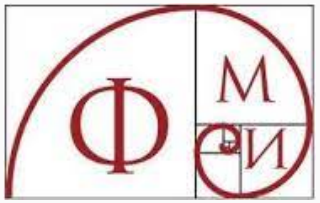
Кодиране

Кристиян Стоименов

15 февруари 2026 г.



Факултет по математика и информатика,

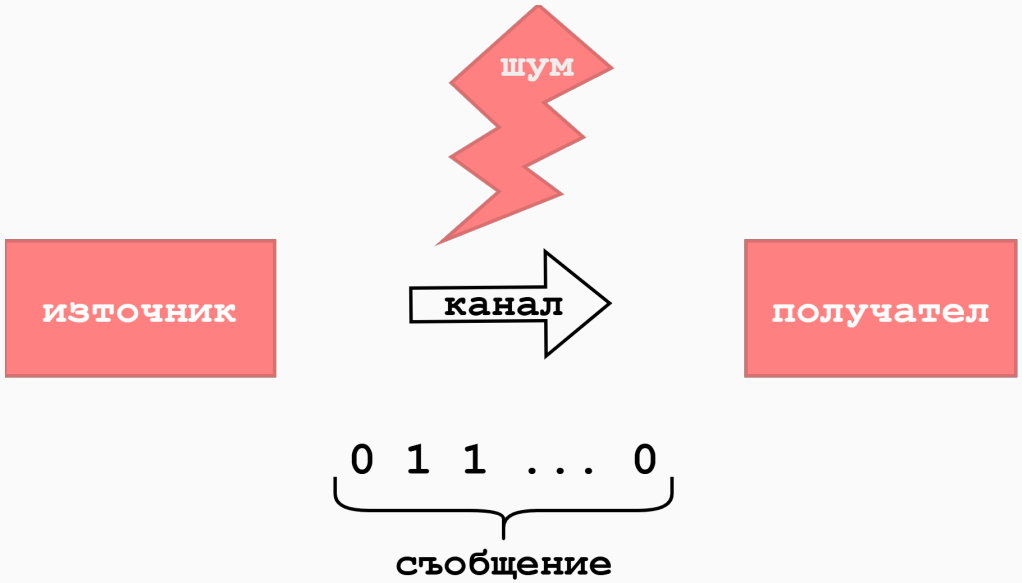
Ефективна реализация
на математически алгоритми
и концепции

Дефиниция

Нека имаме БЧХ код $\mathcal{C}$ над F_q с дължина $q-1$ и конструктивно разстояние δ. Тогава за $\mathcal{C}$ казваме, че е код на Рид-Соломон.

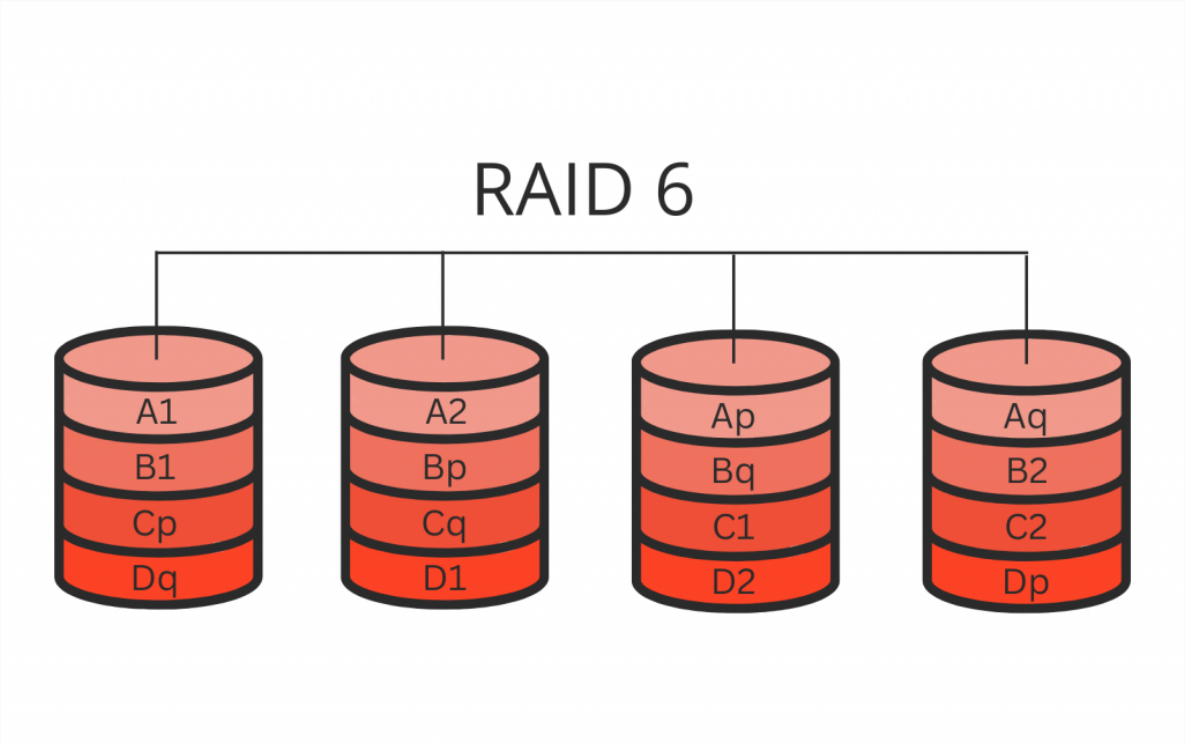
Основни понятия

Шумозащитно кодиране наричаме набор от математически средства и алгоритми, чрез които грешки, възникнали при предаване на информация по канал между два агента, биват разпознати и поправени.



Фигура 1: Опростен модел на предаване на информация през комуникационен канал.

