**Samoopravné kódy**

Obsah:

Úvod 1

Galoas field a field generator 7

Reed- Solomonovy kódy úvod 11

Příklad NASA, ten jejich 12

Příklad NASA, oprava chyb 15

Souhrn postupu pro kódování a dekódování 25

Příklad NASA podruhé – složitější chyby 26

Příklad Kubalík 3 – jiný field generator, 3 chyby 32

Implementace RS kódů u GPON a VDSL 42

**Úvod**

**FEC - forward error correction**

Lze říci, že je to výrazně chytřejší bratr CRC. CRC uměl opravit jednu chybu, FEC umí opravit hodně chyb .

úvodni přehled a názvy kódů máte na

<https://www.techtarget.com/searchmobilecomputing/definition/forward-error-correction>

Kódy jsou následující (podrobněji v odkazu výše)

Bose-Chaudhuri-Hocquenghem (BCH)

Hamming

Low-density parity check (LDPC)

Reed-Solomon (RS)

Turbo

Nadále se budeme věnovat pouze Reed-Solomonovým kódům, protože se používají u GPON a VDSL

**GPON – doporučení G.984.3**

Reed-Solomon (RS) code is a block-based code, which takes a data block of constant size and adds

extra parity bytes at the end, thus creating a codeword. Using those extra bytes, the FEC decoder

processes the data stream, discovers errors, corrects errors and recovers the original data.

Reed-Solomon code is specified in [b-ITU-T J.81].

The most common RS code is RS(255,239), where a 255-byte codeword consists of 239 data bytes

followed by 16 parity bytes. RS(255,239) is used in [b-ITU-T G.975] and [b-ITU-T G.709].

When using a block-based FEC, original data is preserved. Therefore, by ignoring the parity bytes,

even if other the side does not support FEC, the original data can be processed.

Block-based FEC is not efficient for very high BER (e.g., for 10–3 BERs, decoding error will be

generated).

Z  BER 10-4 udělá FEC BER 10-15 (pokud netušíte, co je BER, gymnásium ! )

Rámec RS kódu u GPON má tedy 255 bytů , z nich je 239 použito pro skutečný přenos dat, dalších 16 je pro zabezpečení. Jedná se o systematický kód.

The Hamming distance of the RS(255,239) code is dmin = 17. The code can correct up to 8 symbol

errors in the FEC code word when it is used for error correction. The FEC can detect up to 16 symbol

errors in the FEC code word when it is used for error detection capability only

„symbol error“ tady znamená chybu v bytu, nikoli bitu. Jak uvidíme dále, jako symbol je použit byte. To znamená, že pokud v jednom byte je osm bitů špatně, je to stále jedna chyba v jednom symbolu. A těchto špatných symbolů je RS(255,239) schopen opravit osm. Udělejte si pokus: jakému BER to odpovídá ?

VDSL – doplnit podle Fabiniho . Cetin označuje své profily „bez retransmise“ – používá FEC, a „s retransmisí“ – používá CRC

V knize od Fabiniho je mnoho vzorců, ale žádná pořádná odpověď.

AI dodala tuto odpověď:

„ U systému VDSL2 se často používá Reed-Solomonův kód s parametry n=255 a k=239. To znamená, že každé kódové slovo obsahuje 255 symbolů, z nichž 239 jsou datové symboly a 16 jsou zabezpečovací symboly “

Ale ono se tomu nesmí moc věřit, AI možná spatlala dohromady VDSL a GPON, to ono to dělá rádo.

Úvod podle

https://tomverbeure.github.io/2022/08/07/Reed-Solomon.html

**symbol**

A symbol is the smallest piece of information. In many coding methods, a symbol is a single bit, with only two possible values. Reed-Solomon coding, however, uses symbols that have more than 2 values.

Takže jako symbol bývá použit byte , který má 256 hodnot . U CRC byl jako symbol použit bit. V našich úvahách v tomto textu budeme jako symbol používat čtyři bity, aby na výkladu bylo něco vidět.

**a message zpráva**

A message is the original piece of information that need to be encoded. A message is a sequence of symbols.

For example, a message could be “Hello world “, where each character is a symbol.

**a message word** to je kus zprávy

A message word is a fixed length section of the overall message. Reed-Solomon coding is a block coding algorithm where a message is split up into multiple message words, and the encoding algorithm is performed on each message word without any dependency on previously received message words.

Message words are usually indicated with the letter *m*, and *k* is often used to indicate the number of symbols in the message word. In other words, *k* is the size or the length of the message word.

In vector notation: *m*=(*m*0,*m*1,...*mk*−1)

For example, if a message word is defined as having a size 4, our previous message would be split into the following message words: (‘H’, ‘e’, ‘l’, ‘l’), (‘o’, ‘ ‘, ‘w’, ‘o’), (‘r’, ‘l’, ‘d’, ‘ ‘).

Takže zpráva „Nazdar Tomáši“ , pokud má message word o velikosti 4 písmena, obsahuje čtyři message word „Nazd“ „ar T“ „omáš“ „i “ doplnili jsme 3 mezery

**an alphabet abeceda**

An alphabet is the full set of values that can be assigned to a symbol.

*Q* is often used as the number of values in an alphabet.

When a symbol is a single bit, the alphabet consists of 2 values, 0 and 1.

If we had a system where messages only consist of upper case and lower case letters and a space, then an alphabet could be (‘A’, …, ‘Z’, ‘a’, ‘b’, ‘c’, … , ‘z’, ‘ ‘), and the size of the alphabet would be 53. Almost all coding algorithms require the ability to perform mathematical operations such as addition, subtraction, multiplication and division on the symbols of a word, so don’t expect to see this kind of alphabet in the real world!

In practice, the alphabet of most coding algorithms is a sequence of binary digits. 8 bits is common, but it doesn’t have to be. I hesitate to call it an 8-bit *number*, because that would suggest that regular integer math can be used on it, and that’s almost never the case. Instead, the operations of such an alphabet use the rules of a Galois field.

Pokud je symbol byte, pak samozřejmě abeceda má 256 prvků . Pokud je symbol bit, tak má abeceda dva prvky.

**a code word kódové slovo**

A code word is what you get after you run a message word through an encoder.

In the case of a Reed-Solomon encoder, a code word has a length *n* symbols, where *n*>*k*

and the symbols are using the same alphabet as the message word.

Code words are often indicated with a vector *s*=(*s*0,*s*1,...,*sn*−1)

Kódové slovo je to, co vychází z kodéru. Je to to, co patří do kódu. Říkali jsme tomu kódová složka. U CRC je to zpráva a za ní zbytek po dělení, tam samozřejmě bylo message == mesage word

When our message of three 4-symbol message words is converted into three 6-symbol code words, it could have the following code words:

(‘H’, ‘e’, ‘l’, ‘l’, ‘a’, ‘b’), (‘o’, ‘ ‘, ‘w’, ‘o’, ‘g’, ‘t’), (‘r’, ‘l’, ‘d’, ‘ ‘, ‘m’, ‘j’)

Notice how the first 4 symbols of each code word are the same as their corresponding message word, but that 2 symbols are added for error detection and correction. This makes it a [systematic code](https://en.wikipedia.org/wiki/Systematic_code). As we’ll soon see, not all coding schemes are systematic in nature, but it’s a nice property to have.

Pro kódování platí, že do koderu vchází k symbolů a vychází n symbolů .

Pokud je k < n , pak koder přidává redundanci a nazývá se

pokud je k==n , pak se jedná o scrambler

pokud je k > n , pak se jedná o kompresní kód

For Reed-Solomon codes, the length *n*

* of a code word must be smaller or equal than the number of values in the alphabet that it uses.

Tahle nenápadná poznámka je zcela zásadní. Říká totiž, jaký je vztah mezi velikostí symbolu a maximální možnou délkou rámce

Pro symbol bit je “number of values in the alphabet” rovna ……. , takže maximální počet symbolů v kódovém slově je ….. , takže vidíme, že …..

Dále budeme pracovat se symboly, které mají 4 bity. Potom “number of values in the alphabet” je ….. a maximální délka kódového slova je ……..

Pro GPON nebo VDSL je symbol byte, takže počet hodnot v abecedě je ……, délka code word je …..

Code word pro RS kód - za message word, který má k symbolů, se přidá zabezpečovací část, která má 2\*t symbolů . Code word má n symbolů n = k + 2\*t

|  |  |
| --- | --- |
| Code word | |
| Message word | zabezpečení |
| k symbol | 2\*t symbolů |
| Celkem n symbolů | |

Symbol může být leccos, nejčastěji bit nebo byte . U RS kódů v telekomunikacích **byte !!!!**

n = k + 2\*t

t – error correction capability, tedy kolik symbolů se dá opravit . t je rovno u RS kódů polovině zabezpečovacích symbolů.

RS kód se označuje RS( n , k)

U GPON je to kód RS( 255,239 ) tedy kódové slovo má 255 bytů, message word má 239 bytů, 2\*t je 16 bytů , t je 8 . Kód tedy opraví 8 symbolů ve slově délky 255 symbolů, symbol je byte. To znamená, že osm bitů špatně v jednom byte je jedna chyba v jednom symbolu.

u RS kódů se pojmem „kódování“ označuje zobrazení množiny vstupních symbolů (k-tuple), která má k prvků, do množiny výstupních symbolů (n-tuple), která má n prvků

k < n

kód musí být redundandní , to dá rozum

Soubor NASA.pdf v aktuálním adresáři je stažen z

<https://ntrs.nasa.gov/api/citations/19900019023/downloads/19900019023.pdf>

<https://ntrs.nasa.gov/citations/19900019023>

a odkaz na něj je v

<https://tomverbeure.github.io/2022/08/07/Reed-Solomon.html>

tento odkaz má na konci seznam zdrojů, a tam je ten paper NASA

NASA.pdf má na úvod všechno to, co jsme se užili o CRC – Booleova algebra, XOR, sčítání modulo 2 , generující polynom, kořeny generujícího polynomu ( str. 19 )

očekáváme tedy jako samozřejmost následující znalosti:

EX-OR - sčítání modulo 2

generující polynom

dělení polynomů, zbytek po dělení

primitivní polynom

kořen polynomu

mocniny kořenů polynomu

Pan William A. Geisel, to je ten, co napsal příručku NASA, upozorňuje:

Existuje mnoho a mnoho tutoriálů, které se zabývají samoopravnými kódy. Bohužel velmi často používají pro demonstraci činnosti jako symbol bit. To jim umožní udělat výpočty velice hezké a krátké, ale není na tom nic vidět. V příručce NASA bude použit symbol o délce 4 bity, tedy abeceda má 16 položek. To sice zkomplikuje výpočty, ale zase na tom je vidět, co a jak se musí udělat. Skutečný RS kód pro GPON a VDSL má jako symbol byte. To už je pro výuku moc, potom má abeceda 256 položek a výpočty jsou neskutečně dlouhé.

