

# Код Рида-Маллера

Илья Коннов

Факультет компьютерных наук  
Высшая Школа Экономики

14 марта 2022 г.

Существует три различных варианта этого доклада:

1. Краткая презентация, которую несложно рассказать, но может быть сложно понять ([ReedMuller-trans.pdf](#)).
2. Более длинная презентация с ценными комментариями, дополнительными доказательствами и интересными фактами ([ReedMuller-slides.pdf](#)).
3. Текстовая статья со всем содержимым длинной презентации, комментариями на своих местах, а также бонусным приложением с более подробным описанием алгоритма ([ReedMuller-article.pdf](#)). **Вы сейчас читаете именно эту версию.** Невошедшее в презентацию помечено линиями слева, а названия слайдов можно найти справа.

Их все можно посмотреть здесь: <https://sldr.xyz/ReedMuller/>

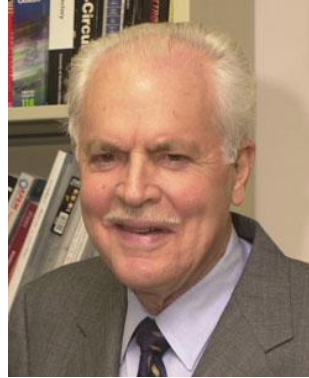
По любым вопросам: [r-m@sldr.xyz](mailto:r-m@sldr.xyz) или [t.me/iliago](https://t.me/iliago) или [vk.com/iliago](https://vk.com/iliago).

# Содержание

<b>1</b>	<b>Введение</b>	<b>3</b>
<b>2</b>	<b>Кодирование</b>	<b>4</b>
<b>3</b>	<b>Свойства кода</b>	<b>7</b>
3.1	Конструкция Плоткина . . . . .	8
3.2	Минимальное расстояние . . . . .	9
3.3	Параметры . . . . .	10
<b>4</b>	<b>Декодирование</b>	<b>11</b>
4.1	Алгоритм Рида . . . . .	12
4.1.1	Пример . . . . .	13
<b>5</b>	<b>Домашнее задание</b>	<b>15</b>
<b>6</b>	<b>Источники</b>	<b>16</b>
<b>A</b>	<b>Reed's Algorithm: Unique decoding up to half the code distance</b>	<b>17</b>
A.1	Дополнительные доказательства . . . . .	19
A.2	Реализация алгоритма . . . . .	22

---

Код описан Дэвидом Маллером (автор идеи) и Ирвингом Ридом (автор метода декодирования) в сентябре 1954 года.



**Авторы**

---

Обозначается как  $\text{RM}(r, m)$ , где  $r$  — ранг, а  $2^m$  — длина кода. Кодировать сообщения длиной  $k = \sum_{i=0}^r C_m^i$  при помощи  $2^m$  бит. Традиционно, считается что коды бинарные и работают над битами, т.е.  $\mathbb{F}_2$ .

Соглашение: сложение векторов  $u, v \in \mathbb{F}_2^n$  будем обозначать как  $u \oplus v = (u_1 + v_1, u_2 + v_2, \dots, u_n + v_n)$ .

**Введение**

## 1 Введение

---

Всякую булеву функцию можно записать при помощи таблицы истинности:

$x$	$y$	$f(x, y)$
0	0	1
0	1	0
1	0	0
1	1	0

Или при помощи многочлена Жегалкина:

$$f(x, y) = xy + x + y + 1$$

**Булевы функции  
и многочлен  
Жегалкина**

---

В общем случае, многочлены будут иметь следующий вид:

$$f(x_1, x_2, \dots, x_m) = \sum_{S \subseteq \{1, \dots, m\}} c_S \prod_{i \in S} x_i$$

Например, для  $m = 2$ :  $f(x_1, x_2) = c_{12} \cdot x_{\{1\}} x_2 + c_{\{2\}} \cdot x_2 + c_{\{1\}} \cdot x_1 + c_{\emptyset} \cdot 1$   
Всего  $n = 2^m$  коэффициентов для описания каждой функции.

**Многочлены  
Жегалкина**

---

Рассмотрим функции, степень многочленов которых не больше  $r$ :

$$\{f(x_1, x_2, \dots, x_m) \mid \deg f \leq r\}$$

Каждую можно записать следующим образом:

$$f(x_1, x_2, \dots, x_m) = \sum_{\substack{S \subseteq \{1, \dots, m\} \\ |S| \leq r}} c_S \prod_{i \in S} x_i$$

**Функции  
небольшой  
степени**

В каждом произведении используется не больше  $r$  переменных.

Замечу, что при  $S = \emptyset$ , мы считаем, что  $\prod_{i \in S} x_i = 1$ , таким образом всегда появляется свободный член.

Сколько тогда всего коэффициентов используется?

$$k = C_m^0 + C_m^1 + C_m^2 + \dots + C_m^r = \sum_{i=0}^r C_m^i$$

Если говорить несколько проще, то для составления многочленов мы сложим сначала одночлены  $(x + y + z + \dots)$ , затем произведения одночленов  $(xy + yz + xz + \dots)$  и т.д. вплоть до  $r$  множителей (поскольку мы работаем в поле  $\mathbb{F}_2$ , здесь нету  $x^2, y^2, z^2$ , т.к.  $a^2 = a$ ). Тогда легко видеть, почему  $k$  именно такое: мы складываем все возможные перестановки сначала для 0 переменных, потом для одной, двух, и так вплоть до  $r$  (не больше, ведь  $\deg f \leq r$ ).

## 2 Кодирование

Пусть каждое сообщение (длины  $k$ ) — коэффициенты многочлена от  $m$  переменных степени не больше  $r$ . Тогда мы можем его представить при помощи  $2^m$  бит, подставив все возможные комбинации значений переменных. Их  $2^m$ , поскольку рассматриваем многочлены только над  $\mathbb{F}_2$  от  $m$  переменных. Таким образом получим таблицу истинности, из которой позднее сможем восстановить исходный многочлен, а вместе с ним и сообщение.