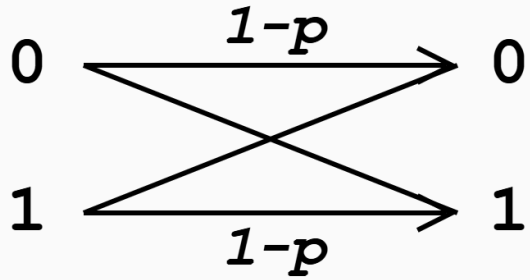
Фигура 2: Voyager 2 е безпилотна космическа сонда на НАСА, изстреляна през 1977 г., която изследва външните планети на Слънчевата система. Единственият апарат, посетил Уран и Нептун. Снимката е взема от [NAS26].



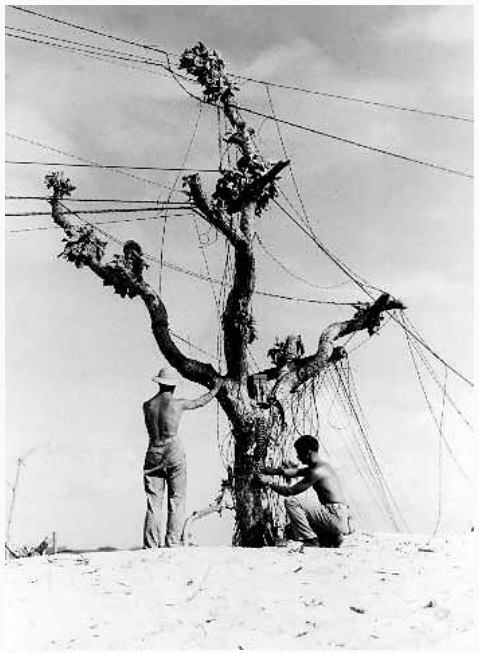
Фигура 3: RAID-6 е схема за нареждане на масив от дискове с цел максимално бързодействие и едновременно с това издръжливост при грешка. [Ana20]

- Приложенията на шумозащитното кодиране са много. Някои други от тях включват CD [99] и QR кодовете [**qr**].
- Основния фактор, допускащ ефективността на шумозащитното кодиране е следната теоремата за шумния канал (Шанън, 1948) [Mac03]. В общо линии тя гласи, че за произволен шумен канал, съобщения могат да се предават без загуба на информация, стига преносът да не надхвърля *капацитета* - максималната пропускателна способност, на канала.

Какво представлява шумът?



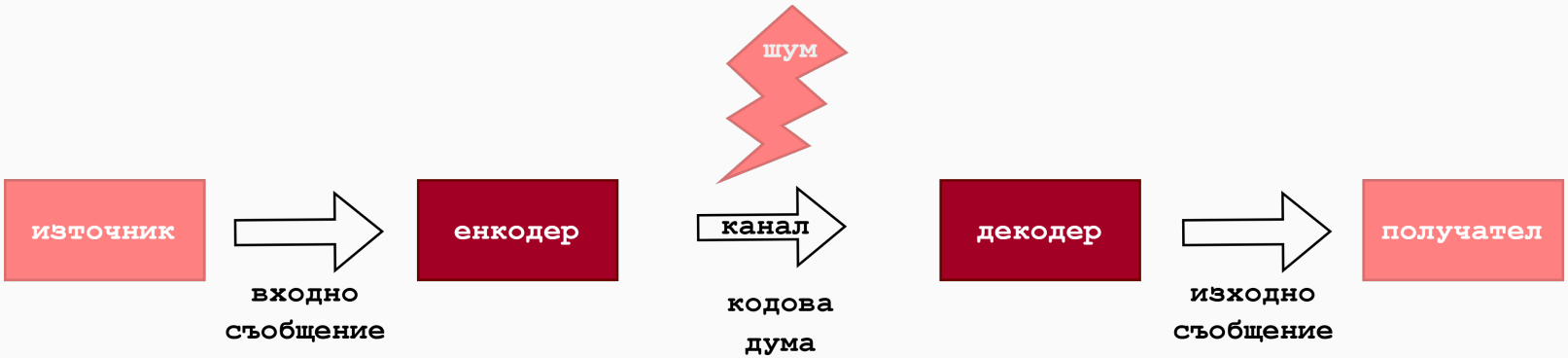
p := P(обръщане)



Фигура 4: Двоичен симетричен канал.

Шумозащитен код, или за по-кратко само *код*, наричаме систематичен подход за съпоставяне между съобщение и кодова дума. Това съпоставяне се разглежда в две посоки:

- Посоката „съобщение $\mapsto$ кодова дума" се нарича *кодиране*. *Кодирането изменя съобщението, добавяйки проверочни символи* [1]*, такива че входното съобщение да бъде възстановено в случай на промяна по време на предаване.*
- Обратната, „кодова дума $\mapsto$ съобщение", се нарича *декодиране* и именно тя е същината на един шумозащитен код. *Декодирането използва въведените от кодирането проверочни символи, за да поправи възникнали грешки от предаването,* ***ако свойствата на шумозащитния код позволяват това****.*



Фигура 5: Опростен модел на предаване на информация през комуникационен канал, използвайки шумозащитен код.

[1] на англ. redundancy

Код с повторение

Нека разгледаме един елементарен пример за шумозащитен код, за да илюстрираме по-ясно модела на предаване на информация през комуникационен канал.

Следва да фиксираме няколко основни параметъра на нашия код - дължина на съобщението и брой проверочни символа. От тях получаваме директно и максималния брой грешки, които шумозащитния код е способен да възстанови.

Нека за нашите цели съобщението се състои от 16 бита, а броят на проверочните символи е 32 бита. Така кодовата дума става за дължина 48 бита.