A moje poznámka:

AI funguje tak, že shromažďuje obrovské množství dat z Internetu . Protože ( jak řekl pan Geisel ) naprostá většina tutoriálů na Internetu používá jako symbol bit, bude i AI fungovat se symbolem délky jednoho bitu (protože práci pana Geisela nenajde nebo ji zahodí jako minoritní). A potom vám zcela logicky dodá jako odpověď na vaše dotazy pitomosti. Takže doporučuji, abyste si všechny příklady počítali sami.

strana 18:

zde je použit generující polynom P(x) = x4 +x +1

a dále se počítají mocniny kořenů generujícího polynomu, to známe z CRC

m-tuple - vector - naše vyjádření pomocí binárních čísel

4-tuple

A tady už začíná Galoas field

máme mocniny kořenů generujícího polynomu x4 +x +1

Tento polynom se u Galoas field nazývá „field generator“. Jak uvidíme, polynom skutečně vytváří Galoasovo pole.

kořen se zde značí α alfa, my jsme ho u CRC značili a ( tím myslíme písmenko a )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mocnina kořenu | |  | m-tuple |  |
| α-∞ | -∞ |  | 0000 | Nula. Opravdu nula |
| α0 | 0 |  | 0001 |  |
| α1 | 1 |  | 0010 |  |
| α2 | 2 |  | 0100 |  |
| α3 | 3 |  | 1000 |  |
| α4 | 4 |  | 0011 |  |
| α5 | 5 |  | 0110 |  |
| α6 | 6 |  | 1100 |  |
| α7 | 7 |  | 1011 |  |
| α8 | 8 |  | 0101 |  |
| α9 | 9 |  | 1010 |  |
| α10 | 10 |  | 0111 |  |
| α11 | 11 |  | 1110 |  |
| α12 | 12 |  | 1111 |  |
| α13 | 13 |  | 1101 |  |
| α14 | 14 |  | 1001 |  |

Jak si pamatujeme, po α14 následuje α15 , ale to je α0 . Celé pole se tedy cyklí.

Systém mocnin kořenů generujícího polynomu se nazývá „Galoas field“ . Pro polynom x4 +x +1 je v tabulce výše.

Jednotlivé prvky tabulky mohou být označeny několikerým způsobem:

- power representation: první sloupec tabulky např. α9 dále zůstaneme v tomto řádku

polynom: není v tabulce, ale umíme to α3 + α1

vector m-tuple u nás 4-tuple 1010

Všechny tyto zápisy vyjadřují stejný prvek Galoas fileld, jsou ekvivalentní. Použijeme ten, který se nám lépe hodí – pro násobení power representation, pro sčítání vector

Do Galoas field je přidán řádek 0 - v něm je číslo 0. Opravdu číslo nula. To se nedá vyrobit žádným dělením z cyklením, proto cyklení začíná od prvního řádku 0001 . Ale nulu potřebujeme, je tam. Tím také dostaneme 16 elementů Galoas field, protože stupeň generujícího polynomu je 4 .

Mocnina kořene pro nulu je ( minus nekonečno ) . Pokud vám není jasné, že je to fakt pravda, kontaktujte učitele matematiky.

Galoasovo pole vytváří algebru. Algebra je množina { čísla, operace } . Pro operace musí platit nějaké podmínky, nám to zatím stačí takto. Výše uvedené Galoas field má 16 čísel. Zkusíme nějaké operace

**Sčítání**

úplně normálně sečteme polynomial representation nebo vector representation, nezapomeneme, že + je XOR .

některé příklady:

α12  + α13  = α1

α8  + α5  = α4

α3  + α6  = α2

α12  + α12  = 0

α7  + α11  = α8

Jednotlivá čísla si zapište pomocí vektorové representace, sečtěte pomocí EX-OR, a převeďte zpátky do power representation.

Můžeme udělat ještě další věc: jednotlivá čísla v Galoasově poli si můžeme označit pomocí desítkové hodnoty “vector representation” . Pak dostáváme (sčítáme stejná čísla jako výše):

15 + 13 = 2

5 + 6 = 3

4 + 12 = 4

15 + 15 = 0

11 + 14 = 5

Ještě jednou: znaky 13 15 5 6 3 ……. atd jsou použity jenom pro označení čísla v Galoasově poli. Zkonstruovali jsme je tak, že jsme vzali desítkovou hodnotu “vector representation”.

Znak “15” tedy znamená číslo α12 nebo 1111 . “číslo” , o kterém zde mluvíme , je řádek Galoasova pole.

Protože znaky 13 15 5 6 3 ……. Jsou jenom označení čísla v Galoasově poli, klidně si můžeme určit jiné znaky. Čísla můžeme označit jenom mocninou alfa . Pak by výše uvedená tabulka pro sčítání vypadala takto:

12 + 13 = 1

8 + 5 = 4

3 + 6 = 2

15 + 15 = -∞

7 + 11 = 8

V příručce NASA je na straně 28 a 29 tabulka pro sčítání, můžete ji používat. (“strana” je to číslíčko, které je napsáno psacím strojem dole na stránce, ne to, co vám ukáže AcrobatReader)

V literatuře se často používá některý z uvedených způsobů vyjádření čísla v Galoasově poli. Autoři pravidelně neuvádějí, co jejich znaky znamenají, takže číst takovéto materiály je opravdu zážitek.

**Odčítání** – již víme, že je stejné jako sčítání. Proto také v naší algebře neexistuje relace větší-menší. A prvky Galoas filed na straně 13 NESJOU uspořádány podle velikosti, protože nelze říct, který prvek je větší a který menší. (matematický šťoural vám řekne, že se i v naší algebře dá zavést metrika, a potom by existovalo větší-menší. Ale metriku zavádět nebudeme.)

V souboru generpoly.xlsm máte na listu odcitatko nástroj ke sčítání prvků Galoas field, do D15 – D29 zadáte power representation čísla, jenom mocninu, a dole vyjde součet, opět power representation. -100 znamená 0 , tedy alf na minus nekonečno. „nezadáno“ , „mezera“ znamená také číslo 0 .

**Násobení:** úplně jednoduše vynásobíme „power representation“ , tedy sečteme mocniny u alfa. Pokud vyjde více než 15 , tak 15 prostě odečteme, protože Galoas field se po 15-ti cyklí.

Sečteme exponenty u power representation , tedy αi , a je to. Pokud vyjde moc velká mocnina, použijeme toho, že se Galoas Field cyklí, pro stupeň 4 po 15 , a odečteme 15 . Pokud je to stále moc, tak zase odečteme 15. Tím děláme operaci modulo , tedy zbytek po dělení. Protože všechny zápisy prvků Galoas field jsou totožné, můžete klidně vynásobit polynomial representation. Musí to vyjít stejně. Pro generující polynom stupně N tedy děláme operaci modulo ( 2N-1 )

**Dělení:** úplně jednoduše vydělíme „power representation“ , tedy odečteme mocniny u alfa. Pokud vyjde méně než 0 , tak prostě přičteme 15 , protože Galoas field se po 15-ti cyklí.

**primitive element -** dál to nepoužívají, klidně přeskočte

stupeň, na kterém se to cyklí, je 15 ( o 1 více než konec tabulky ) 15 = 24 – 1 4 je stupeň generujícího polynomu

„primitive element“ je ten, který není soudělný s číslem 15 .

Tím je míněno, že stupeň (alfa na p) αp  není soudělný s 15 .

ten, který je nesoudělný

0

1 ANO 0010 tady fakt nevím, proč, ale ta 1 je taková magická, asi se nepovažuje za soudělnou

2 ANO 0100

3

4 ANO 0011

5

6

7 ANO 1011

8 ANO 0101

9

10

11 ANO 1110

12

13 ANO 1101 13 prvočíslo

14 ANO 1001 15 = 5\*3 14 = 2\*7

a tohle nám souhlasí se stranou 24 NASA,

Pokud si teď zvolíme stupeň generujícího polynomu 7 , tak se nám Galoas field cyklí u 127. Protože 127 je prvočíslo, tak úplně všechny prvky tohoto Galoas fileld jsou primitive elements

Pokud si teď zvolíme stupeň generujícího polynomu 8 , tak se nám Galoas field cyklí u 255. Protože 255 není prvočíslo a 255 = 5 \* 3 \* 17 , tak jenom některé prvky Galoas fileld jsou primitive elements. Vypadnou ty, jejichž stupeň je dělitelný 5 3 17 .

Konec od “primitive element”

Zpět ke field generator x4 +x +1

Když jsme počítali mocniny kořenů primitivního polynomu u CRC, tak jsme tam mocniny α1 α2 ….  prostě doplnili. Na straně 25 je další rozšírění. α1 totiž nemusí být jenom X , ale může to být jakýkoli primitive element z Galoas field, které je generováno generujícím polynomem , u nás x4 +x +1 .

Netuším, zdali to k něčemu bude, kdyžtak doplníme strana 26 NASA

V dalším se to občas zmiňuje, ale všechny příklady začínají u alfa na prvou, takže to asi můžeme vypustit

strana 30 u NASA

Až se naučíte pojmy jako grupa, okruh, těleso …. v matematice, můžete si přečíst stranu 32

tabulka 1.4.5-2 na straně 29: jedná se o tabulku sčítání. V tabulce jsou jenom mocniny alfa, tedy αk , a v tabulce je jenom číslo k . Vzhledem k tomu, že všechny zápisy jsou identické, můžeme si to takto dovolit. Číslu 1 pak samozřejmě odpovídá k == 0 ( α0 == 1 ) , a máme problém s číslem 0 – nula. To je zobrazeno jako „minus nekonečno“ v souladu s tím, že α(minus nekonecno) == 0 .

a dále jsme na straně 33

**Blokové kódy Reed-Solomonovy**

Na vstupu koderu máme k prvků (k-tuple) , a ty jsou zobrazeny do n prvků (n-tuple) na výstupu .

Lineární blokový kód je takový, u kterého součet libovolných dvou kódových slov je zase kódové slovo. Ano, odzkoušeno

RS kód je nebinární, lineární, cyklický, BCH

Cyklické kódy jsou podmnožina lineárních kódů .

Další vlastnost: jakýkoli cyklický posuv kódového slova je zase kódové slovo ???? tohle se mi nepodařilo ověřit. Nebo jsem to špatně pochopil

„kódové slovo“ je samozřejmě to po zakódování, takže doleva posunutá zpráva + zbytek po dělení generujícím polynomem

Délka bloku n = 2m-1 kódových symbolů m je u nás počet bitů v symbolu

Number of parity-checks n – k <= 2\*t ( 2 krát t ) kódových symbolů

RS kód se označuje RS( n , k)

U GPON je to kód RS( 255,239 ) takže parity check je 255 – 239 = 16

m = 8 , n = 255 , a tajemné t je 8

n – k <= 2\*t 255 – 239 = 8 \* 2

Na straně 43 :

Délka bloku n = 2m-1 kódových symbolů

Number of parity-checks n – k = 2\*t ( 2 krát t ) kódových symbolů

t je počet chybných symbolů, které je kód schopen opravit. Symbol je byte !!!!!