Зафиксировав в таблице порядок строк, можно выделить **вектор значений**, который и будет кодом.

$x$	$y$	$f(x, y)$	
0	0	1	
0	1	0	$\Rightarrow \text{Eval}(f) = (1 \ 0 \ 0 \ 0)$
1	0	0	
1	1	0	

Вектор значений — обозначается  $\text{Eval}(f)$  — столбец таблицы истинности, содержащий значения функции. Имеет смысл только при зафиксированном порядке строк в таблице. У меня он везде самый обычный, как в примере выше.

Здесь и далее я для краткости и удобства записываю битовые векторы не как  $(1 \ 0 \ 0 \ 1)$ , а как **1001** при помощи нескучного шрифта.

Для кодирования очень важно понимать, как именно биты сообщения ставятся в соответствие коэффициентам многочлена. Поэтому давайте введём **соглашение**: если упорядочить элементы множества  $u$  каждого коэффициента по возрастанию, то коэффициенты сортируются в лексикографическом порядке:  $c_{1,2}$  раньше  $c_{1,3}$ , поскольку  $2 < 3$  и  $c_{2,3}$  раньше  $c_{3,4}$ , поскольку  $2 < 3$ .

*Идея кодирования*

*Пример*

Пример для  $m = 4$ :

$$\begin{aligned}
 f(x_1, x_2, x_3, x_4) = & c_{\{1,2,3,4\}}x_1x_2x_3x_4 \\
 & + c_{\{1,2,3\}}x_1x_2x_3 + c_{\{1,2,4\}}x_1x_2x_4 + c_{\{1,3,4\}}x_1x_3x_4 + \\
 & \quad + c_{\{2,3,4\}}x_2x_3x_4 \\
 & + c_{\{1,2\}}x_1x_2 + c_{\{1,3\}}x_1x_3 + c_{\{1,4\}}x_1x_4 + c_{\{2,3\}}x_2x_3 + \\
 & \quad + c_{\{2,4\}}x_2x_4 + c_{\{3,4\}}x_3x_4 \\
 & + c_{\{1\}}x_1 + c_{\{2\}}x_2 + c_{\{3\}}x_3 + c_{\{4\}}x_4 + c_{\emptyset}
 \end{aligned}$$

Также можно кодировать множества при помощи битов, используя отношение  $x \in A \iff v_x = 1$  (нумерация битов слева направо, начиная с единицы), где свойство остортированности сохраняется и хорошо видно (но только в пределах группы мономов одной степени):

$$\begin{aligned}
 f(x_1, x_2, x_3, x_4) = & c_{1111}x_1x_2x_3x_4 \\
 & + c_{1110}x_1x_2x_3 + c_{1101}x_1x_2x_4 + c_{1011}x_1x_3x_4 + c_{0111}x_2x_3x_4 \\
 & + c_{1100}x_1x_2 + c_{1010}x_1x_3 + c_{1001}x_1x_4 + c_{0110}x_2x_3 + \\
 & \quad + c_{0101}x_2x_4 + c_{0011}x_3x_4 \\
 & + c_{1000}x_1 + c_{0100}x_2 + c_{0010}x_3 + c_{0001}x_4 + c_{0000}
 \end{aligned}$$

С помощью этого примера легко увидеть порядок для всех остальных конфигураций кода, если вычеркнуть заведомо невозможные слагаемые (напр., содержащие  $x_4$  для  $m = 3$  или мономы слишком большой степени для  $r < 4$ ).

- $r = 1$  (степень многочлена),  $m = 2$  (переменных). Это  $\text{RM}(1, 2)$ .
- Тогда наш многочлен:  $f(x_1, x_2) = c_{\{2\}}x_2 + c_{\{1\}}x_1 + c_{\emptyset}$ .
- Сообщение: 011, тогда  $f(x_1, x_2) = 0 + x_1 + 1$ .
- Подставим всевозможные комбинации:

$x_1$	$x_2$	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	0

Обратите внимание на то, какой используется порядок переменных в таблице истинности. Очень важно чтобы при кодировании и декодировании было согласие и взаимопонимание касательно того, какому набору переменных соответствует каждая строчка.

- Получили код:  $\text{Eval}(f) = 1100$ .

Теперь покажем, как можно декодировать когда потерь нет. Этот пример — продолжение предыдущего.

- Мы получили код: 1100

*Декодирование  
когда потерь нет*

- Представим таблицу истинности.

$x_1$	$x_2$	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	0

- Подстановками в  $f(x_1, x_2) = c_2x_2 + c_1x_1 + c_0$  получим СЛАУ.
 
$$\begin{cases} c_0 = 1 \\ c_2 + c_0 = 1 \\ c_1 + c_0 = 0 \\ c_1 + c_2 + c_0 = 0 \end{cases}$$
- $c_{\{1\}} = 1, c_{\{2\}} = 0, c_{\emptyset} = 1$ , исходное сообщение: 011.

### Коды 0-го порядка

Отдельно стоит рассмотреть вариант кода при  $r = 0$ , он нам в будущем пригодится для доказательств.

Для случая  $\text{RM}(0, m)$  нужна функция от  $m$  аргументов, степени не выше 0. Таких функций существует всего лишь две, поскольку мы можем влиять лишь на свободный член. Все остальные коэффициенты обнуляются из-за требования  $\deg f \leq 0$ .

- $f(x_1, x_2, \dots, x_m) = 0$
- $g(x_1, x_2, \dots, x_m) = 1$

Таблица истинности:

$x_1$	$x_2$	...	$x_m$	$f(x_1, \dots, x_m)$	$g(x_1, \dots, x_m)$
0	0	...	0	0	1
0	0	...	1	0	1
		$\ddots$		$\vdots$	$\vdots$
1	1	...	1	0	1

Здесь число строк, как и в любой другой таблице истинности, равно  $2^m$ , а колонки со значениями никак не зависят от аргументов функций. Получается две колонки – одна с нулями, другая с единицами.