Използваме следния алгоритъм за кодиране. Ефектът от него е, че всеки бит от съобщението се предава точно по три пъти, т.е. повтаряме съобщението три пъти. Оттам и името на този код.

```
1  u64 encode(u16 msg)
2  {
3    u64 codeword=msg|msg<<16|msg<<32;
4    return codeword;
5  }
```

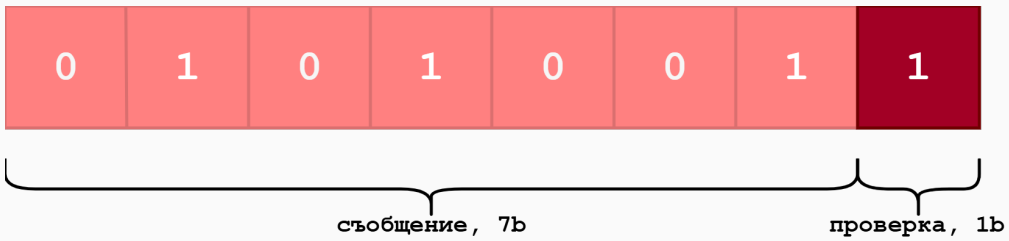
Декодирането също става очевидно - за всеки от битовете в получената кодова дума избираме стойност за съобщението, съответстваща на модата от стойностите на трите копия.

```
1  u16 decode(u64 codeword)
2  {
3    u16 msg=0;
4    for (int i=0; i<16; ++i
5    {
6      int l1=1<<i, l2=1<<(16+i), l3=1<<(32+i);
7      int b1=!!(codeword&l1), b2=!!(codeword&l2),
8        b3=!!(codeword&l3);
9      msg |= b1+b2+b3>>1<<i;
10   }
11   return msg;
12 }
```

Разграничаваме *откриване* от *поправяне* на грешка. В първия случай имаме единствено информация, че полученото съобщение се различава от изпратеното. Във втория случай знаем не само това, ами и кои точно битове са били промени.

Друг вариант за грешка е обаче някои от битовете да бъдат *пропуснати*, а не обърнати. Също така възникват и други особености при бройни системи, които не са двоичната - в такива случаи не е достатъчно да знаем къде е възникнала грешката, тъй като има повече от една възможна друга стойност.

Прост пример за това е т.нар. *проверка по четност*. Нейната идея е следната. Ако имаме съобщение, съставено от например 7 двоични символа, то проверка по четност реализираме, добавяйки символ с такава стойност, че броя на всички единици сред съобщението да бъде четен. Очевидно в случая можем да забележим единствено нечетен брой грешки, а да поправим, не можем да нито една от тях.



Фигура 6: Прост пример за проверка по четност.

Код на Хеминг

Преди да се насочим към общата теория на линейните блокови кодове, нека разгледаме как е функционирал първият шумозащитен код - малко по-нетривиален пример от кода с повторение. Той е известен като код на Хеминг, който е негов изобретател през 40-те години в Bell Labs.

Основната идея, на която се основава той е следната: ако поставим *достатъчно* на брой проверки по четност и при това те да бъдат на *точните* места, получаваме удобен и лесен начин за откриване на *единствена грешка*.

Ще разгледаме как се случва това при съобщения от 11 бита, които се поставят в 15-битови кодови думи. Подходът се запазва аналогично при блок с произволна дължина 2^m.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

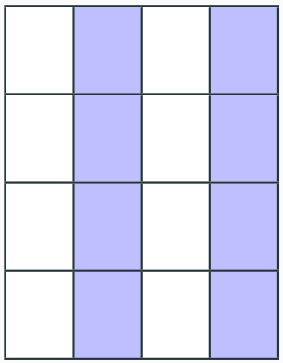
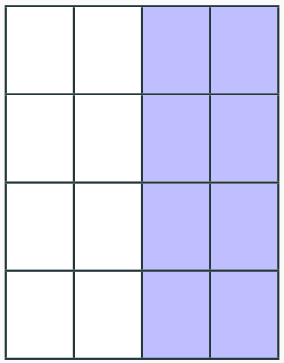
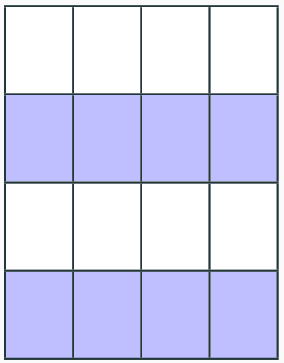
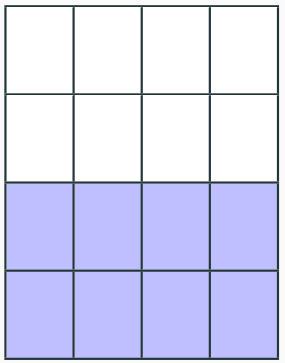
Използваме този нагласен „формат”, попълвайки съобщението, което искаме да изпратим (11 бита) в зелените клетки. В жълтите ще поставим проверочните битовете, а оражевото временно ще игнорираме.

Стойностите, сега намиращи се в клетките на шаблона, отговарят на „индекса”, който ще използваме за съответната клетка (т.е. аналогично можем да си представим и последователно разположение, вместо таблично).

Нека попълним клетките със съобщението 10011001101:

			1
	0	0	1
	1	0	0
1	1	0	1

Жълтите клетки - проверките по четност - попълваме със стойностите, съответстващи на следните четири подгрупи от клетки:

Можем да забележим, че първите две подгрупи успяват да намерят колоната, в която е възникнала грешка, докато вторите две - редичката.

Нека попълним и жълтите клетки с проверките по четност за примерното съобщение:

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

Както може би забелязвате, интересното за позициите на *жълтите клетки* (и причината за именно този избор), е това, че всеки проверочен бит попада в точно една от подгрупите.

Друго важно наблюдение ни показва по какъв начин избрахме точно това да са подгрупите - ако разгледаме отново шаблона за попълване с въведени индексите на всяка от клетките ще забележим, че например първата подгрупа включва тези и само тези клетки, чиито индески завършват на 1.

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1011	1110	1111

Аналогично следващите подгрупи ни дават тези, включващи `..1.`, `.1..` и `1...`.

Така до момента можем да попълним всички без един от битовете в онзи 16-битов пакет:

	1	1	1
0	0	0	1
0	1	0	0
1	1	0	1

Нека го игнорираме още малко и да разгледаме как точно поправяме възникнала грешка.

Бележка: До момента можем да кажем, че сме се запознали с (почти цялата) процедура по кодиране и следва да декодираме.