Od strany 49 se text zabývá kódováním RS kódu.  Jako příklad je tam RS( 15,9 ) , t je 3 .

15 – 9 = 2\*3

Message word má 9 symbolů. Parity check má 6 symbolů. Náš kód je tedy schopen opravit 3 symboly.

K zabezpečení použijeme naše Galoasovo pole na straně 5 , field generator 10011

Počet prvků Galoasova pole je 16 , tedy 24 , m je 4

Kód je RS( 15,9 ) , tedy kódové slovo má 15 symbolů, message word má 9 symbolů. No a teď začíná ta pravá magie. Symbol má 4 bity – je to přece prvek našeho Galoas field.

A zkusíme si zabezpečit nějakou zprávu

Vezmeme si zprávu od NASA

Takže NASA má field generator x4 +x +1 , mocniny kořenů jsou v tabulce na str. 5 , message word od NASA je

0000 0000 0000 0000 0000 0000 0000 1110 0000

Mezery mezi čtveřicemi bitů jsou jenom proto, aby se zpráva dala snáze číst. Zpráva má 9 symbolů a 36 bitů . Zprávu přepíšeme pomocí „power representation“ , viz dále

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | α11 | 0 |

Jak jsem už mnohokrát řekl, čučením do skript se nic nenaučíte. Takže se hbitě podívejte na stranu 5 , najděte si čtveřici bitů v našem galoas field a podle toho doplňte mocninu kořene alfa na … . Připomínám, že nula je doopravdy nula, takže čtveřice bitů 0000 se zobrazí jako 0 .

Prvek α11 je posunut o jedno místo doleva, takže naše message word je M(x) = α11.X

Potřebujeme generující polynom. Ten se spočítá jako

G(X) =

To podivné PI je product, to znamená vynásob. Násobí se tedy ty závorky, které jsou za tím, a proměnná se přitom zvyšuje od 1 do 2\*t. No a alfa se postupně umocňuje na vyšší a vyšší exponent. U nás je 2\*t = 6 , takže násobíme do 6. A za mocniny alfa samozřejmě dosazujeme to, co je v Galoas field. Za vyšší mocniny alfa dosazujeme nižší mocniny, např. takto:

α4 = α1 + 1

α11 = α3 + α2 + α1

a tak dále a tak dále, podívejte se do tabulky na straně 5.

Při výpočtu používá NASA pojem FCR – first cosecutive root.

Velmi zhruba se jedná o následující: Galoas field se cyklí. Při našich výpočtech tedy nemusíme začínat od α1 , ale můžeme začít „někde“ , a postupovat dál. Výpočty musí fungovat. Až dojdeme na konec Galoas field, pak prostě přetečeme do 1 , anormálně tím polem postupujeme dá.

Protože autoři NASA jsou vědecky korektní, uvádějí všude tuto možnost. Naštěstí ono sami vždy začínají u α1 . Proto jsem ve vzorcích v této příručce vždy FCR vypustil, abych vás tím neděsil.

U NASA na straně 56 vychází

G(X) = X6+ α10X5+ α14X4+ α4X3+ α6X2+ α9X+ α6

V souboru generpoly.xlsm máte na listu delime opravdické dělítko, které lze použít pro jakékoli polynomy až do stupně 14. Do řádku 6 zadáte mocniny alfa u odpovídajícího Xi , nejdříve dělenec, potom dělitel. Pokud chcete zadat číslo 0 – nula, napíšete !N (jako na minus nekonečno) . Pokud zadáte 0 , pak vězte, že jste zadali α0, tedy 1 .

Dále musíte (nebo ještě lépe před tím) zadat na listu „koreny“ field generator, v buňkách N7 – R7 . Udělal jsem to pro mnohočlen stupně 4 , pokud chcete něco jiného, upravte si to.

Vrátíme se na list delime, a spustíme makro stisknutím Cntrl a (a je malé písmeno a) . No a ono to počítá, a dole vyjde zbytek !

Přátelské upozornění: pokud stisknete Cntrl a na jiném listu, tak vám to tento list totálně přeorá. Ostatně se podívejte, jak je to makro udělané, jste už velcí a programovat umíte.

No a teď to budeme dělit generujícím polynomem. Stejně jako u CRC tu celou zprávu posuneme doleva, v našem případě o tolik symbolů, kolik činí zabezpečovací část code word. To je u nás o 6 symbolů. To prostě znamená, že naši zprávu vynásobíme x6 .

( α11.X ) \* x6 = α11.X7

A tohleto celé teď vydělíme generujícím polynomem ( to NENI “field generator”) , zjistíme zbytek a ten přidáme za původní zprávu.

Použijeme dělítko ze souboru generpoly.xlsxm , list delime



Zbytek vychází

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X5 | X4 | X3 | X2 | X1 | 1 |
| 5 | 4 | 3 | 2 | 1 | 0 |
| 8 | 10 | 4 | 14 | 8 | 12 |

A celá zabezpečená zpráva



Po posunutí, vydělení generujícím polynomem a přidání zbytku vyšlo u NASA kódové slovo:

C(x) = α11 \* X7 + α8 \* X5  + α10 \* X4 + α4 \* X3  + α14 \* X2  + α8 \* X1  + α12

Po přepsání do bitů vychází:

1110 0000 0101 0111 0011 1001 0101 1111

Předpokládám, že vás zarazilo, že tahle zabezpečená zpráva je kratší než ta původní. Nezapomeňte ale, že původní zpráva má zleva mnoho nul, které se tady vypouštějí.

Ještě jednou opakuji generující polynom NASA G(X) = X6+ α10X5+ α14X4+ α4X3+ α6X2+ α9X+ α6

Až to budete zkoušet pomocí dělítka, nezapomeňte, že u X6 je koeficient 1 , tedy α0 . A dále nezapomeňte v tom Excelu používat vložit jinak – vložit hodnotu. Jinak se Excel pokusí vložit vzorce, a zcela tím zlikviduje sestavený algoritmus.

Jenom dále poznamenávám (a po kdovíkolikáté opakuji), že u GPON a VDSL se používá RS(255,239) , kde je symbol byte. Takže Galoasovo pole má 256 řádků, mocniny u alfa stoupají do 254 . Takže si zhruba představte, jak vypadá ten mnohočlen, kterým je reprezentována naše zpráva u RS(255,239) .

**Dekódování RS kódu**

No, ono to vlastně není dekódování. Kód je systematický, takže zpráva je to, co je na začátku, když odstraníme “parity check” symbols. Ale následující kapitola se zabývá tím, jak poznat, kde je chyba, a jak ji opravit.

NASA - CHAPTER 4 REED-SOLOMON DECODING

Strana 59 . To je to číslíčko, které je dole na stránce v textu, NE to, co udává AcrobatReader

THE DECODING PROCESS IS A FIVE-STAGE PROCESS:

1. Calculate the syndrome components from the received word.

2. Calculate the error-locator word from the syndrome components.

3. Calculate the error locations from the error-locator numbers which are from the error-locator word.

4. Calculate the error values from the syndrome components and the error-locator numbers.

5. Calculate the decoded code word from the received word, the error locations, and the error values.

Tohle vyblila AI:

Oprava chyb pomocí Reed-Solomonova (RS) kódu zahrnuje několik kroků. Zde je obecný postup:

1. **Přijetí kódového slova**: Přijměte kódové slovo, které může obsahovat chyby. Toto kódové slovo má délku n (takhle jsme to vždycky značili, RS(15,9) RS(n,k) ).
2. **Výpočet syndromů**: Vypočítejte syndromy z přijatého kódového slova. Syndromy jsou výsledkem dosazení přijatého kódového slova do generujícího polynomu. Pokud jsou všechny syndromy nulové, kódové slovo je bez chyb.
3. **Určení polynomu chyb (error-locator)**: Pokud jsou syndromy nenulové, použijte algoritmus, jako je Berlekamp-Massey, k určení polynomu chyb. Tento polynom popisuje pozice a velikosti chyb v kódovém slově.
4. **Určení kořenů polynomu chyb**: Najděte kořeny polynomu chyb pomocí Chienova vyhledávacího algoritmu. Kořeny odpovídají pozicím chyb v kódovém slově.
5. **Oprava chyb**: Pomocí Forneyova algoritmu vypočítejte hodnoty chyb a opravte je na příslušných pozicích v kódovém slově. (Poznámka: u CRC jsme měli symbol bit, takže stačilo, abychom zjistili, kde je chyba, a bit jsme prostě negovali. Pokud máme symbol větší než jeden bit, potřebujeme vědět, o kolik jsme se vzdálili od původní hodnoty. U VDSL a GPON je symbol byte, jak už v tomto textu mnohokrát zaznělo )
6. **Rekonstrukce původního kódového slova**: Po opravě chyb získáte původní kódové slovo, které by mělo být bez chyb.

**Krok 1: Výpočet syndromů**

Syndromy Si​ se vypočítají dosazením kořenů generujícího polynomu do přijatého kódového slova r(x). Pro každý syndrom Si​ platí:

Si​ =r(αi)

The word "syndrome" is defined in a dictionary as a group of signs and symptoms that occur together and characterize a particular abnormality. It is also defined as a set of concurrent things that usually form an identifiable pattern. In the coding application syndrome components Si are these individual characteristics that characterize a particular error pattern (abnormality). The syndrome s(X) is simply the accumulation of these characteristics where Si=s(ai).

Syndrom se počítá tak, že do zprávy se dosadí postupně mocniny kořenů z Galoasova pole, mocniny jsou od 1 do 2\*t 2t je u RS(n,k) 2\*t = n-k , jak už dávno víme, takže 2t je 6

Dosazujeme tedy α1 , α2 , α3 , α4 , …. , α2\*t ,

Počet syndromů je tedy stejný jako počet zabezpečovacích symbolů u RS kódu.

U našeho příkladu RS( 15,9 ) tedy dosazujeme : α1 , α2 , α3 , α4 , α5 , α6

Prostě úplně normálně v kódovém slově místo X napíšeme α1 . Prostě dosadíme .

C(x) = α11 \* X7 + α8 \* X5  + α10 \* X4 + α4 \* X3  + α14 \* X2  + α8 \* X1  + α12

Po dosazení

S(x) = α11 \* α7 + α8 \* α5  + α10 \* α4 + α4 \* α3  + α14 \* α2  + α8 \* α1  + α12

To vynásobíme, posčítáme, a něco vyjde.

Račte si to spočítat, vyjde 0 . To je proto, že ve zprávě není žádná chyba.