Вывод: это  $2^m$ -кратное повторение символа

- Сообщение 0 даст код  $\underbrace{00\dots 0}_{2^m}$
- Сообщение 1 даст код  $\underbrace{11\dots 1}_{2^m}$

### Коды $m$ -го порядка

Есть ещё один тривиальный случай, когда  $m = r$ .

Есть  $m$  переменных, и мы рассматриваем многочлены  $f \in \mathbb{F}_2[x_1, \dots, x_m]$  :  $\deg f \leq m$ , т.е. все возможные. Для  $\text{RM}(m, m)$  мы используем все доступные коэффициенты многочлена для кодирования сообщения. Тогда нет избыточности:  $k = \sum_{i=0}^m C_m^i = 2^m = n$  – длина сообщения равна длине кода.

Чем меньше порядок кода  $r$ , тем больше избыточность.

### 3 Свойства кода

#### Доказательство линейности

Хотим показать, что этот код является линейным, т.е. что его кодовые слова образуют линейное пространство, и у нас есть изоморфизм из пространства сообщений ( $\mathbb{F}_2^k$ ) в пространство слов ( $\mathbb{F}_2^m$ ).

Для этого необходимо немного формализовать всё описанное раньше.

Пусть  $C(x)$  кодирует сообщение  $x \in \mathbb{F}_2^k$  в код  $C(x) \in \mathbb{F}_2^m$ .

$$C(x) = (p_x(a_i) \mid a_i \in \mathbb{F}_2^m)$$

где  $p_x(a_i)$  — соответствующий сообщению  $x$  многочлен. Пояснение: перебираем все векторы  $a_i$  ( $2^m$  штук), подставляем каждый в  $p_x$  в качестве переменных и таким образом получаем вектор значений (длины  $2^m$ ). Именно он и называется кодом.

Причём  $p_x$  берёт в качестве своих коэффициентов биты из  $x$ . Поскольку многочлены степени не выше  $r$  образуют линейное пространство, то  $p_{(x \oplus y)} = p_x + p_y$ .

Напомним, что базис пространства многочленов выглядит примерно так:  $1, x, y, z, xy, yz, xz$  (для трёх переменных, степени не выше 2).

Чтобы преобразовать сообщение в многочлен, мы берём каждый бит сообщения и умножаем его на соответствующий базисный вектор. Очевидно, такое преобразование будет изоморфизмом. Именно поэтому  $p_{(x \oplus y)} = p_x + p_y$ . Обратите внимание, что сообщение  $x$  это не просто число ( $\mathbb{Z}_{2^k}$ ) и мы рассматриваем его биты, а реально вектор битов ( $\mathbb{Z}_2^k$ ). У него операция сложения побитовая.

Тогда:

$$C(x \oplus y)_i = p_{(x \oplus y)}(a_i) = p_x(a_i) + p_y(a_i) = C(x)_i + C(y)_i$$

т.е.  $\forall x, y \quad C(x \oplus y) = C(x) + C(y)$ , ч.т.д.

Здесь я использую запись  $C(x)_i$  для  $i$ -го элемента вектора  $C(x)$ . Поскольку  $i$  произвольное, то и весь вектор получился равен. Таким образом, этот код действительно линейный и к нему применимы уже известные теоремы!

#### Последствия линейности

1. Существует порождающая матрица  $G$ .

$$C(x) = x_{1 \times k} G_{k \times n} = c_{1 \times n}$$

Так можно кодировать сообщения  $x$  в коды  $c$ . Но искать её мы не будем, обойдёмся одними многочленами, это интереснее.

2. Минимальное расстояние будет равно минимальному весу Хемминга среди всех кодов. Вес Хемминга вектора — количество в нём ненулевых элементов.

$$d = \min_{\substack{c \in C \\ c \neq 0}} w(c)$$

Доказательство очень просто: минимальное расстояние — вес разности каких-то двух различных кодов, но разность двух кодов тоже

будет кодом, т.к. мы в линейном пространстве. Значит достаточно найти минимальный вес, но не учитывая нулевой вектор, т.к. разность равна нулю тогда и только тогда, когда коды равны.

3. Корректирующая способность:

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

Однако мы ещё не знаем как выглядят наши коды (как выглядят таблицы истинности функций степени не больше  $r$ ?). А значит не можем ничего сказать про минимальное расстояние.

### 3.1 Конструкция Плоткина

Хотим понять как выглядят кодовые слова.

- Код — вектор значений функции  $f(x_1, \dots, x_m) \in \text{RM}(r, m)$ , причём  $\deg f \leq r$ . Порядок очевидно не больше  $r$ , потому что это условие для включения в пространство кодов  $\text{RM}(r, m)$ .
- Разделим функцию по  $x_1$ :  $f(x_1, \dots, x_m) = g(x_2, \dots, x_m) + x_1 h(x_2, \dots, x_m)$ . Теперь у нас есть две функции от меньшего числа аргументов. Очевидно, так можно сделать всегда, когда  $m > 1$ .
- Заметим, что  $\deg f \leq r$ , а значит  $\deg g \leq r$  и  $\deg h \leq r - 1$ .

**Конструкция  
Плоткина:**  
*многочлены*

Теперь рассмотрим те же функции, но со стороны их таблиц истинности. Нам же интересны именно коды, а они как раз очень тесно связаны с этими таблицами.

Ранее:  $f(x_1, \dots, x_m) = g(x_2, \dots, x_m) + x_1 h(x_2, \dots, x_m)$ .

- Заметим, что таблица истинности  $f$  состоит из двух частей: при  $x_1 = 0$  и при  $x_1 = 1$ .

$$\text{Eval}(f) = \begin{pmatrix} \text{Eval}^{[x_1=0]}(f) \\ \text{Eval}^{[x_1=1]}(f) \end{pmatrix}$$

Про обозначения:  $\text{Eval}(f)$  — таблица для всей функции (вектор значений, если точнее),  $\text{Eval}^{[x_1=0]}(f)$  — кусок таблицы при  $x_1 = 0$ ,  $\text{Eval}^{[x_1=1]}(f)$  — кусок таблицы при  $x_1 = 1$ . Они нам после этого доказательства больше не понадобятся.