Допускаме, че бит с индекс 6 е бил обърнат.

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

Извършваме проверките по четност, използвайки подгрупите, определящи стойностите на проверочните битове една по една и записваме резултата от всеки от тях.

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

0

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

1

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

1

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

0

Магия! Получихме именно 6.

Основната причина този декодиращ алгоритъм да работи (и при това така елегантно) се дължи на организацията на „подгрупите” от битове, чрез които ефективно извършваме двоично търсене - най-десният бит грешен ли е? а този вляво? а този вляво? . . . а най-левият бит грешен ли е?

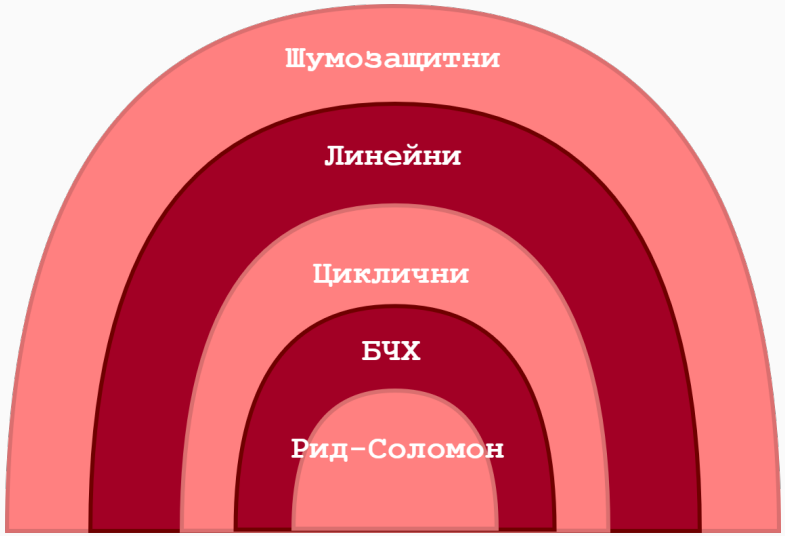
Последното парче от пъзела е бит с индекс 0 - каква функция изпълнява той? Всъщност той не е полезен за алгоритъма, който разгледахме до момента. Тъй като обаче обичайно 15-битов блок от данни е твърде необичайно (не е степен на 2), последният бит се използва за проверка по четност на останалите 15 бита. По този начин получаваме механизъм за поправяне на единствена грешка, но улавяне на две възникнали. Тази добавка на още един бит за четност обичайно се нарича разширен код на Хеминг.

Възникват няколко естествени въпроса.

- Има ли подход, по който да кодираме информация така, че да можем да възстановим дори повече от една грешка?
- Има ли как да постигнем това придържайки се към прости алгоритми за кодиране и декодиране? Целта ти е тези операции да бъдат *бързи*.

Настоящият проект не претендира да отговаря изчерпателно на тези два въпроса, а само да приближи изготвилия труда, както и слушателите на това представяне, към *истината* ☺.

Видове кодове



Фигура 7: „Йерархия” на видовете кодове.

Линейни кодове

Нека разгледаме какво точно представлява един шумозащитен код.

Оттук насетне считаме, че източник съставя съобщения използвайки *символи* от крайно поле $\mathbb{F}$ с мощност q. Тъй като полетата, които ни интересуват са крайни, обичайно ги бележим с $GF(q)$.

Дефиниция

(n, k) блоков код $\mathcal{C}$ над азбука от q символа наричаме множество от q^k вектора, всеки от които има дължина n. Тези вектори наричаме кодови думи.

На всеки блоков код съответства енкодер, който съпоставя на съобщението $m \in \mathbb{F}^k$ съответна кодова дума $c \in \mathcal{C}$.

Често кодовете, които ни интересуват имат следното свойство.

Дефиниция

Нека блоковия (n, k) код $\mathcal{C}$ над $GF(q)$ образува линейно подпространство на линейното пространство от наредените n-торки $\mathbb{F}_q^n$. Тогава за $\mathcal{C}$ казваме, че е линеен блоков код.

Използвайки аксиомите на линейно пространство, получаваме начин за „конструиране" на нови кодови думи. Например, ако $c_1, c_2 \in \mathcal{C}$, то $c_1 + c_2 \in \mathcal{C}$.

Третата основна характеристика на блоковите кодове освен дължината на съобщението и дължината на кодовата дума е неговото минимално разстояние. Именно то определя максималният брой грешки, които кодът може да поправи. За целта ни е необходима следната метрика.

Дефиниция

Разстояние на Хеминг между две кодови думи $a, b \in \mathcal{C}$, *където* $\mathcal{C}$ *е* (n, k) *код, наричаме*

$$d(a, b) := |\{ i \mid a_i \neq b_i \}|$$

Дефиниция

Минимално разстояние на $\mathcal{C}$ наричаме

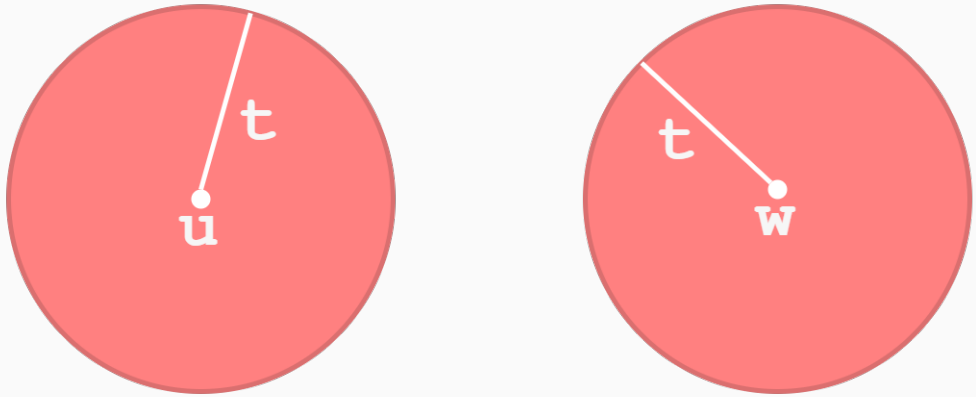
$$d(\mathcal{C}) := \min\{\, d(a,b) \mid a,b \in \mathcal{C},\ a \neq b \,\}$$

Стойността

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

наричаме радиус на опаковка и съвпада с максималния брой грешки, които могат да бъдат поправени от кода $\mathcal{C}$.