Zkusíme dosadit α2 .

S(x) = α11 \* α14 + α8 \* α10  + α10 \* α8 + α4 \* α6  + α14 \* α4  + α8 \* α2  + α12

Vyjde také nula, protože jsme nikde neudělali žádnou chybu. Spočítejte si to, u mocnin větších než 14 použijte cyklení, převeďte na vektorovou representaci a sčítejte, samozřejmě EX-OR .

A tak dále a tak dále, až do α6 .

V souboru generpoly.xlsxm máte na listu syndromy výpočet syndromů. Do řádku 6 vložíme koeficienty alfa na …, v řádcích 9 – 14 jsou vypočtené syndromy. Mocniny alfa jsou větší než 14 , takže v dalších řádcích výsledek „odcyklíme“ . Dále musíte (to už tam není automaticky) přehodit řádky na sloupce, v Excelu je to CntrlC a transponovat, samozřejmě „vložit jinak - hodnoty“ . Z řádků děláme sloupce proto, abychom následně mohli použít list „odcitatko“. Pokud se v nějaké buňce objeví #HODNOTA! , znamená to, že požítáte s !N, tedy s nulou. Buňku vypusťte. Nebo si to nějak upravte. Na listu odcitatko zadáte do buněk D16 – D29 hodnoty mocnin alfa pro syndrom, a dole je výsledek. Do sloupce se občas nevejde celý syndrom, pak to musíte udělat nadvakrát.

Ještě jednou opakuji: nejdříve jsme spočítali syndromy u zabezpečené zprávy bez chyb, a všechny vyšly 0 . Tak se pozná, že zpráva došla správně.

V příručce NASA potom udělali chybu v kódovém slově.

Code word bez chyby u NASA .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1001 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | α11 | 0 | α8 | α10 | α4 | α14 | α8 | α12 |

Udělali chybu u X8 a u X2

Code word s chybami u NASA .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 1110 | 0000 | 0101 | 0111 | 0011 | 1000 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | α11 | 0 | α8 | α10 | α4 | α3 | α8 | α12 |

Takže kódové slovo s chybou je

C(x) = X8 + α11 \* X7 + α8 \* X5  + α10 \* X4 + α4 \* X3  + α3 \* X2  + α8 \* X1  + α12

Tohle je NASA, strana 63

Spočítáme syndrom, tentokrát zkusíme třeba α4 , takže počítáme syndrom S4 .

S4(x) = α32 + α11 \* α28 + α8 \* α20 + α10 \* α16 + α4 \* α12  + α3 \* α8 + α8 \* α4  + α12

S4(x) = α32 + α39  + α28 + α26  + α16  + α11  + α12   + α12

S4(x) = α32 + α39  + α28 + α26  + α16  + α11

S4(x) = α2 + α9  + α13 + α11  + α1  + α11

S4(x) = α2 + α9  + α13 + α1  = 0001 vektorově, to je alfa na 0 , to je 1 , souhlasí s NASA

NASA má:

S1 = 1

S2 = 1

S3 = α5

S4 = 1

S5 = 0

S6 = α10

Index u S znamená, na kolikátou jsme umocnili alfa při jeho dosazení. Takže výše námi spočtený syndrom je S4 .

A ze syndromů se spočítá, ke je chyba a jak je velká.

Opět máte v souboru generpoly.xlsm počítátko na syndromy. Je na listu syndromy . Do řádku 6 zadáte mocniny alfa, opět nula je !N . V řádcích pod tím je výpočet podle výše uvedené rovnice, v řádcích 17 – 22 jsou hodnoty odcyklené. Dále vezmete tyto řádky a transponujete si je někam dolů, ti znamená zaměníte řádky za sloupce. Je to proto, abychom mohli použít sčítátko na listu odcitatko . Pokud jste někde použili !N , bude Excel dělat nějaké nesmysly. Tyto řádky vypustíte, odpovídají číslům 0 .

A pak sloupec, který odpovídá počítanému syndromu, zkopírujete a vložíte na list odcitatko počínaje buňkou D16 . Nevejde se tam celý rozsah 15 symbolů, takže je užitečné mít někde v používaném mnohočlenu nuly - !N . Opět udělejte „vložit hodnoty“ . No a dole v buňce D32 je výsledek, mocnina alfa. Hodnota -100 znamená na minus nekonečno, tedy číslo 0 – nula. Pokud v buňce D16 – D29 není nic, tak to také znamená číslo 0 .

První otázka je, zda jsou v kódovém slově nějaké chyby. Pokud jsou úplně všechny syndromy rovny 0 , pak je zpráva bez chyb.

Pokud je několik syndromů nenulových, pak ve zprávě nějaké chyby jsou. Ale zatím ještě nevíme, kolik je tam chyb. Nám vyšlo, že jeden syndrom je nulový, pět syndromů není nula**. To ale neznamená, že by ve zprávě bylo pět chyb !** K počtu chyb vede ještě dosti dlouhá cesta.

Dále se výpočet podobá spíše hádání, ale asi to opravdu jinak nejde.

**Error – locator polynomial** česky asi mnohočlen chyb, není jednotná terminologie

NASA strana 64

Písmenem T označíme počet chyb. T samozřejmě neznáme, budeme ho postupně určovat. Protože nám vyšly některé syndromy nenulové, je zřejmě T > 0

Error-locator polynomial označujeme σ(X)

Dále se v příručce NASA zavádí

Reciprocal error-locator polynomial σr(X) česky asi „inversní mnohočlen chyb“

Z NASA jsem nepochopil, jaký je vztah mezi σ(X) a σr(X) . Ale protože ONI všude používají σr(X) , tak se tím nebudeme trápit.

Stupeň tohoto polynomu je roven počtu chyb.

Takže pro jednu chybu, T=1, je stupeň 1

σr(X) = X1 + σ1

Pro dvě chyby , T=2, je stupeň 2

σr(X) = X2 + σ1X + σ2

Pro tři chyby , T=3, je stupeň 3

σr(X) = X3 + σ1X2 + σ2X+ σ3

Pro čtyři chyby , T=4, je stupeň 4

σr(X) = X4 + σ1X3 + σ2X2+ σ3 X + σ4

a tak dále a tak dále.

Čísla σ1 σ2 σ3 σ4 atd. jsou koeficienty reversního chybového polynomu. Reversní chybový polynom je v normalizovaném tvaru, to znamená, že koeficient u nejvyšší mocniny je 1 .

Na pořadí indexů samozřejmě záleží, mám to doufám dobře, je to odpozorováno z dalšího textu NASA. A výklad na dalších stranách předpokládá právě toto indexování u sigma.

Na spočtení koeficientů chybového polynomu je mnoho metod, například:

Berlekamp – Masey algoritmus

Euclidean Division Algoritm

Linear recursion

Tyto jsou uvedeny v příručce NASA, ale asi jich je ještě více.

Já jsem zvolil Linear recursion , začíná od strany 70 NASA

Pokusíme se tedy určit číslíčka σ1 σ2 σ3 σ4

Uděláme si následující soustavu rovnic (NASA str. 70 nahoře):

=

Vypadá to hrozně, ale není se čeho bát.

Si je syndrom, u nás je to S1 – S6 , i tedy postupuje od 1 do 6 , budeme mít šest rovnic.

Suma sčítá od 0 do T-1 . Sčítací proměnná je j, ta se tedy mění od 0 do T-1 . Sigma je neznámá, kterou máme vypočítat.

Protože nevíme, kolik je T (kolik máme chyb) , tak to prostě zkusíme. Nejdříve dosadíme T=1

Suma se nám tím dost scvrkne, protože sčítáme pro j od 0 do ( 1-1) , takže do 0 Ze sumy tedy zbyde jenom jeden sčítanec.

Pro i=1 dostáváme:

S1 = S1+0-1 . σ1-0  takže to je S1 = S0 . σ1  No ale S0 není, takže tato první rovnice je nesmyslná.

Pro i=2 dostáváme:

S2 = S2+0-1 . σ1-0  takže to je S2 = S1 . σ1

Pro i=3 dostáváme:

S3 = S3+0-1 . σ1-0  takže to je S3 = S2 . σ1

A dále si to vypočtěte sami.

Soustava našich rovnic pro jedno jediné sigma σ1 je tedy:

S2 = S1 . σ1

S3 = S2 . σ1

S4 = S3 . σ1

S5 = S4 . σ1

S6 = S5 . σ1

Za syndromy si dosaďte, a vidíme, že rovnice si odporují. Takže vyřešit sigma σ1  není možné. Proto předpoklad T = 1 neplatí, v přijatém kódovém slově není jedna chyba ( a nějaké chyby tam jsou, takže tam může být 2 ,3, 4, …… 14,15 chyb )

No tak zkusíme T = 2 !

V sumě sčítáme dva členy, T-1 = 2-1 = 1 , takže j nabývá hodnot 0 a 1

Ze sumy dostáváme následující rovnice:

S3 = S1 . σ2  + S2 . σ1

S4 = S2 . σ2  + S3 . σ1

S5 = S3 . σ2  + S4 . σ1

S6 = S4 . σ2  + S5 . σ1

Pokud platí náš předpoklad, že T=2 (dvě chyby), tak tato soustava rovnic musí být řešitelná. To také znamená, že rovnice jsou lineárně závislé. Dále to znamená, že pro nalezení výsledku si můžeme vybrat kteroukoli dvojici rovnic. To, že jsou rovnice řešitelné, se pozná tak, že vypočteme determinant soustavy a ten musí být nenulový. V NASA to tak dělají, vy se zeptejte vyučující(ho) na matematiku nebo ask google.

Protože si můžeme vybrat kteroukoli dvojici rovnic, vybereme si tu, která obsahuje syndrom S5 = 0

( syndromy viz strana 17 )

S6 = S4 . σ2  + S5 . σ1

Po dosazení za syndrom:

α10 = 1. σ2  + 0 . σ1

No a vychází krásných σ2  = α10

Protože teď známe σ2 , dosadíme do kterékoli z uvedených rovnic, např. do

S5 = S3 . σ2  + S4 . σ1

0 = α5 . α10 + 1 . σ1

Proměnná σ1 vychází α15 , a to je samozřejmě 1 . (Pokud to pro vás není samozřejmost, tak zpátky na stromy a zopakovat naše Galoas field na straně 6 )

V NASA výpočet končí na straně 71 , používají se tam determinanty. Samozřejmě, pokud bychom nenašli S5 = 0 , tak nám nezbývá než použít nějakou metodu pro řešení soustav rovnic, například ty determinanty . My jsme se odchýlili od NASA, oni to počítají z prvních dvou rovnic pomocí determinantů.

