlepšení vlastností povrchu CD (*EFM* kódování). Ty jsou však z hlediska samoopravných kódů nezajímavé.

Kód $C_2$ je určen k opravě pouze jedné chyby. Tím docílíme toho, že tři chyby povedou k selhání dekodéru (jinak bychom dekódovali na špatné slovo). Při přehrávání hudby je spousta možností, jak nahradit špatný vzorek (interpolace, ztišení), což je lepší, než přehrát chybně dekódovaný vzorek. Pokud nastane selhání, označí dekodér $C_2$ celých 28 symbolů jako výmazy. Ty jsou díky prokládání rozloženy do 28 kódových slov (rámců) $C_1$. Kód $C_1$ je následně většinou použit pouze pro dekódování výmazů.

Pokud se v kódovém slově $C_1$ objeví více jak 4 výmazy, vrátí dekodér $C_1$ slovo délky 24 symbolů se všemi symboly označenými jako vymazané (12 vzorků).

Standard CIRC obsahuje pouze popis kódu jako takového, nikoliv procesu dekódování. Uvedený postup je ten nejčastější. Záleží na výrobci čtecí mechaniky jaké metody použije. Vše se pak promítá do koncové ceny.

## 4.3 DVD

Dalším z rodiny optických médií jsou *DVD* (Digital-Video-Disk). Pro kódování používá dvou RS kódů nad osmi-bitovými symboly, spojených dohromady v součinovém kódu *RS-PC* (Reed-Solomon Product Code). Konkrétně jsou to kódy $C_1 - (182, 172)$ a $C_2 - (208, 192)$ opět nad $\mathbb{F}_{256}$. Výsledné součinový kód má parametry $(182 \cdot 208, 172 \cdot 192, 17 \cdot 11 - 1)$.

## 4.4 Další aplikace

Výčet aplikací by mohl pokračovat stále dál. Reed-Solomonovy kódy též najdeme například v bezdrátových sítích WiFi, digitální televizi DVB-T, mobilní komunikaci GSM, diskových RAID polích, technologiích xDSL a mnoha dalších.

# Literatura


[1] F.J. MacWilliams, Sloane N.A.: *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977

[2] Irving S. Reed, Xuemin Chen: *Error-Control Coding for Data Networks*, Kluwer Academic Publishers, Norwell, 1999

[3] James L. Massey: *Shift-Register Synthesis and BCH Decoding*, IEEE, 1969

[4] Robert J. McEliece, James B. Shearer: *A Property of Euclid's Algorithm and an Application to Padé Approximation*, SIAM, 1978

[5] Tomáš Kaiser: *skripta k přednášce Samoopravné kódy*, MFF-UK, 2004

[6] Ladislav Bican: *Lineární algebra a geometrie*, Academia, 2000