(n,k) код $\mathcal{C}$, при който $d := d(\mathcal{C})$, бележим като (n,k,d) код.



Фигура 8: Абстрактно представяне на кодови думи заедно с радиус на опаковка.

Оттук насетне ще разглеждаме линейни блокови кодове. Нека $\mathcal{C}$ е (n, k, d) линеен код. От линейната алгебра ни е известно понятието *базис* и знаем също така, че произволен елемент $c \in \mathcal{C}$ може да се представи (по единствен начин) като *линейна комбинация* на базисните вектори. Нека $g_0, \cdots, g_{k-1}$ образува базис на $\mathcal{C}$. Тогава за произволна кодова дума $c \in \mathcal{C}$ е вярно, че

$$c = \sum_{i=0}^{k-1} m_i g_i$$

Или, записвайки вектори g_i като редове на една матрица G

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix}$$

и съобщението като вектор-ред

$$m = [m_0 \cdots m_{k-1}]$$

кодовата дума, съответстваща на съобщението m получаваме като

$$c = mG$$

Матрицата G се нарича *пораждаща* за кода $\mathcal{C}$. Чрез нея директно получаваме алгоритъм за кодиране - а именно чрез умножение.

Нека за илюстрация разгледаме един от първите използвани шумозащитни кодове - $(7, 4)$ кодът на Хеминг. Чрез него 4-битови съобщения се кодират в 7-битови кодови думи, използвайки 3 проверочни бита. Една негова пораждаща матрица е

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Тогава съобщението $m = [1001]$ кодираме по следния начин

$$c = mG = [1100101]$$

Използвайки метриката *разстояние по Хеминг* схемата на декодиране се нарича *метод на максималното правдоподобие*. Имаме следната

Теорема

За двоичен симетричен канал с вероятност за грешка $p < \frac{1}{2}$ декодирането по метода на максималното правдоподобие е еквивалентно на декодиране до най-близката кодова дума.

Сега ще разгледаме как работи това декодиране чрез таблица на Слепян.

Преди да посочим алгоритъма, ни необходима е следната

Дефиниция

Нека $\mathcal{C}$ е двоичен (n, k) код. Съседен клас на кода $\mathcal{C}$, определен от вектора y наричаме множеството

$$y + \mathcal{C} := \{ y + c \mid c \in \mathcal{C} \}$$

Лидер на съседен клас $y + \mathcal{C}$ наричаме елемента

$$\hat{c} := \min_{c \in y + \mathcal{C}} \mathrm{wt}(c)$$

където $\mathrm{wt}(c)$ е теглото на кодовата дума c, т.е. броят не ненулевите битове из двоичната кодова дума.

Нека разгледаме сега алгоритъма за декодиране на получена дума.

Нека източникът е изпратил кодовата дума c, а поради шума на канала се е добавила някаква грешка e. До декодера достига (в общия случай) некодова дума $y = c + e$. Тогава е вярно, че

$$y + c = (c + e) + c = (c + c) + e = e$$

тъй като в $GF(2)$ събирането съвпада с операцията ИЗКЛЮЧВАЩО ИЛИ, а то има свойството $\forall a \left(a \oplus a = 0\right)$.

Оттук сме уверени, че получената дума y и грешката e принадлежат на един и същи съседен клас, както и, че всички възможно грешки при получена дума y са именно думите от този съседен клас.

Тогава това, което се изисква от декодер, използващ метода на максималното правдоподобие, е да намерим най-близката кодова дума $\tilde{c}$. Естествено е също така да допуснем по острието на Окам, че грешката е от минимално тегло - т.е. това е лидерът на съседния клас $y + \mathcal{C}$.

Най-елементарният подход е да изброим всички съседни класове и да си харесваме. Използваме т.нар. стандартна таблица или *таблица на Слепян*.

Щом кодовите думи има дължина k, то всички такива са общо 2^k. Полагаме $M := 2^k$. Имаме, че кодът $\mathcal{C} = \{c_1, \cdots, c_M\}$. Конструираме таблицата от M стълба и 2^{n-k} реда. В първия ред разполагаме кодовите думи от $\mathcal{C}$, започвайки от $\mathcal{O}$. На всеки от следващите t реда, където t е радиусът на опаковка, разполагаме съседен клас $e + \mathcal{C} = e, e + c_1, \cdots, e + c_{M-1}$, сортирайки съседните класове според теглата на техните лидери. *Защо само следващите t реда?* Ами, защото онези съседни класове „под чертата" не можем да декодираме еднозначно.

След като вече имаме генерирана таблица на Слепян, алгоритъмът е следният. Получена дума y, не непременно кодова, търсим из таблицата на Слепян, и интерпретираме лидера на съответния ред като възникналата грешка.

Тогава възстановената кодова дума се получава като сума от получената дума и грешката.