**!!!!! A tady důrazné upozornění:**

V matematice existuje mnoho metod, jak řešit soustavu rovnic. Například Gaussova eliminační metoda, determinanty atd. atd. Také existuje mnoho programů, které vám soustavu rovnic vyřeší. Tyto programy ale používají „normální algebru“ , tedy tu, ve které je nekonečně mnoho reálných čísel a 1 + 1 = 2 . To není naše algebra ! My máme tolik čísel, kolik vygeneroval „generator polynom“ a + je u nás EX-OR. Takže z Internetu stažený program nebude fungovat. Stejně tak AI . Pochopitelně, RS kódy se dávno používají, takže jsou k disposici programy, které pracují tak, jak my potřebujeme. Ale musíte si je najít ! První blábol, kterým vám odpoví AI, to opravdu nebude.

Výsledné hodnoty koeficientů error-locator polynomu jsou:

σ1 = 1

σ2  = α10

A vlastní polynom má tvar:

σ(X) = X2 + 1.X + α10

Ještě poznámka k výpočtu: NASA počítá z prvních dvou rovnic, my jsme použili 4. rovnici a 5., a vyšlo nám to stejně jako u NASA .

Ale tím jsme ještě neskončili.

Soustava rovnic má pro T = 2 řešení. To znamená, že v kódovém slově jsou dvě chyby .

Ale pořád ještě nevíme, kde jsou ty chyby a jaké jsou .

Kdybychom teď pokračovali pro T=3 , T=4 atd. atd., bude vždy vycházet, že soustava nemá řešení. Je to poměrně jasné: pokud jsou v kódovém slově dvě chyby, tak tam nemohou být tři , nebo čtyři nebo pět …. atd . chyb.

Nabízí se otázka, do jak velkého T máme počítat. U RS kódů RS(n,k) platí n-k = 2\*t , a t je počet chyb, které kód umí opravit ( tohle už v tomto textu bylo mnohokrát). Takže T je polovina z ( n-k)

Náš kód je RS(15,9) 15 - 9 = 6 , takže T jsou 3 .

Pokud dojdeme do hodnoty 3 , a rovnice stále nebudou mít řešení, znamená to, že chyb je více než 3 a nejsme schopni je opravit. Opakujeme, že „chyba“ je špatný symbol, takže u našeho příkladu čtyři bity v symbolu špatně je jedna chyba.

Takže máme inversní error-locator polynomial

σr(X) = X2 + X + α10

A z něj chceme dostat, kde jsou chyby a jak jsou velké.

Jsme na straně 72 NASA

Pro nalezení místa chyb uvádí NASA dvě metody:

1. Chien Search
2. Explicit factorization

Zkusíme **ChienSearch .**

U této metody musíme najít kořeny error-locator polynomial σr(X) . Je to úplně normální matematika, kterou znáte, prostě máme říct, pro které X je σr(X) roven nule.

U NASA na straně 73 řeší hledání kořene polynomu tak, že tupě dosazují všechna čísla z Galoasova pole. Čísla z Galoasova pole mají různá vyjádření, ale nejvhodnější je „power representation“ . Dosazujeme tedy postupně za X α0 , α1 , α2 , α3 , …………… α14 . Proč ne dále ? No protože Galoasovo pole se u 15 cyklí !

Takže tupě dosazujeme za X , a zkoušíme, zda vyjde 0.

Opět upozorňuji: v „normální algebře“ to takhle nejde a jste naučeni z matematiky, že dosazováním nelze kořen vypočítat. Ale ONI mají nekonečně mnoho čísel ! Mu máme jenom několik málo čísel (Galoas field), takže jsme schopni všechna existující čísla dosadit.

Zkuste si to:

Dosadíme za X do σr(X) = X2 + X + α10 hodnotu α3 ( pro ty trochu pomalejší: „dosadit“ znamená, že místo X se do rovnice napíše α3 ) ( a začali jsme „zprostředka“, protože první mocnina je pro ukázku nevhodná )

Vychází:

σr(α3) = α6 + α3  + α10  = α4 ( převést na vector representation, sečíst X-OR, a zpátky )

A vy si dosaďte všechna αi a počítejte a počítejte. Tam, kde vám vyjde po dosazení 0, to je kořen error-locator polynomu . Asi po dvacáté: naučíte se to jenom tak, že si sami budete počítat.

σr(α2) = α4 + α2  + α10  = 0

----- atd

σr(α8) = α16 + α8  + α10  = α1 + α8  + α10 = 0

Dosaďte si tam sami, čučením do skript se nic nenaučíte

Vidíme, že σr vychází nulový pro α2 a α8 .

**Kořeny reversního chybového mnohočlenu jsou tedy α2 a α8**  **.**

**Chyby jsou tedy na pozici 2 a 8** .

Teď se podíváme na stranu 16 , kde máme „Code word s chybami u NASA “

A vidíme, že chyby jsou opravdu na pozicích 2 a 8 !

Zbývá poslední krok, zjistit, jak velké jsou chyby. Ještě jednou opakuji: u CRC jsme měli symbol o délce 1 bit, takže když jsme našli místo chyby , tak jsme prostě bit invertovali. Abeceda měla dva prvky, takže jiná možnost nebyla. U našeho příkladu ale máme symbol o délce 4 bity, takže máme celkem 16 možností. Pokud má například naše pozice 2 hodnotu 1000 , a víme, že je zde chyba, tak jenom víme, že správná hodnota není 1000 , ale něco jiného. Takže musíme zjistit, o kolik jsme se odchýlili od správné hodnoty.

Na straně 74 NASA uvádí metodu pro nalezení hodnoty chyby

**Direct Solution**

Máme následující soustavu rovnic:

=

- Si je nám známý syndrom, máme jich šest a spočítali jsme je na straně 17 .

- T je počet chyb. U nás je T = 2 .

- j je počítátko v sumě. To se mění od 1 do 2 ( do T )

- z nevím jak se to jmenuje, ale budeme za to dosazovat hodnoty kořenů „error-locator polynomial“ z je umocněno na i-tou , i souhlasí s indexem syndromu

- y je „error value“ , hodnota chyby.

Nejdříve si vyjádříme sumu

S1 = y1 (z1)1 + y2 (z2)1

S2 = y1 (z1)2 + y2 (z2)2

S3 = y1 (z1)3 + y2 (z2)3

S4 = y1 (z1)4 + y2 (z2)4

S5 = y1 (z1)5 + y2 (z2)5

S6 = y1 (z1)5 + y2 (z2)5

Protože máme dvě chyby, budeme ze soustavy potřebovat dvě rovnice. Soustava je lineárně závislá, takže si můžeme zvolit kterékoli.

Z rovnic chceme vypočítat y .

Zde se odchýlíme od postupu NASA na staně 75 .

My si vybereme pátou rovnici a dosadíme za syndrom ze strany 17 . !!!! to jsme udělali proto, že syndrom je 0 . Jinak budete muset použít nějakou jinou metodu pro řešení soustav rovnic.

0 = y1 (z1)5 + y2 (z2)5

To samozřejmě znamená, že y1 (z1)5 = y2 (z2)5

Za z dosadíme hodnoty kořenů chybového mnohočlenu ze strany 21 , to je α2 a α8

y1 (α2)5 = y2 (α8)5

y1 α10 = y2 α40  mocninu 40 odcyklíme , cyklí se po 15

y1 α10 = y2 α10

A vidíme, že y1 = y2 !!!!!!

Tedy obě chyby mají stejnou hodnotu.

Potom použijeme jakoukoli další rovnici , třeba první

S1 = y1 (z1)1 + y1 (z2)1  Tady jsme již dosadili y1 = y2

Za z dosadíme hodnoty kořenů chybového polynomu α2 a α8 , za syndrom také dosadíme

1 = y1 α2 + y1 α8

1 = y1 ( α2 + α8)

1 = y1 ( α2 + α8) mociny alfa si sečtěte

1 = y1 . 1

A tedy y1 = y2 = 1

Teď tedy známe hodnotu chyby , u obou chyb je to 1 , ve vector zobrazení je to 0001

A hodnotu chyby prostě přičteme k code word, které máme s chybami .

Code word s chybami u NASA .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 1110 | 0000 | 0101 | 0111 | 0011 | 1000 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | α11 | 0 | α8 | α10 | α4 | α3 | α8 | α12 |
|  |  |  |  |  |  | 0001 |  |  |  |  |  | 0001 |  |  |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1001 | 0101 | 1111 |

**A je to opraveno .**

U příkladu z NASA trochu vadí, že obě chyby vyšly velice jednoduše, takže nevidíme, jak by vypadala oprava na vyšších bitech čtveřice. Zkusíme si na to později udělat vlastní příklad.

**Postup při kódování a dek****ódování RS kódu**

1. Zvolíme „field generator“ . To musí být primitivní polynom.
2. Určíme n a k u RS(n,k) . Přitom je nutno vzít v úvahu stupeň field generatoru.
3. Sestavíme Galoas field, to jsou mocniny kořenů field generatoru. Doplníme 0 .
4. Sestavíme generující polynom pro zabezpečení message word G(X) =
5. 2t = n-k , 2t je počet zabezpečujících symbolů, t maximální počet opravitelných chyb
6. Message word posuneme o 2t symbolů doleva , doplníme 2t nulových symbolů
7. Posunuté message word vydělíme generujícím polynomem a zjistíme zbytek
8. Zbytek přidáme za message word místo 0 . Tím jsme dostali code word.
9. A tohle pošleme na druhou stranu, tohleto je již zabezpečené code word
10. Na druhé straně přijde code word s chybami.
11. Spočítáme syndromy – dosadíme postupně hodnoty Galoas field až do hodnoty 2t
12. Pokud jsou všechny syndromy 0 , přišla zpráva bez chyb
13. Pokud je některý syndrom nenulový, jsou někde ve zprávě nějaké chyby.
14. Spočítáme error-locator polynomial. Ten postupně odhadujeme, od stupně 1 do T .
15. Najdeme kořeny error-locator polynomial metodou Chien Search
16. Kořeny nám určují polohu chyb .
17. Metodou Direct Solution spočteme hodnoty „error value“
18. Error value přičteme na správném místě k přijatému code word , a tím jsme chybu opravili

**Příklad** **NASA podruhé**

U příkladu z NASA trochu vadí, že obě chyby vyšly velice jednoduše. Podle našeho slibu z předchozí kapitoly uděláme chyby trochu komplikovanější.

Opět stejně s minulým příkladem budeme mít

field generator 10011

generující polynom pro výpočet „parity check“

stejně jako minule G(X) = X6+ α10X5+ α14X4+ α4X3+ α6X2+ α9X+ α6

nezabezpečená zpráva je stejně jako minule

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | α11 | 0 |

Zabezpečená zpráva stejně jako minule

Code word bez chyby u NASA .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1001 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | α11 | 0 | α8 | α10 | α4 | α14 | α8 | α12 |

A pozor, teď je změna, chyby jsou jinak , jsou na pozicích 9, 2 a **mají několik bitů chybně** ( u 9 tři bity chybně, u 2 jeden bit)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 1101 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1000 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | α13 | 0 | α11 | 0 | α8 | α10 | α4 | α3 | α8 | α12 |

Takže kódové slovo s chybou je JINAK než u NASA původně

C(x) = α13 \*X9 + α11 \* X7 + α8 \* X5  + α10 \* X4 + α4 \* X3  + α3 \* X2  + α8 \* X1  + α12

Samozřejmě, kódové slovo by mělo začínat u X14, ale tam jsou samé nuly.