- Причём  $\text{Eval}^{[x_1=0]}(f) = \text{Eval}(g)$ , а  $\text{Eval}^{[x_1=0]}(f) \oplus \text{Eval}^{[x_1=1]}(f) = \text{Eval}(h)$ . Это всё следует из ранее полученного утверждения. Если мы подставим  $x_1 = 0$ , то останется только  $g$  — первое равенство очевидно. Если же мы рассмотрим  $\text{Eval}^{[x_1=1]}(f)$ , то получим  $\text{Eval}(g + h)$ , но если туда прибавить ещё раз  $\text{Eval}(g)$ , то останется только  $\text{Eval}(h)$  (поскольку  $1 + 1 = 0$  в  $\mathbb{F}_2$ ) — получили второе равенство.

**Конструкция  
Плоткина:**  
*таблица  
истинности*



- Таким образом,  $\text{Eval}(f) = (\text{Eval}(g) \mid \text{Eval}(g) \oplus \text{Eval}(h))$ . Палочка по центру — конкатенация векторов.

Теперь собираем всё это в одно важное утверждение.

Если дана  $f(x_1, \dots, x_m)$ , причём  $\deg f \leq r$ , то можно её разделить:

$$f(x_1, \dots, x_m) = g(x_2, \dots, x_m) + x_1 h(x_2, \dots, x_m)$$

Причём мы уже знаем, что  $\deg g \leq r$  и  $\deg h \leq r - 1$ , если  $\deg f \leq r$

Также известно, что  $\text{Eval}(f) = (\text{Eval}(g) \mid \text{Eval}(g) \oplus \text{Eval}(h))$ .

Заметим, что  $\text{Eval}(f)$  – кодовое слово (как и для  $g$  и  $h$ ).

Тогда:  $c = \text{Eval}(f) \in \text{RM}(r, m)$  (т.к.  $\deg f \leq r$ )

$u = \text{Eval}(g) \in \text{RM}(r, m - 1)$  (т.к.  $\deg g \leq r$ )

$v = \text{Eval}(h) \in \text{RM}(r - 1, m - 1)$  (т.к.  $\deg h \leq r - 1$ )

Напомню, что  $\text{RM}(r, m)$  включает в себя **все** функции (их таблицы истинности, если точнее) от  $m$  аргументов и степени не выше  $r$ . Очевидно, наши годятся.

**Конструкция  
Плоткина: вывод**

**Теорема.** Для всякого кодового слова  $c \in \text{RM}(r, m)$  можно найти  $u \in \text{RM}(r, m - 1)$  и  $v \in \text{RM}(r - 1, m - 1)$ , такие что  $c = (u \mid u + v)$ .

**Конструкция  
Плоткина**

Что здесь важно отметить — оба наших новых кодовых слова  $u, v$  получились «меньше», чем исходное  $c$ .

Это позволяет, во-первых, устраивать индукцию, чем мы скоро и займёмся. Во-вторых, это позволяет легко строить большие порождающие матрицы, но мы этим не будем заниматься.

## 3.2 Минимальное расстояние

Хотим найти минимальное расстояние для кода  $\text{RM}(r, m)$

**Минимальное  
расстояние**

$$d = \min_{c \in C, c \neq 0} w(c)$$

Предположим, что  $d = 2^{m-r}$  и докажем по индукции.

**База:**  $\text{RM}(0, m)$  — единственный бит повторён  $2^m$  раз. Очевидно,  $w(\underbrace{11\dots 1}_{2^m}) = 2^m = 2^{m-0} \geq 2^{m-r}$ .

Случай  $\text{RM}(0, m)$  мы разбирали раньше, но я напомню. Здесь длина сообщения равна  $k = \sum_{i=0}^r C_m^i = C_m^0 = 1$ , а длина кода  $n = 2^m$ . Причём мы просто берём один бит и повторяем его  $2^m$  раз (в таблице истинности).

Замечу, что не рассматриваю второй случай  $w(00\dots 0)$ , поскольку он нам не нужен для расчёта минимального расстояния. Вариант с нулевым вектором явно выкидывается, см. определение  $d$  выше.

**Гипотеза:** Если  $v \in \text{RM}(r - 1, m - 1)$ , то  $w(v) \geq 2^{m-r}$ .

**Шаг:** Хотим доказать для  $c \in \text{RM}(r, m)$ .

$$\begin{aligned} w(c) &\stackrel{(1)}{=} w((u \mid u \oplus v)) \stackrel{(2)}{=} w(u) + w(u \oplus v) \geq \\ &\stackrel{(3)}{\geq} w(u) + (w(v) - w(u)) = w(v) \stackrel{IH}{\geq} 2^{m-r} \blacksquare \end{aligned}$$

Теперь немного объяснений.

Переход (1): используем конструкцию Плоткина, чтобы разбить  $c$  на конкатенацию двух кодовых слов поменьше.

Переход (2):  $w((x \mid y)) = w(x) + w(y)$ . Вес это всего лишь число ненулевых элементов, поэтому нет разницы как мы будем группировать части вектора.

Переход (3):  $w(u \oplus v) \geq w(v) - w(u)$ . Если у нас в  $v$  стоит  $w(v)$  бит, то прибавив к нему  $u$ , мы сможем изменить (обнулить) не больше  $w(u)$  бит. Возможно появится больше единиц, но нас интересует нижняя граница.

Переход (ИН): предположение индукции в чистом виде.

**Код с весом  $2^{m-r}$**

До этого мы доказали, что расстояние между кодами не может превышать  $2^{m-r}$ . Однако из этого не следует, что код с таким весом действительно существует. Поэтому чтобы завершить доказательство того, что минимальное расстояние  $d = 2^{m-r}$ , нужно показать существование такого кода.