0	c_1	c_2	$\ldots\ldots$	c_{M-1}	
e_1	$e_1 + c_1$	$e_1 + c_2$	$\ldots\ldots$	$e_1 + c_{M-1}$	съседни класове с лидери с тегло $\leq t$
$\ldots\ldots$					
e_s	$e_s + c_1$	$e_s + c_2$	$\ldots\ldots$	$e_s + c_{M-1}$	
e_{s+1}	$e_{s+1} + c_1$	$e_{s+1} + c_2$	$\ldots\ldots$	$e_{s+1} + c_{M-1}$	съседни класове с лидери с тегло $> t$
$\ldots\ldots$					
e_N	$e_N + c_1$	$e_N + c_2$	$\ldots\ldots$	$e_N + c_{M-1}$	

Таблица 1: Стандартна таблица на Слепян

Нека разгледаме една примерна стандартна таблица на Слепян.

```
Number of cosets: 8
Coset size: 16
0       1       2       3       4       5       6       7       8       9       10      11      12
-------------------------------------------------------------------------------------------------
0000000 |  0000000 0010110 0011001 0100101 0101010 1110000 1001100 1000011 0111100 0110011 0001111
0000001 |  0000001 0010111 0011000 0100100 0101011 1110001 1001101 1000010 0111101 0110010 0001110
0000010 |  0000010 0010100 0011011 0100111 0101000 1110010 1001110 1000001 0111110 0110001 0001101
0000011 |  0000011 0010101 0011010 0100110 0101001 1110011 1001111 1000000 0111111 0110000 0001100
0000100 |  0000100 0010010 0011101 0100001 0101110 1110100 1001000 1000111 0111000 0110111 0001011
0000101 |  0000101 0010011 0011100 0100000 0101111 1110101 1001001 1000110 0111001 0110110 0001010
0000110 |  0000110 0010000 0011111 0100011 0101100 1110110 1001010 1000101 0111010 0110101 0001001
0000111 |  0000111 0010001 0011110 0100010 0101101 1110111 1001011 1000100 0111011 0110100 0001000
```

Както забелязвате, този метод е крайно непрактичен по множество причини. Съществуват подходи за неговото опростяване на алгоритъма за общия случай на линеен код. Нека разгледаме метода на *декодиране чрез синдроми*.

Дефиниция

Проверочно съотношение за двоичен код $\mathcal{C}$ се нарича такова хомогенно линейно уравнение

$$p : a_1 x_1 + \cdots a_n x_n = 0$$

за което е изпълнено, че всяка кодова дума $c \in \mathcal{C}$ е решение на p, т.е.

$$a_1 c_1 + \cdots a_n c_n = 0$$

Дефиниция

Двоичен код, определен от проверочните съотношения $p_1, \cdots, p_n$, наричаме множеството от всички двоични думи с дължина n, които са решения на системата уравнения

$$\begin{cases} p_1: & a_{1,1}x_1 + \cdots + a_{1,n}x_n = 0 \\ p_2: & a_{2,1}x_1 + \cdots + a_{2,n}x_n = 0 \\ & \vdots \\ p_s: & a_{s,1}x_1 + \cdots + a_{s,n}x_n = 0 \end{cases}$$

Коефициентите на тази система образуват проверочна матрица за кода $\mathcal{C}$, съставен от това множеството от двоични думи с дължина n.

Проверочна матрица ни интересува, поради следното си основно свойство.

Теорема

Ако имаме линеен код $\mathcal{C}$, който се определя от проверочната си матрица H, то е вярно, че

$$c \in \mathcal{C} \iff cH^T = 0$$

Следващият резултат пък излага как да получим проверочна матрица, ако разполагаме с пораждащата такава.

Теорема

Проверочна матрица на линеен (n, k) код има вида $H = (A|E_{n-k})$ тогава *съществува пораждаща матрица G на този код от вида $G = (E_k|A^T)$.*

Използвайки проверочната матрица можем да получим по-ефикасен алгоритъм за декодиране. Той се основава на следната

Дефиниция

Нека имаме линеен (n, k) *код* $\mathcal{C}$. *Ако* x *е произволен* n*-мерен вектор, синдром на* x *наричаме* $(n-k)$*-мерния вектор*

$$s^T = Hx^T$$

За синдромите можем да направим следните две тривиални наблюдения.

- Произволен вектор $c \in \mathbb{F}^n$ е принадлежи на кода TCTK c има нулев синдром.
- Два вектора $a, b \in \mathbb{F}^n$ имат равни синдроми TCTK принадлежат на един и същи съседен клас на кода. Доказателството на това твърдения е следното:

$$Hx^T = Hy^T \iff H(x+y)^T \iff x+y \in \mathcal{C} \iff x \in y+\mathcal{C}$$

Основавайки се на последното свойство, можем да модифицираме таблицата на Слепян така, че тя да съдържа единствено две колони. В първата имаме лидерите на съседните класове на кода, а във втората - техните синдроми.

Алгоритъмът за декодиране на произволен линеен код става следния. За получен вектор y пресмятаме синдрома $s^T = Hy^T$ и намираме реда от таблицата, чиято дясна колона е полученият синдром. Лидерът, намиращ се в лявата колона е възникналата грешка.

Нека разгледаме една примерна таблица със синдроми.

```
 1   Syndrome | Leader
 2   ------------------
 3   000      | 0000000
 4   001      | 0000001
 5   010      | 0000010
 6   011      | 1000000
 7   100      | 0000100
 8   101      | 0100000
 9   110      | 0010000
10   111      | 0001000
```

Циклични кодове

Дефиниция

Нека имаме вектор $b = [b_0, b_1, \cdots, b_{n-1}]$. *Циклично преместване на* b *наричаме вектор* b', *получен като координатите са разместени така, че втората е записана на първо място, третата на втората и т.н.. Първата е записана на последно място.*

$$b' := [b_1, b_2, \cdots b_0]$$

Дефиниция

Линеен блоков код, който заедно с всяка своя кодова дума $c = [c_0, \cdots, c_{n-1}]$ *съдържа още и неговото циклично завъртане* $c' = [c_1, \cdots, c_0]$, *се нарича* цикличен код.