Spočítáme syndromy u zprávy s chybami

Na určení syndromů jsem vám udělal počítátko. Je to soubor generpoly.xlsm . Je v něm makro, takže možná budete muset přemlouvat Windows, aby ho spustily.

Na listu koreny zadáme do buněk N7 – R7 field generator 10011 . Ve sloupcích F – I vidíme prvky Galoas field, vektorová representace. Správné zadání field generator je podmínkou správné funkce dalších listů.

Na listu syndromy zadáme do řádku 6 mocniny alfa , které jsou ve zprávě . Pokud potřebujeme zadat nulu, napíšeme !N ( na minus nekonečno) . Zadání čísla 0 do řádku 6 znamená α0 , tedy 1 .

Řádky 17 – 22 transponujeme někam dolů, to znamená přehodíme řádky za sloupce. Zrušíme buňky, ve kterých jsou nesmysly po dosazení !N, tedy čísla 0 . Sloupce mocnin alfa pak vložíme na list odcitatko, počínaje buňkou D16 dolů. Tento list generuje součet mocnin alfa, výsledek je v buňce D32. Používejte „vložit hodnoty“ , jinak vám Excel bude vkládat vzorce z jiných listů a výsledkem bude naprostý chaos.

S1 = α12

S2 = 1 tady vyšlo v generpoly číslo 0 . To znamná α0 , tedy 1

S3 = α7

S4 = α5

S5 = α9

S6 = α2

Syndromy jsou nenulové, takže ve zprávě jsou chyby.

**Error – locator polynomial**

NASA strana 64

=

Jak se s tím počítá jsme již vysvětlili, viz strana 19 . Počítáme s tím, že první příklad NASA jste si prošli a umíte ho, tady dále děláme zkrácenou versi.

Pro T = 1 dostáváme:

S2 = S1 . σ1

S3 = S2 . σ1

S4 = S3 . σ1

S5 = S4 . σ1

S6 = S5 . σ1

Dosazujeme:

1 = α12. σ1

α7 = 1 . σ1

A už vidíme, že si rovnice odporují, takže T = 1 neplatí, je tam více chyb než 1 .

No tak zkusíme T = 2 !

V sumě sčítáme dva členy, T-1 = 2-1 = 1 , takže j nabývá hodnot 0 a 1

Ze sumy dostáváme následující rovnice:

S3 = S1 . σ2  + S2 . σ1

S4 = S2 . σ2  + S3 . σ1

S5 = S3 . σ2  + S4 . σ1

S6 = S4 . σ2  + S5 . σ1

Dosazuji syndromy S1 = α12 S2 = 1 S3 = α7 S4 = α5 S5 = α9 S6 = α2

α7 = α12 . σ2  + 1 . σ1  (SOUSTAVA B1)

α5 = 1 . σ2  + α7 . σ1

α9 = α7 . σ2  + α5 . σ1

α2 = α5 . σ2  + α9 . σ1

Vezmeme první dvě rovnice

α7 = α12 . σ2  + 1 . σ1

α5 = 1 . σ2  + α7 . σ1

α7 = α12 . σ2  + 1 . σ1

α17 = α12 . σ2  + α19 . σ1  druhou jsme vynásobili = α12 ,aby po sečtení zmizelo σ2

součet

a7 + a2 = (1 + a4 ) σ1

a12 = ( a1 ) σ1

σ1  = α11

α5 = 1 . σ2  + α7 . α11 dosadili jsme do druhé rovnice soustavy (SOUSTAVA B1)

α5 = 1 . σ2  + α3

α11 = 1 . σ2

α11 = σ2

Zkouška - dosadíme do soustavy nahoře (SOUSTAVA B1)

α7 = α12 . σ2  + 1 . σ1

α5 = 1 . σ2  + α7 . σ1

α9 = α7 . σ2  + α5 . σ1

α2 = α5 . σ2  + α9 . σ1

α7 = α12 . α11+ 1 . α11

α5 = 1 . α11 + α7 . α11

α9 = α7 . α11 + α5 . α11

α2 = α5 . α11 + α9 . α11

α7 = α23+ α11

α5 = α11 + α18

α9 = α18 + α16

α2 = α16 + α20 Račte si dopočítat sami, velké mocniny odcyklíme, a generpoly.xlsxm list odcitatko

ANO, naše sigma jsou správně

σ1  = α11

σ2  = α11

Takže máme inversní error-locator polynomial ( viz strana 19 )

Pro dvě chyby , T=2, je stupeň 2

σr(X) = X2 + σ1X + σ2

po dosazení našich sigma

σ(X) = X2 + α11.X + α11

A hledáme kořeny tohoto polynomu. Používáme Chien Search

Dosadíme α0

σ(X) = 1 + α11 + α11 =  Sečtěte a vidíme, že to není 0

Dosadíme α1

σ(X) = α2+ α11. α1 + α11 = α2+ α12 + α11  Sečtěte a vidíme, že to není 0

Dosadíme α2

σ(X) = α4+ α11. α2 + α11 = α4+ α13 + α11  Sečtěte a vidíme, že to JE 0 , takže první kořen je α2

Dosadíme α3

σ(X) = α6+ α11. α3 + α11 = α6+ α14 + α11  Sečtěte a vidíme, že to není 0

a dosadíme dál a dál a dál

Dosadíme α9

σ(X) = α18+ α11. α9 + α11 = α3+ α5 + α11  Sečtěte a vidíme, že to JE 0 , takže druhý kořen je α9

Tady již můžeme skončit, protože víme, že chyby jsou dvě.

Kořeny Error locator polynomial jsou tedy α2 a α9

Chyby jsou na pozicích 2 a 9 . Podívejte se na zprávu s chybami a vidíme, že je to opravdu tak .

Dále pokračujeme určením hodnoty chyby.

Použijeme metodu Direct Solution

Máme následující soustavu rovnic:

=

- Si je nám známý syndrom, máme jich šest a spočítali jsme je na straně 27 .

- T je počet chyb. U nás je T = 2 .

- j je počítátko v sumě. To se mění od 1 do 2 ( do T )

- z nevím jak se to jmenuje, ale budeme za to dosazovat hodnoty kořenů „error-locator polynomial“ z je umocněno na i-tou , i souhlasí s indexem syndromu

- y je „error value“ , hodnota chyby.

Nejdříve si vyjádříme sumu

S1 = y1 (z1)1 + y2 (z2)1

S2 = y1 (z1)2 + y2 (z2)2

S3 = y1 (z1)3 + y2 (z2)3

S4 = y1 (z1)4 + y2 (z2)4

S5 = y1 (z1)5 + y2 (z2)5

S6 = y1 (z1)5 + y2 (z2)5

Za z dosazujeme kořeny error locator polynomial, tedy tedy α2 a α9, a za S syndromy ze strany 27

S1 = y1 (α2)1 + y2 (α9)1

S2 = y1 (α2)2 + y2 (α9)2

S3 = y1 (α2)3 + y2 (α9)3

S4 = y1 (α2)4 + y2 (α9)4

S5 = y1 (α2)5 + y2 (α9)5

S6 = y1 (α2)5 + y2 (α9)5

Použili jsme první dvě rovnice, je možno použít kterékoli.

α12 = y1 α2 + y2 α9

1 = y1 α4 + y2 α18

První rovnice krat α2 a sečteme

α14 = y1 α4 + y2 α11

1 = y1 α4 + y2 α18

1 + α14 = y2 (α18 + α11)

α3 = y2 α5

Takže y2  =  α13 1101

A dosadíme do první rovnice

α12 = y1 α2 + α13 α9

α12 = y1 α2 + α22

α12 = y1 α2 + α7

α2 = y1 α2

1 = y1 0001

Error value tedy jsou:

y1  =α0 0001

y2 =  α13 1101

A teď vezmeme zprávu s chybami a error value přičteme

5ýdkz jsou postupně: s chybami , error value , po opravě == bez chyb

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000 | 0000 | 0000 | 0000 | 1101 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1000 | 0101 | 1111 |
| 0 | 0 | 0 | 0 | 0 | α13 | 0 | α11 | 0 | α8 | α10 | α4 | α3 | α8 | α12 |
|  |  |  |  |  | 1101 |  |  |  |  |  |  | 0001 |  |  |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 1110 | 0000 | 0101 | 0111 | 0011 | 1001 | 0101 | 1111 |

**A je to opraveno .**

Vidíme tedy, že lze opravit i chyby ve více bitech, pokud se chybné bity nacházejí v jednom symbolu. U nás jsme měli dva symboly špatně, tedy dvě chyby , a měli jsme 3 a 1 bit chybně v symbolu.

**Kubalíkův příklad 3**

Mějme field generator x4+x3+1: [1,1,0,0,1]

Ten nám vytvoří Galoasovo pole:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mocnina kořenu | |  | m-tuple |  |
| α-∞ | -∞ |  | 0000 | Nula. Opravdu nula |
| α0 | 0 |  | 0001 |  |
| α1 | 1 |  | 0010 |  |
| α2 | 2 |  | 0100 |  |
| α3 | 3 |  | 1000 |  |
| α4 | 4 |  | 1001 |  |
| α5 | 5 |  | 1011 |  |
| α6 | 6 |  | 1111 |  |
| α7 | 7 |  | 0111 |  |
| α8 | 8 |  | 1110 |  |
| α9 | 9 |  | 0101 |  |
| α10 | 10 |  | 1010 |  |
| α11 | 11 |  | 1101 |  |
| α12 | 12 |  | 0011 |  |
| α13 | 13 |  | 0110 |  |
| α14 | 14 |  | 1100 |  |

Uděláme si kód RS(15,9) . To znamená, že máme 6 zabezpečovacích symbolů.

Generující polynom je: G(X) =

G(X) = (X+ α1). (X+ α2). (X+ α3). (X+ α4). (X+ α5). (X+ α6) =

V následujících výpočtech počítáme součin celkem třikrát, abychom se vyhnuli chybám. Výsledek je na straně 33 .