**Дано:**  $\text{RM}(r, m)$ ,  $0 \leq r \leq m$

**Хотим:** такой  $c \in \text{RM}(r, m)$ , что  $w(c) = 2^{m-r}$

Рассмотрим функцию:

$$f(x_1, x_2, \dots, x_m) = \prod_{i=1}^r x_i = x_1 x_2 \dots x_r$$

Очевидно,  $\deg(f) \leq r$ , а значит она подходит под требования  $\text{RM}(r, m)$ .

В её таблице истинности ровно  $2^{m-r}$  строк, когда  $f(\dots) = 1$ :

$\overbrace{x_1 \ x_2 \ \dots \ x_r}^r$				$\overbrace{x_{r+1} \ \dots \ x_m}^{m-r}$			$f$
1	1	...	1	*	...	*	1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	...	1	*	...	*	1

Небольшое пояснение: функция равна единице тогда и только тогда, когда  $x_1 = x_2 = \dots = x_r = 1$ . Получается,  $r$  аргументов из  $m$  зафиксированы, но другие могут меняться произвольно. Получается как раз  $2^{m-r}$  вариантов. На этом доказательство о минимальном весе можно завершить.

### 3.3 Параметры

Теперь можно подвести итоги исследования свойств. Для бинарного кода  $\text{RM}(r, m)$ :

**Свойства и параметры**

- $0 \leq r \leq m$
- Длина кода:  $2^m$
- Длина сообщения:  $k = \sum_{i=0}^r C_m^i$
- Минимальное расстояние:  $d = 2^{m-r}$
- Корректирующая способность:  $t = 2^{m-r-1} - 1$ , поскольку  $t = \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{2^{m-r}-1}{2} \rfloor = \lfloor 2^{m-r-1} - 0.5 \rfloor = 2^{m-r-1} - 1$

- Существует порождающая матрица  $G$  для кодирования, она позволяет делать так:  $C(x) = xG$ . Но я, как обычно, её избегаю. Рекомендую почитать «Коды Рида-Маллера: Примеры исправления ошибок», если интересно.
- Проверочная матрица  $H$  совпадает с порождающей для  $RM(m - r - 1, m)$ , но это я это доказывать не собираюсь. Однако доказательство можно найти в «Reed-Muller Codes: Theory and Algorithms», раздел Duality.

*Возможные варианты*

$m \backslash r$	0	1	2	3	4
1	$k = 1$ $n = 2$ $t = 0$	$k = 2$ $n = 2$ $t = 0$	—	—	—
2	$k = 1$ $n = 4$ $t = 1$	$k = 3$ $n = 4$ $t = 0$	$k = 4$ $n = 4$ $t = 0$	—	—
3	$k = 1$ $n = 8$ $t = 3$	$k = 4$ $n = 8$ $t = 1$	$k = 7$ $n = 8$ $t = 0$	$k = 8$ $n = 8$ $t = 0$	—
4	$k = 1$ $n = 16$ $t = 7$	$k = 5$ $n = 16$ $t = 3$	$k = 11$ $n = 16$ $t = 1$	$k = 15$ $n = 16$ $t = 0$	$k = 16$ $n = 16$ $t = 0$

У красных кодов минимальное расстояние  $d$  равно единице — они совершенно бесполезны, там количество кодов равно количеству сообщений; у желтых кодов  $d = 2$  — они могут определить наличие ошибки, но не могут её исправить. Для всех остальных кодов  $d = 2(t + 1)$ .

Напоминание:  $k$  — длина сообщения,  $n$  — длина кода, а  $t$  — количество ошибок, которое код точно сможет исправить. Заодно о параметрах кода:  $m$  — количество переменных у функции (очень влияет на длину кода), а  $r$  — максимальная степень многочлена (очень влияет на длину сообщения, и соответственно надёжность кода), причём  $r \leq m$ . Конечно, таблицу можно продолжать и дальше.

И кстати, случай  $m = 0, k = 0$  (не влез) будет собой представлять кодирование единственного бита совершенно без изменений.

## 4 Декодирование

*Как линейный код*

Этот код является линейным кодом, к нему применимы все обычные (и неэффективные методы):

- Перебор по всему пространству кодовых слов в поисках ближайшего. Этот способ применим ко всем кодам, но никто в здравом уме им не пользуется.

- С использованием синдромов:  $s = rH^T$ . Здесь  $s$  — синдром,  $r$  — полученное сообщение,  $H$  — проверочная матрица. Этот метод обычен для линейных кодов.

Эти способы нужно иметь в виду, но о них было рассказано и без меня, так что я их пропущу.

## 4.1 Алгоритм Рида

Начать стоит с нескольких определений, без которых алгоритм Рида объяснить не получится.

### Определения

1. Пусть  $A \subseteq \{1, \dots, m\}$  для  $m \in \mathbb{N}$
2. Подпространство  $V_A \subseteq \mathbb{F}_2^m$ , которое обнуляет все  $v_i$ , если  $i \notin A$ :  
 $V_A = \{v \in \mathbb{F}_2^m : v_i = 0 \ \forall i \notin A\}$
3. Аналогично для  $V_{\bar{A}}$ , где  $\bar{A} = \{1, \dots, m\} \setminus A$ :  $V_{\bar{A}} = \{v \in \mathbb{F}_2^m : v_i = 0 \ \forall i \in A\}$

Пример:

- Пусть  $m = 3, A = \{1, 2\}$ , тогда...
- $\mathbb{F}_2^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$  — все 8 векторов этого пространства
- $V_A = \{000, 010, 100, 110\}$  ( $v_3 = 0 \ \forall v$ ) — обнулилась третья позиция, первые две остались
- $\bar{A} = \{1, 2, 3\} \setminus A = \{3\}$
- $V_{\bar{A}} = \{000, 001\}$  ( $v_1 = v_2 = 0 \ \forall v$ ) — осталась только третья позиция, остальные обнулились.