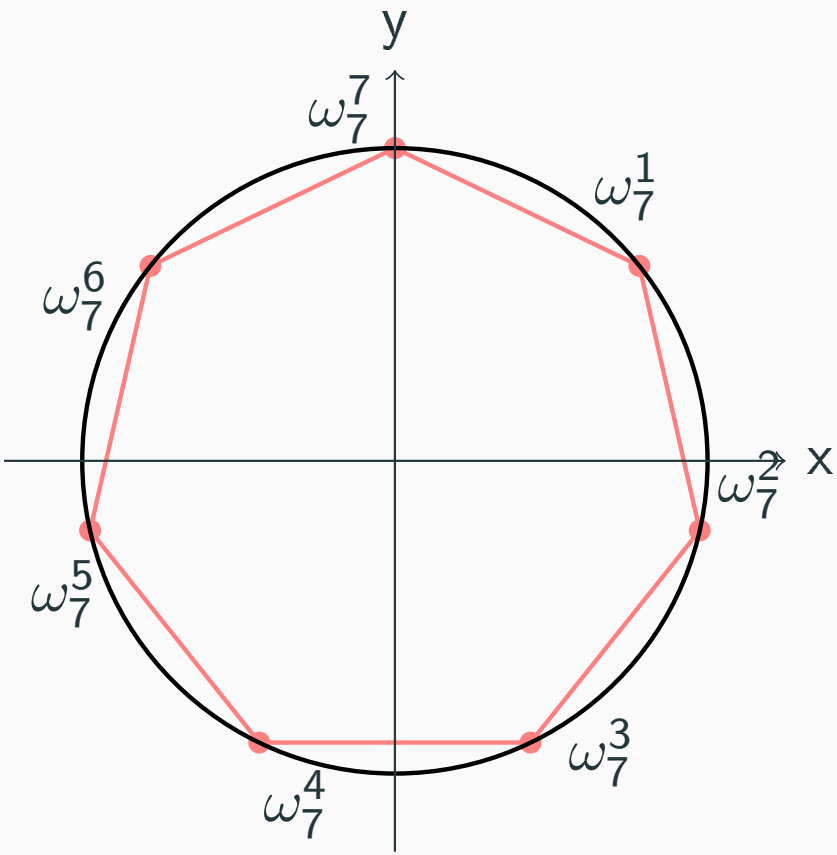
При работа с циклични кодове е удачно да имаме дефинирани не само операциите от линейното пространство, но още и *умножение на вектори* над полето $GF(q)$.

Поради тази причина се използва следното съответствие

$$c = [c_0, \cdots, c_{n-1}] \mapsto c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

и още по-конкретно израза

$$c(\omega) = c_0 + c_1 \omega + \cdots + c_{n-1} \omega^{n-1} \qquad \text{където } \omega^n = 1$$



Фигура 9: Решения на уравнението $x = \sqrt[7]{1}$. **Забележка:** Решенията са точно елементите на цикличната група $\mathbb{C}_7$.

Цикличните кодове, понеже са линейни, имат своя пораждаща матрица. Но освен нея те имат и пораждащ полином -

Дефиниция

Пораждащ полином $g(x)$ *на цикличен код* $\mathcal{C}$ *се нарича полиномът от най-ниска степен, съответстващ на ненулева кодова дума* $g \in \mathcal{C}$. *Бележим* $\mathcal{C} = \langle g \rangle$.

Той е особен, поради свойствата, следващи от следната

Теорема

Ако имаме цикличен код $\mathcal{C} = \langle g \rangle$, то:

- *полиномът $g(x)$ дели $x^n - 1$ без остатък;*
- *вектор c е кодова дума тогава и само тогава $g(x)$ дели $c(x)$;*
- *ако $deg(g) = r$, то $\mathcal{C}$ е $(n, n-r)$ код, а пораждаща матрица G изглежда така -*

$$G = \begin{pmatrix} g_0 & g_1 & \dots & g_r & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & g_r & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & g_0 & g_1 & \dots & g_r \end{pmatrix}$$

Можем да видим един примерен цикличен код, който се поражда от полинома $x^3 + x + 1$, и има кодови думи с дължина 7 бита.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Ха! Изненада! Това отново е $(7, 4)$ кодът на Хеминг! ☺

Сега можем да посочим как работи кодирането на съобщение, използвайки цикличен код.

Нека имаме $\mathcal{C} = \langle g \rangle$. Входното съобщение представяме чрез *информационен полином*

$$i(x) = i_0 + \cdots + i_{k-1}x^{k-1}$$

Имаме два подхода на кодиране, определени от това дали съобщението е част от получената кодова дума, или е „разбъркано". Първият се нарича систематичен и именно него ще обясним тук. В него информационната последователност образува старшите членове на кодовата дума, т.е.

$$c(x) = x^{n-k}i(x) + t(x)$$

Знаейки, че $g(x) \mid c(x)$, можем да използваме теоремата на Евклид -

$$x^{n-k} i(x) = g(x)q(x) + r(x)$$

и тогава

$$c(x) = x^{n-k} i(x) + t(x) = g(x)q(x) + (r(x) + t(x))$$

където $r(x) + t(x)$ е от степен по-малка от $n - k$ и се дели на $g(x)$, откъдето $r(x) = -t(x)$.

Нека разгледаме един пример. Отново ще разгледаме познатия $(7,4)$ код на Хеминг. За него имаме, че $\mathcal{C} = \langle g \rangle$ при $g(x) = x^3 + x + 1$. Да вземем например съобщението 1001. На него съпоставяме информационен полином

$$i(x) = 1 + 0x^1 + 0x^2 + 1x^3 = x^3 + 1$$

Кодовата дума получаваме като

$$c(x) = x^{7-4} i(x) + t(x) = x^3(x^3 + 1) + t(x) = x^6 + x^3 + t(x)$$

Търсим $t(x)$, разделяйки $x^3 i(x)$ на $g(x)$, т.е. $x^6 + x^3$ на $x^3 + x + 1$. Получаваме частно $q(x) = x^3 + x$ и остатък $r(x) = x^2 + x$. Тогава $t(x) = -r(x) = x^2 + x$, понеже говорим за двоични кодове, т.е. в $GF(2)$.

Кодовата дума накра съответства на полинома

$$c(x) = x^6 + x^3 + t(x) = x^6 + x^3 + x^2 + x$$

А самата кодова дума е 1001110.

За да декодираме циклични кодове, въвеждаме следното понятие.