(X+ α1). (X+ α2). (X+ α3). (X+ α4). (X+ α5). (X+ α6) =

= (X2 + Xα2+ Xα1 + α3). (X+ α3). (X+ α4). (X+ α5). (X+ α6) = sčítáme α2+ α1

= (X2 + Xα13 + α3). (X+ α3). (X+ α4). (X+ α5). (X+ α6) =

= (X3 + X2α13 + Xα3 + X2α3 + Xα16 + α6). (X+ α4). (X+ α5). (X+ α6) = sčítáme alfa u stejných mocnin X

= (X3 + X2α8 + Xα10 + α6). (X+ α4). (X+ α5). (X+ α6) =

= (X4 + X3α8 + X2α10 + Xα6 + X3α4+ X2α12 + Xα14 + α10 ) . (X+ α5). (X+ α6) =

= (X4 + X3α7 + X2α4 + Xα12 + α10 ) .(X+ α5) . (X+ α6) =

= (X5 + X4α7 + X3α4 + X2α12 + Xα10 + X4α5+ X3α12 + X2α9 + Xα17 + α15 ) . (X+ α6) =

= (X5 + X4α14 + X3α10 + X2α13 + Xα8 + 1 ) . (X+ α6) =

= (X6 + X5α14 + X4α10 + X3α13 + X2α8 + X + X5α6+ X4α20 + X3α16 + X2α19 + Xα14 + α6 ) =

= (X6 + X5α12 + X4 + X3α2 + X2α7 + Xα11 + α6 ) =

Podruhé, tentokrát postupujeme zezadu:

G(X) = (X+ α1). (X+ α2). (X+ α3). (X+ α4). (X+ α5). (X+ α6) =

= (X+ α1). (X+ α2). (X+ α3). (X+ α4). (X2+ α5 X + α6 X + α11 ) =

= (X+ α1). (X+ α2). (X+ α3). (X+ α4). (X2+ α2X + α11 ) =

= (X+ α1). (X+ α2). (X+ α3). (X3+ α2X2 + Xα11 + X2α4+ α6X + α15 ) =

= (X+ α1). (X+ α2). (X+ α3). (X3+ α11X2 + α1X + 1) =

= (X+ α1). (X+ α2). (X4+ α11X3 + α1X2 + X+ X3α3+ α14X2 + α4X + α3) =

= (X+ α1). (X+ α2). (X4+ α9X3 + α8X2 + α3X + α3) =

= (X+ α1). (X5+ α9X4 + α8X3 + α3X2 + α3X + α2X4+ α11X3 + α10X2 + α5X + α5 ) =

= (X+ α1). (X5+ X4 + α12X3 + α1X2 + α12X + α5 ) =

= X6+ X5 + α12X4 + α1X3 + α12X2 + α5X + X5α1+ X4α1+ α13X3 + α2X2 + α13X + α6 =

= X6 + α12X5 + X4 + α2X3 + α7X2 + α11X + α6 =

A tohle vyblilo AI

G(X)=X6+

X5α6+X5α5+X5α4+X5α3+X5α2+X5α+

X4α11+X4α10+X4α7+X4α4+X4α3+

X3α15+X3α14+X3α12+X3α11+X3α10+X3α9+X3α7+X3α6

+X2α18+X2α17+X2α14+X2α11+X2α10+

Xα20+Xα19+Xα18+Xα17+Xα16+Xα15

+α21

Když to posčítáme, dostaneme

X6+X5α12+ X4.1 +X3α2+ X2α7 +Xα11  + α6

X6 + α12X5 + X4 + α2X3 + α7X2 + α11X + α6 =

**Generující polynom** jsme spočítali třikrát, vychází shodně

G(X) = X6 + α12X5 + X4 + α2X3 + α7X2 + α11X + α6

Zkusíme si zabezpečit nějakou zprávu

1010 1100 0000 0100 1001 0110 0010 1111 0011

Zprávu jsem si vymyslel. Mezery mezi čtveřicemi bitů jsou jenom proto, aby se zpráva dala snáze číst. Zpráva má 9 symbolů a 36 bitů . Zprávu přepíšeme pomocí „power representation“ , viz dále

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 1100 | 0000 | 0100 | 1001 | 0110 | 0010 | 1111 | 0011 |
| α10 | α14 | 0 | α2 | α4 | α13 | α1 | α6 | α12 |

Jak jsem už mnohokrát řekl, čučením do skript se nic nenaučíte. Takže se hbitě podívejte na stranu 32, najděte si čtveřici bitů v našem Galoas field a podle toho doplňte mocninu kořene alfa na … . Připomínám, že nule je doopravdy nula, takže čtveřice bitů 0000 se zobrazí jako 0 .

Dále si připomeneme, že posun o jistý počet pozic doleva znamená vynásobení xk .

(x4 + x3 +1) \* x2 = x6 + x5 + x2 , takže z 11001 dostaneme 1100100

První čtveřice bitů 1010 je o osm míst („místo“ zde je jeden symbol, takže 4 bity ) posunuta doleva oproti poslední čtveřici. Znamená to, že ji musíme vynásobit číslem x8 . Poslední čtveřice není posunuta, to znamená vynásobit 1 , tedy x0 .

Takže naše zpráva je

α10 \*x8 + α14 \*x7 + 0\*x6  + α2 \*x5 + α4 \* x4  + α13 \* x3  + α1 \* x2  + α6 \* x1  + α12 \* x0

Samozřejmě, 0\* x6 je 0 a α12 \* x0 je α12 .

No a teď to budeme dělit generujícím polynomem. Stejně jako u CRC tu celou zprávu posuneme doleva, v našem případě o tolik symbolů, kolik činí zabezpečovací část code word. To je u nás o 6 symbolů. To prostě znamená, že naši zprávu vynásobíme x6 .

(α10 \* x8 + α14 \* x7 + α2 \* x5 + α4 \* x4  + α13 \* x3  + α1 \* x2  + α6 \* x1  + α12) \* x6 =

α10 \* x14 + α14 \*x13 + α2 \*x11 + α4\*x10  + α13\*x9  + α1\*x8  + α6\*x7  + α12\*x6

A tohleto celé teď vydělíme generujícím polynomem ( to NENI “field generator”, generující polynom pro zabezpečení u RS máme na straně 26 ), zjistíme zbytek a ten přidáme za původní zprávu.

(α10 \*x14 + α14 \*x13 + α2\*x11 + α4\*x10 + α13\*x9 + α1\*x8 + α6\*x7 + α12\*x6 ) : ( X6+α12X5+X4+α2X3+ α7X2+ α11X + α6 )

V souboru generpoly.xlsm máte na listu delime delitko pro výpočet zbytků po dělení. Do řádku 6 zadáte dělenec a dělitel, zadáváme exponenty u alfa. Takže pokud chceme zadat číslo 1 , je to alfa na nultou, zadáme 0. Pokud chceme zadat číslo 0 – nula, tak to je alfa na minus nekonečno, to je v souboru označeno jako !N . Pak je nutno spustit makro - Cntrl a (a je malé a) . Dole na řádce 44 pak zjistíte zbytek.

U výše uvedené zprávy je zbytek

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X5 | X4 | X3 | X2 | X1 | 1 |
| 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 9 | 9 | 2 | 5 | 14 |

úplně celá zabezpečená zpráva je



Zabezpečenou zprávu si přepíšeme jako mnohočlen

M(X)= α10 \*X14 + α14 \*X13 + α2\*X11 + α4\*X10 + α13\*X9 + α1\*X8 + α6\*X7 + α12\*X6 +α1X5+ α9X4+α9X3+ α2X2+ α5X + α14

A ještě jednou pro jasné pochopení pomocí bitů, tedy „vector representation“

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 1100 | 0000 | 0100 | 1001 | 0110 | 0010 | 1111 | 0011 | 0010 | 0101 | 0101 | 0100 | 1011 | 1100 |
| α10 | α14 | 0 | α2 | α4 | α13 | α1 | α6 | α12 | α1 | α9 | α9 | α2 | α5 | α14 |

Teď doporučuji spočítat syndromy. Můžete k tomu použít soubor generpoly.xlsm, list syndromy a odcitatko. Všechny syndromy jsou 0 , protože zpráva je bez chyb.

Ve zprávě uděláme chybu. Nebudeme se vodbejvat a uděláme tři chyby.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 0011 | 0000 | 0100 | 1001 | 1111 | 0010 | 1111 | 0011 | 0010 | 0101 | 1101 | 0100 | 1011 | 1100 |
| α10 | α12 | 0 | α2 | α4 | α6 | α1 | α6 | α12 | α1 | α9 | α11 | α2 | α5 | α14 |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

U symbolu 13 jsou čtyři bity špatně, u symbolu 9 dva bity špatně, u symbolu 3 jeden bit špatně. Ve zprávě máme tři chyby, protože tři symboly jsou špatně.

Takže naše zpráva s chybou vypadá takto:

Mc(X)= α10 \*X14 + α12 \*X13 + α2\*X11 + α4\*X10 + α6\*X9 + α1\*X8 + α6\*X7 + α12\*X6 +α1X5+ α9X4+α11X3+ α2X2+ α5X + α14

Zprávu budeme opravovat.

Spočítáme syndromy S1 – S6 – to znamená, že dosadíme za X do Mc(X) hodnoty od α1 do α6

Výpočet můžete sledovat v souboru generpoly.xlsxm , list syndromy . Vidíme mocniny alfa. Na konci jsou zaměněny řádky za sloupce, abychom mohli snadno použít list odcitatko , D16 – D29

S1 = 0

S2 = α13

S3 = 0

S4 = α11

S5 = α14

S6 = α14

Suma pro výpočet Linear recursion je na straně 18 .

=

Pro počet chyb T = 1 a T = 2 dostáváme soustavy rovnic, které nejsou řešitelné. To ponechám vašemu laskavému samostudiu.

Pro T=3 přejde suma do soustavy rovnic (pro i = 1,2,3 se v soustavě vyskytují neexistující syndromy, proto tyto případy neuvádíme):

=

S4 = S1 . σ3  + S2 . σ2  + S3 . σ1

S5 = S2 . σ3  + S3 . σ2  + S4 . σ1

S6 = S3 . σ3  + S4 . σ2  + S5 . σ1

Z této soustavy vypočítáme koeficienty sigma .

Za syndromy dosadíme:

α11 = 0 . σ3  + α13. σ2  + 0 . σ1  (X1)

α14 = α13 . σ3  + 0 . σ2  + α11 . σ1 (X2)

α14 = 0 . σ3  + α11 . σ2  + α14. σ1 (X1)

A rovnice upravíme

α11 = α13. σ2  (X1)

α14 = α13 . σ3  + α11 . σ1 (X2)

α14 = α11 . σ2  + α14. σ1 (X3)

připomínám, že alfa je konstanta, známá, sigma je neznámá, kterou máme vyřešit.

Rovnice vyšly velice hezky. To tak nemusí být pokaždé. Standardně budeme muset použít nějakou metodu pro řešení soustav rovnic, například determinanty. A než začnete hledat nějaký program, přečtěte si ještě jednou upozornění na straně 21 .