Если фиксировано  $V_A \subseteq \mathbb{F}_2^m$ , то для каждого  $b \in \mathbb{F}_2^m$  существует смежный класс  $V_A + b$ :

### Смежные классы

$$(V_A + b) = \{v + b \mid v \in V_A\}$$

Утверждается, что если брать  $b \in V_{\bar{A}}$ , то полученные смежные классы будут все различны (и это будут все смежные классы). Почему все смежные классы  $(V_A + b)$  можно получить именно перебором  $b \in V_{\bar{A}}$  можно найти в разделе «Дополнительные доказательства» из пдфки [\[ссылка\]](#)

Теперь, наконец, сам алгоритм Рида с объяснением, что тут происходит. Почему он именно такой и почему это работает — см. раздел (на русском) «Reed's Algorithm: Unique decoding up to half the code distance» [\[A\]](#) в пдфке.

### Алгоритм Рида для кода $RM(r, m)$

Декодирует сообщение  $u$ , если использовался  $RM(r, m)$ . Для  $RM(2, 2)$ :  
 $f(x_1, x_2) = u_{\{1,2\}}x_1x_2 + u_{\{2\}}x_2 + u_{\{1\}}x_1 + u_{\emptyset}$ .

---

```

Data: vector  $y = (y_z \in \mathbb{F}_2 \mid z \in \mathbb{F}_2^m)$ 
for  $t \leftarrow r$  to 0 do
  foreach  $A \subseteq \{1, \dots, m\}$  with  $|A| = t$  do
     $c = 0$ 
    foreach  $b \in V_{\bar{A}}$  do
       $c += \left( \sum_{z \in (V_A + b)} y_z \right) \bmod 2$ 
    end
     $u_A \leftarrow \mathbf{1} [c \geq 2^{m-t-1}]$ 
  end
   $y -= \text{Eval} \left( \sum_{\substack{A \subseteq \{1, \dots, m\} \\ |A| = t}} u_A \prod_{i \in A} x_i \right)$ 
end

```

---

На вход поступает бинарный вектор  $y$  длины  $2^m$ . Это вектор значений функции, возможно с ошибками (но их не больше, чем  $t = 2^{m-r-1} - 1$ ). Цель — восстановить все коэффициенты при многочлене вида  $f(x_1, \dots, x_m) = u_\emptyset + u_1 x_1 + x_2 x_2 + \dots + u_{1,2,\dots,r} x_{1,2,\dots,r}$ , где  $\deg f \leq r$ . Обратите внимание, что для индексов при  $u$  используются подмножества  $A \subseteq \{1, \dots, m\}$ ,  $|A| \leq r$ , причём каждый  $u_A$  умножается на моном  $\prod_{i \in A} x_i$ .

Будем восстанавливать сначала коэффициенты  $u_A$  при старших степенях, потом поменьше и так пока не восстановим их все. Начинаем с  $t = r$ . Хотим восстановить все коэффициенты при мономах степени  $t$ . Для этого перебираем все  $A$ ,  $|A| = t$  и для каждого восстанавливаем коэффициент  $u_A$  при  $x_{A_1} x_{A_2} \dots x_{A_t}$ .

Чтобы восстановить коэффициент, нужно перебрать все смежные классы вида  $(V_A + b)$ :

$$\begin{aligned}
 V_A &= \{v \in \mathbb{F}_2^m \\
 &\quad : v_i = 0 \forall i \notin A\} \\
 b &\in \{v \in \mathbb{F}_2^m \\
 &\quad : v_i = 0 \forall i \in A\}
 \end{aligned}$$

Считаем количество ( $c$ ) смежных классов, в которых  $\sum_{z \in (V_A + b)} y_z = 1 \pmod{2}$ . Если это количество больше порогового значения, то считаем, что  $u_A = 1$ , иначе же  $u_A = 0$ . Пороговое значение  $(2^{m-t-1})$  здесь — половина от числа смежных классов. Таким образом, если большинство сумм дало 1, то  $u_A = 1$ , иначе  $u_A = 0$ .

Затем мы вычитаем из  $y$  (вектор значений функции) всё найденное на этой итерации, после чего переходим к мономам меньшей степени. Повторять до восстановления всех коэффициентов.

#### 4.1.1 Пример

---

Ранее: 011 кодируется как 1100 при помощи  $\text{RM}(1, 2)$  (см. **самый первый пример**).

*Пример*

Положим  $y_{00} = 1, y_{01} = 1, y_{10} = 0, y_{11} = 0$  — именно так, поскольку  $1100$  — вектор значений, который мы сейчас распаковываем обратно в таблицу истинности. В индексе при  $y$  находится вектор значений переменных, а его ( $y$ ) значение — значение функции при этих аргументах.

Здесь  $m = 2$ , значит  $A \subseteq \{1, 2\}$ . Причём  $r = 1$ , т.е.  $|A| \leq 1$ .

Как происходит кодирование, схематически:

$$101 \rightsquigarrow (f(x_1, x_2) = x_1 + 1) \rightsquigarrow \begin{array}{c|c|c} x_1 & x_2 & f \\ \hline 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{array} \rightsquigarrow \begin{array}{l} y_{00} = 1 \\ y_{01} = 1 \\ y_{10} = 0 \\ y_{11} = 0 \end{array} \rightsquigarrow 1100$$

Теперь начинаем декодирование.