Дефиниция

Нека имаме цикличен (n, k) *код* $\mathcal{C} = \langle g \rangle$. *Полиномът* $h(x)$, *определен от*

$$h(x) := \frac{x^n - 1}{g(x)}$$

наричаме проверочен полином на кода.

За него е верен следният резултат.

Теорема

Нека имаме цикличен (n, k) код $\mathcal{C}$. Тогава вектор $c \in \mathbb{F}^n$ е кодова дума ТСТК *за $c(x)$ е изпълнено*

$$h(\omega).c(\omega) = 0$$

където ω е n-ти корен на единицата.

Знаем, че всяка кодова дума на цикличния код $\mathcal{C} = \langle g \rangle$ се дели без остатък на пораждащия полином. Понеже две думи дават един и същи остатък при деление на $g(x)$ тяхната разлика дава остатък 0, т.е. е кодова дума, получаваме, че думите, които имат един и същи остатък на $g(x)$ принадлежат на един и същи съседен клас на кода. Въвеждаме понятието синдром (за цикличен код).

Дефиниция

Нека имаме цикличен код $\mathcal{C} = \langle g \rangle$. *Тогава синдром на кодовата дума* a *(със съответен полином* $a(x)$*), както и на целия съседен клас* $a + \mathcal{C}$ *наричаме полинома* $s(x)$*, определен от*

$$a(x) = q(x).g(x) + s(x)$$

За степента на $s(x)$ *тогава имаме, че* $deg(s) < (n - k)$.

Разбира се, тъй като цикличните кодове същинско подмножество на линейните кодове, декодирането е напълно възможно и чрез вече разгледания подход с таблица на синдромите.

Директна полза от цикличните кодове обаче е възможността за по-бързо декодиране, а именно т.нар. *декодер на Мегит*.

При него таблицата на синдромите се съкращава значително, поради следното наблюдение. Прилагайки циклично завъртане на всички думи от един съседен клас на цикличен код $\mathcal{C}$, получаваме отново съседен клас на същия този код $\mathcal{C}$.

Поради тази причина в таблицата участват само синдромите на онези съседни класове, които имат лидер с ненулев стари коефициент пред x^{n-1}.

Алгоритъмът изглежда по следния начин. Отново разполагаме с цикличен (n, k) код $\mathcal{C} = \langle g \rangle$. При преноса на кодова дума c на другия край на канала достига дума y, която не е непременно кодова, ами е променена с грешка e -

$$y = c + e$$

За да декодираме, съставяме съкратена таблица със синдроми, в която участват единствено синдромите на съседните класове, които имат лидер с ненулев коефициент пред x^{n-1}. Тогава първо пресмятаме синдрома на и го сравняваме с наличните синдроми от таблицата. Ако намерим същия, поправяме грешката и сме готови. Ако търсения синдром не е в таблицата, то получената дума y се измества циклично в $\hat{y}$. Тогава пресмятаме синдрома на $\hat{y}$ и търсим из таблицата. Продължаваме така, докато намерим синдром в таблицата.

Защо са особено интересни цикличните кодове? Дори без да сме запознати с особеностите на още по-фините БЧХ кодове и кодове на Рид-Соломон, можем да забележим следния факт. Съществува естествено хардуерна асоциация с цикличните кодове, а именно изместващ регистър. Той има две основни „операции" - *изместване* и *обратна връзка*. Първата съвпада еднозначно с основната операция на цикличните кодове - цикличното завъртане; а втората - с операцията събиране в полето $GF(2)$, реализирано чрез ИЗКЛЮЧВАЩО ИЛИ.

Създаване на хардуерни ускорители или копроцесори с цел кодиране и декодиране на циклични кодове и широко разпространено.

Нататък

Следващите две стъпки от нашия преглед са към БЧХ кодовете и кодовете на Рид-Соломон.

Основната идея и на двата вида кодове е да изразим цикличния код чрез корените на пораждащия полином g. Тогава използваме факта, че можем да фиксираме полином от n-та степен с кои да е $n+1$ точки от него.

Дефиниция

Нека n и q са взаимно прости числа, а F_{q^m} е разширание на F_q, което съдържа всички n корена на полинома $x^n - 1$. Нека също α е примитивен n-ти корен на 1 над полето F_q. БЧХ код над F_q с дължина n и конструктивно разстояние δ е цикличен код с дължина n и пораждащ полином

$$g(x) = [M_{\alpha^b}(x), M_{\alpha^{b+1}}(x), \ldots, M_{\alpha^{b+\delta-2}}(x)],$$

където M_{α^s} е минималния полином на елемента α^s над F_q.

Дефиниция

Нека имаме БЧХ код $\mathcal{C}$ над F_q с дължина $q-1$ и конструктивно разстояние δ. Тогава за $\mathcal{C}$ казваме, че е код на Рид-Соломон.

Основното негово предимство е ефикасния декодер - т.нар. декодер на Берлекамп-Мейси, който е приложим. Също така от основната теорема за БЧХ кодове, получаваме конструктивен алгоритъм за създаване на БЧХ кодове, които поправят до t на брой грешки.

Благодаря за вниманието!

Литература

[99] ***IEC 60908:1999 — Compact Disc Digital Audio System***. Defines Cross-Interleaved Reed–Solomon Coding (CIRC) for error correction in audio CDs. International Electrotechnical Commission, 1999. URL: https://standards.iteh.ai/catalog/standards/clc/7e75f508-8dea-4477-8151-dfac03511623/en-60908-1999.

[Ana20] Anadoxin. ***Reed–Solomon Error Recovery in RAID-6***. https://anadoxin.org/blog/error-recovery-in-raid6.html. достъпено на 02.02.2026. 2020.

[Mac03] David J. C. MacKay. ***Information Theory, Inference, and Learning Algorithms***. https://www.inference.org.uk/itprnn/book.pdf достъпено на 02.02.2026. Cambridge University Press, 2003. ISBN: 0-521-64298-1.

[NAS26] NASA. ***Voyager 2 – NASA Science***. https://science.nasa.gov/mission/voyager/voyager-2/. достъпено на 02.02.2026. 2026.