Vezmeme (X1) , vydělíme α13

σ2 = α11-13 = α-2 = α13

Dosadíme do (X3)

α14 = α11 . α13 + α14. σ1

α14 = α24 + α14. σ1

α14 + α9 = α14. σ1

α4 = α14. σ1

α-10 = σ1

α5 = σ1

A dosadíme do (X2)

α14 = α13 . σ3  + α11 . α5

α14 = α13 . σ3  + α16

α14 = α13 . σ3  + α1

α8 = α13 . σ3

α-5 = σ3

α10 = σ3

Celkem vychází

α5 = σ1

α13 = σ2

α10 = σ3

Uděláme zkoušku:

S1 = 0  S2 = α13  S3 = 0 S4 = α11  S5 = α14  S6 = α14

S4 = S1 . σ3  + S2 . σ2  + S3 . σ1

S5 = S2 . σ3  + S3 . σ2  + S4 . σ1

S6 = S3 . σ3  + S4 . σ2  + S5 . σ1

S4 = 0 . σ3  + α13 . σ2  + 0 . σ1

S5 = α13 . σ3  + 0 . σ2  + α11 . σ1

S6 = 0 . σ3  + α11 . σ2  + α14 . σ1

S4 =  α13 . α13

S5 = α13 . α10  + α11 . α5

S6 = α11 . α13 + α14 . α5

S4 =  α26 = α11

S5 = α23 + α16 = α8 + α1  = α14

S6 = α24 + α19 = α9 + α4 = α14

Zkouška nám vyšla.

Takže naše koeficienty sigma jsou správně.

σ1 = α5

σ2 = α13

σ3  = α10

A podle strany 18 sestavíme Error – locator polynomial

σr(X) = X3 + σ1X2 + σ2X+ σ3

σr(X) = X3 + α5 X2 + α13 X + α10

A dále hledáme kořeny σr(X) . Použijeme Chien Search , to je tupé dosazování a hádání kořene.

Dosadíme α0 , to je 1

σr(X) = X3 + α5 X2 + α13 X + α10

σr(X) = 1 + α5 + α13  + α10 = α10

nevyšla 0 , takže α0 není kořen.

Pro sčítání můžete použít soubor generpoly.xlsm , list odcitatko

Dosadíme α1

σr(X) = X3 + α5 X2 + α13 X + α10

σr(X) = α3+ α5 α2+ α13 α1+ α10 = α3 + α9 + α14 + α10 = α5 , nevyšla 0, α1 není kořen

Dosadíme α2

σr(X) = α6+ α5 α4+ α13 α2+ α10 = α6 + α9 + α15 + α10 = α1 , to není ono α2 není kořen

Dosadíme α3

σr(X) = X3 + α5 X2 + α13 X + α10

σr(X) = α9 + α5 α6 + α13 α3 + α10 = α9 + α11 + α16 + α10 = α9 + α11 + α1 + α11 = 0 , takže α3 je kořen

Dosadíme α4

σr(X) = α12 + α5 α8 + α13 α4 + α10 = α12 + α13 + α2 + α10 = α5 , to není ono , α4 není kořen

a dosazujeme dál a dál a dál …………….

Dosadíme α9

σr(X) = X3 + α5 X2 + α13 X + α10

σr(X) = α27 + α5 α18 + α13 α9 + α10 = α12 + α8 + α7 + α10 = 0 , to JE ONO , α9 je kořen

a dosazujeme dál a dál a dál …………….

Dosadíme α13

σr(X) = X3 + α5 X2 + α13 X + α10

σr(X) = α39 + α5 α26 + α13 α13 + α10 = α9 + α1 + α11 + α10 = 0 , to JE ONO , α13 je kořen

Zjistili jsme tedy tři kořeny reversního error-locator polynomu:

α3  α9 α13

Mocniny kořenů udávají polohu chyb. Vidíme tedy, že chyby jsou na místech, které odpovídají mocninám X: 3 9 13

Podíváme se na naši zprávu s chybami a zajásáme, protože nám místa chyb souhlasí.

To ale není všechno.

Chyby potřebujeme opravit.

Použijeme metodu Direct Solution, viz strana 23 .

=

- Si je nám známý syndrom, máme jich šest a spočítali jsme je na straně 17 .

- T je počet chyb. U nás je T = 3 .

- j je počítátko v sumě. To se mění od 1 do 3 ( do T )

- z nevím jak se to jmenuje, ale budeme za to dosazovat hodnoty kořenů „error-locator polynomial“ z je umocněno na i-tou , i souhlasí s indexem syndromu

- y je „error value“ , hodnota chyby.

Nejdříve si vyjádříme sumu

S1 = y1 (z1)1 + y2 (z2)1 + y3 (z3)1 (R1)

S2 = y1 (z1)2 + y2 (z2)2 + y3 (z3)2 (R2)

S3 = y1 (z1)3 + y2 (z2)3 + y3 (z3)3 (R3)

S4 = y1 (z1)4 + y2 (z2)4 + y3 (z3)4 (R4)

S5 = y1 (z1)5 + y2 (z2)5 + y3 (z3)5 (R5)

S6 = y1 (z1)5 + y2 (z2)5 + y3 (z3)6 (R6)

Protože máme tři chyby, budeme ze soustavy potřebovat tři rovnice. Soustava je lineárně závislá, takže si můžeme zvolit kterékoli.

Z rovnic chceme vypočítat y .

Zvolíme první tři rovnice:

S1 = 0  S2 = α13  S3 = 0 S4 = α11  S5 = α14  S6 = α14

A za z dosadíme hodnoty kořenů chybového polynomu α3  α9 α13

0 = y1 (α3)1 + y2 (α9)1 + y3 (α13)1 (R1-2)

α13 = y1 (α3)2 + y2 (α9)2 + y3 (α13)2 (R2-2)

0 = y1 (α3)3 + y2 (α9)3 + y3 (α13)3 (R3-2)

0 = y1 α3 + y2 α9 + y3 α13 (R1-3)

α13 = y1 α6 + y2 α18 + y3 α26 (R2-3)

0 = y1 α9 + y2 α27 + y3 α39 (R3-3)

0 = y1 α3 + y2 α9 + y3 α13 (R1-4)

α13 = y1 α6 + y2 α3 + y3 α11 (R2-4)

0 = y1 α9 + y2 α12 + y3 α9 (R3-4)

Dále zredukujeme počet proměnných. Použijeme R1-4 a R2-4 , druhou vynásobíme α2 a poté sečteme.

0 = y1 α3 + y2 α9 + y3 α13 (R1-4)

α15 = y1 α8 + y2 α5 + y3 α13 (R2-4)

α15 = y1 (α8 + α3) + y2 (α5 + α9 ) R1-4 + R2-4 (R7)

α15 = y1 α13 + y2 α8  (R7)

Obdobně použijeme R1-4 a R3-4 , druhou vynásobíme α4 a poté sečteme.

0 = y1 α3 + y2 α9 + y3 α13 (R1-4)

0 = y1 α13 + y2 α16 + y3 α13 (R3-4)

0 = y1 (α3 + α13) + y2 (α9 + α16 ) R1-4 + R3-4 (R8)

0 = y1 α8 + y2 α7 (R8)

Dále vezmeme R7 a R8 a stejným trikem odstraníme proměnnou y2

α15 = y1 α13 + y2 α8

0 = y1 α8 + y2 α7 (R9)

α15 = y1α13 + y2 α8

0 = y1α9 + y2 α8

α15 = y1 (α13 + α9 )

α15 = y1 α12

α3 = y1

A následuje zpětný chod. Dosadíme do (R9)

0 = y1 α8 + y2 α7

0 = α3 α8 + y2 α7

S α11 = y2 α7

α4 = y2

A pak dosadíme například do (R1-3)

0 = y1 α3 + y2 α9 + y3 α13

0 = α3 α3 + α4 α9 + y3 α13

0 = α6 + α13 + y3 α13

y3 α13 = α4

y3 = α-9

y3 = α6

Blížíme se k finále. Možná se vám zdá, že uvedený postup je příliš komplikovaný. Jedná se ale o úplně normální řešení soustav rovnic, a na to máme v matematice mnoho osvědčených a vyzkoušených postupů. Jenom to trvá dlouho, ale není to nic komplikovaného. Stejně tak existuje mnoho programů, které soustavu rovnic vyřeší. Opět ovšem upozorňuji na to, že počítáme v algebře, která má jenom několik čísel ( 16 v našem případě) podle toho, jak jsme si udělali Galoas field . ( text upozornění strana 21 , Galoas filed strana 32 )

Zbývá opravit chyby.

To uděláme tak, že ke zprávě s chybami přičteme právě spočítané error value y1 y2 y3

y1 = α3 1000

y2  = α4 1001

y3 = α6 1111

Zpráva s chybami + error value ve třetím řádku

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 0011 | 0000 | 0100 | 1001 | 1111 | 0010 | 1111 | 0011 | 0010 | 0101 | 1101 | 0100 | 1011 | 1100 |
| α10 | α12 | 0 | α2 | α4 | α6 | α1 | α6 | α12 | α1 | α9 | α11 | α2 | α5 | α14 |
|  | 1111 |  |  |  | 1001 |  |  |  |  |  | 1000 |  |  |  |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Zpráva bez chyb

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 1100 | 0000 | 0100 | 1001 | 0110 | 0010 | 1111 | 0011 | 0010 | 0101 | 0101 | 0100 | 1011 | 1100 |
| α10 | α14 | 0 | α2 | α4 | α13 | α1 | α6 | α12 | α1 | α9 | α9 | α2 | α5 | α14 |

Pokud milujete power representation, tak uděláte prostě

α11 + α3 = α9

α6 + α4 = α13

α12 + α6 = α14

**A je to opraveno .**

Vidíme, že podle postupu v příručce NASA lze opravit i jiný počet chyb, můžeme použít jiný field generator než NASA a můžeme mít i výrazně komplikovanější chyby .

**Implementace RS kódů u GPON**

Popis implementace RS kódů máte ve file implementace.pdf . Po přečtení naší příručky (== to je ta, kterou pávě čtete ) by vám měl být tento soubor bez problémů srozumitelný. Zejména se naučte field generator pro GPON, na to se budeme ptát u maturity. Dále jsem vám udělal ve file generpoly.xlsm list GFproGPON , ve kterém krásně vidíte, jak field generator vypadá. Je tam i „decimal representation“ odvozená z „vector representation“ .

Dále máte ve file T-REC-G.709-202006-I!!PDF-E-1.pdf na straně 180 Annex A, kde je popsán FEC pro GPON. Přečtěte si !

AI tvrdí, že pro VDSL je tentýž polynom definován v ITU-T G.993.2 . Ale jim se nesmí moc věřit.

Ale našel jsem to, doporučení máte na <http://ozeas.sdb.cz/panska/4A/xDSL/T-REC-G.993.2-201902-Teil_1.pdf>

Kapitola 9.3 , opět si račte přečíst.