Шаг 1/3:  $t = 1, A = \{1\}$

- Здесь  $V_A = \{00, 10\}$  (меняется только первый бит),  $V_{\bar{A}} = \{00, 01\}$  (первый бит обнулится). Нужно рассмотреть два смежных класса — по одному на каждый вектор из  $V_{\bar{A}}$ .
- $(V_A + 00) = \{00, 10\}$ , сумма:  $y_{00} + y_{10} = 1 + 0 = 1$
- $(V_A + 01) = \{01, 11\}$ , сумма:  $y_{01} + y_{11} = 1 + 0 = 1$
- Итого:  $u_A = u_{\{1\}} = 1$

Шаг 2/3:  $t = 1, A = \{2\}$

- Здесь  $V_A = \{00, 01\}$ ,  $V_{\bar{A}} = \{00, 10\}$ . Нужно рассмотреть два смежных класса — по одному на каждый вектор из  $V_{\bar{A}}$ .
- $(V_A + 00) = \{00, 01\}$ , сумма:  $y_{00} + y_{01} = 1 + 1 = 0$
- $(V_A + 10) = \{10, 11\}$ , сумма:  $y_{10} + y_{11} = 0 + 0 = 0$
- Итого:  $u_A = u_{\{2\}} = 0$

Перед переходом к  $t = 0$ , нужно вычесть из  $y$  вектор значений следующей функции:

$$g(x_1, x_2) = u_{\{2\}}x_2 + u_{\{1\}}x_1 = 0x_2 + 1x_1 = x_1$$

Здесь мы берём все  $u$ , полученные при  $t = 1$ , домножаем каждую на соответствующие ей  $x$ -ы и получаем функцию от  $m$  переменных.

Вычислим  $\text{Eval}(g)$ :

$x_1$	$x_2$	$g(x_1, x_2)$
0	0	0
0	1	0
1	0	1
1	1	1

Очень важно, чтобы у вас во всех таблицах истинности (в т.ч. той, которая использовалась при кодировании для получения  $y$ ) был одинаковый порядок строк. Иначе чуда не выйдет.

Тогда  $y \leftarrow y - \text{Eval}(g) = 1100 \oplus 0011 = 1111$ . Полезно заметить, что в  $\mathbb{F}_2$  сложение и вычитание — одно и то же.

*Продолжение  
примера:  $t = 0$*

Теперь  $y_{00} = 1, y_{01} = 1, y_{10} = 1, y_{11} = 1$

Шаг 3/3:  $t = 0, A = \emptyset$

- Здесь  $V_A = \{00\}$ , но  $V_{\bar{A}} = \{00, 01, 10, 11\}$ . Нужно рассмотреть **четыре** смежных класса.
- $(V_A + 00) = \{00\}$ , сумма:  $y_{00} = 1$
- $(V_A + 01) = \{01\}$ , сумма:  $y_{01} = 1$
- $(V_A + 10) = \{10\}$ , сумма:  $y_{10} = 1$
- $(V_A + 11) = \{11\}$ , сумма:  $y_{11} = 1$
- Итого:  $u_A = u_{\emptyset} = 1$

Получили  $u_{\{2\}} = 0, u_{\{1\}} = 1, u_{\emptyset} = 1$ . Это значит, что исходный многочлен был таков:

$$f(x_1, x_2) = u_{\{2\}}x_2 + u_{\{1\}}x_1 + u_{\emptyset} = 0 + x_1 + 1,$$

а исходное сообщение: 011, как и ожидалось.

### Время работы

Утверждается, что время работы алгоритма —  $O(n \log^r n)$ , где  $n = 2^m$  — длина кода.

## 5 Домашнее задание

Замечание: каких-либо требований на методы решения нет, но если используете код — приложите его. Различных способов решить существует больше одного.

Номер варианта можете определять как  $1 + ((5n + 98) \bmod 2)$ , но главное напишите его и своё имя.

Для кодирования использовался тот же порядок строк в таблице истинности, что и в остальной презентации; аргументы идут по столбцам слева направо по возрастанию номера. При формировании сообщения, слагаемые сортируются лексикографически, а затем по убыванию степени (см. примеры в презентации).

### Вариант 1

1. Закодировать сообщение: 1001.
2. Декодировать код, если ошибок нет: 1010, использовался  $\text{RM}(1, 2)$ .

3. Декодировать код, полученный с ошибками: 1101 1010, использовался  $RM(1, 3)$

---

### Вариант 2

1. Закодировать сообщение: 0101.
2. Декодировать код, если ошибок нет: 0110, использовался  $RM(1, 2)$ .
3. Декодировать код, полученный с ошибками: 1111 0100, использовался  $RM(1, 3)$

## 6 Источники

1. <https://arxiv.org/pdf/2002.03317.pdf> — великолепный обзор, очень рекомендую.
2. <http://dha.spb.ru/PDF/ReedMullerExamples.pdf> — очень хорошо и подробно, но используется подход через матрицы, а не через полиномы, а это не весело.
3. [https://en.wikipedia.org/wiki/Reed-Muller\\_code](https://en.wikipedia.org/wiki/Reed-Muller_code) — кратко, чётко, понятно, но не описано декодирование.
4. [https://ru.bmstu.wiki/Коды\\_Рида-Маллера](https://ru.bmstu.wiki/Коды_Рида-Маллера) — в целом всё есть, но написано очень непонятно;



Это вольный перевод раздела V-A из «[Reed-Muller Codes: Theory and Algorithms](#)» с моими комментариями и некоторыми дополнительными доказательствами.

## A Reed's Algorithm: Unique decoding up to half the code distance

В этом разделе описывается алгоритм Рида для  $\text{RM}(r, m)$ . Он исправляет любые ошибки, вес которых не превышает  $2^{m-r-1}$ , половину минимального расстояния кода.

Для подмножества  $A \subseteq \{1, \dots, m\}$  определим моном  $x_A = \prod_{i \in A} x_i$ , где  $x_i$  — аргументы булевой функции [напр.,  $x_{\{1,2\}} = x_1 x_2$ ]. Также будем использовать  $V_A := \{z \in \mathbb{F}_2^m : z_i = 0 \forall i \notin A\}$  для обозначения подпространства в  $\mathbb{F}_2^m$  размерности  $|A|$ , т.е.  $V_A$  это подпространство, в котором для всех векторов  $z$  зафиксированы биты  $z_i = 0$  при  $i \notin A$ . Для подпространства  $V_A$  (в пространстве  $\mathbb{F}_2^m$ ) существует  $2^{m-|A|}$  смежных класса вида  $V_A + b := \{z + b \mid z \in V_A\}$ , где фиксировано  $b \in \mathbb{F}_2^m$  [доказательство далее]. Тогда для любого  $A \subseteq \{1, \dots, m\}$  и  $b \in \mathbb{F}_2^m$  мы имеем

$$\sum_{z \in (V_A + b)} \text{Eval}_z(x_A) = 1,$$

а для любых  $A \not\subseteq B$ ,

$$\sum_{z \in (V_A + b)} \text{Eval}_z(x_B) = 0$$

Эти две суммы над  $\mathbb{F}_2$  [т.е.  $1 + 1 + 1 = 1$ ]. Первая сумма вытекает из того, что  $\text{Eval}_z(x_A) = 1$  если и только если  $z_i = 1 \forall i \in A$ , причём существует только один такой  $z \in (V_A + b)$  [доказательство далее]. Для доказательства второй суммы, нужно заметить, что поскольку  $A \not\subseteq B$ , то  $\exists i \in A \setminus B$ , а значит бит  $z_i$  не влияет на значение  $\text{Eval}_z(x_B)$ . Отсюда,  $\text{Eval}_{z, z_i=0}(x_B) = \text{Eval}_{z, z_i=1}(x_B)$ , а значит все единицы в этой сумме взаимоуничтожатся.

Предположим, что битовый вектор  $y = (y_z \mid z \in \mathbb{F}_2^m)$  — зашумлённая версия кодового слова  $\text{Eval}(f) \in \text{RM}(r, m)$ , такого что  $y$  и  $\text{Eval}(f)$  отличаются не более чем в  $2^{m-r-1}$  позициях. Алгоритм Рида позволяет восстановить исходное кодовое слово из  $y$ , извлекая коэффициенты полинома  $f$ . Поскольку  $\deg f \leq r$ , мы всегда это можем записать  $f = \sum_{A \subseteq \{1, \dots, m\}, |A| \leq r} u_A x_A$ , где  $u_A$  — коэффициенты соответствующих мономов. Алгоритм Рида сначала извлекает все коэффициенты при мономх степени  $r$ , затем при степени  $r - 1$ , и так далее пока не найдёт их все.

Чтобы восстановить коэффициент  $u_A$  при  $|A| = r$  [при мономе степени  $r$ ], алгоритм Рида вычисляет сумму  $\sum_{z \in (V_A+b)} y_z$  для каждого из  $2^{m-r}$  смежных классов подпространства  $V_A$ , а затем выбирает коэффициент большинством голосов<sup>1</sup> среди этих  $2^{m-r}$  сумм. Если там больше единиц, чем нулей, то восстанавливаем  $u_A = 1$ , иначе  $u_A = 0$ . Заметим, что если  $y = \text{Eval}(f)$ , т.е. ошибки нет, то:

$$\sum_{s \in (V_A+b)} y_z = \sum_{s \in (V_A+b)} \text{Eval}_z \left( \sum_{\substack{B \subseteq \{1, \dots, m\} \\ |B| \leq r}} u_B x_B \right) = \sum_{\substack{B \subseteq \{1, \dots, m\} \\ |B| \leq r}} u_B \sum_{s \in (V_A+b)} \text{Eval}_z(x_B).$$

Из полученных ранее равенств и при условии, что  $B \subseteq \{1, \dots, m\}$  и  $|B| \leq r = |A|$ , получаем  $\sum_{z \in (V_A+b)} \text{Eval}_z(x_B) = 1$  тогда и только тогда, когда  $B = A$  [из равенства:  $A \subseteq B$ , из ограничения:  $|B| \leq |A|$ ]. Отсюда  $\sum_{z \in (V_A+b)} y_z = u_A$  для всех  $2^{m-r}$  смежных классов вида  $V_A + b$  если  $y = \text{Eval}(f)$ . Поскольку мы допустили, что  $y$  и  $\text{Eval}(f)$  отличаются не более чем в  $2^{m-r-1}$  позициях, есть меньше чем  $2^{m-r-1}$  смежных классов, в которых  $\sum_{z \in (V_A+b)} y_z \neq u_A$ . После голосования большинством среди этих  $2^{m-r}$  сумм, мы найдём правильное значение  $u_A$ .

После вычисления всех коэффициентов при мономах степени  $r$ , мы можем посчитать:

$$y' = y - \text{Eval} \left( \sum_{\substack{B \subseteq \{1, \dots, m\} \\ |B|=r}} u_B x_B \right).$$

Это зашумленная версия кодового слова  $\text{Eval}(f - \sum_{B \subseteq \{1, \dots, m\}, |B|=r} u_B x_B) \in \text{RM}(r-1, m)$ , и количество ошибок в  $y'$  меньше чем  $2^{m-r-1}$  из предположения. Тогда мы можем аналогичным образом восстановить все коэффициенты при мономах степени  $r-1$  используя  $y'$ . Повторять эту процедуру пока не будут восстановлены все коэффициенты  $f$ .

**Теорема.** При декодировании кода  $\text{RM}(r, m)$  для фиксированного  $r$  и растущего  $m$ , алгоритм Рида корректно устраняет любую ошибку с весом Хэмминга не больше  $2^{m-r-1}$  за  $O(n \log^r n)$  по времени, где  $n = 2^m$  — длина кода.

[в источнике она без доказательства, но вы можете прочесть алгоритм ниже и попытаться доказать это самостоятельно]

<sup>1</sup>В оригинале — «performs a majority vote»; я не смог придумать лучшего перевода.

---

**Algorithm 1:** Reed's algorithm for decoding  $\text{RM}(r, m)$ 

---

**Data:** Parameters  $r$  and  $m$  of the RM code, and a binary vector

$y = (y_z \mid z \in \mathbb{F}_2^m)$  of length  $n = 2^m$

**Result:** A codeword  $c \in \text{RM}(r, m)$

$t \leftarrow r$

**while**  $t \geq 0$  **do**

**foreach** subset  $A \subseteq \{1, \dots, m\}$  with  $|A| = t$  **do**

        Calculate  $\sum_{z \in (V_A + b)} y_z$  for all the  $2^{m-t}$  cosets of  $V_A$

$\text{num1} \leftarrow$  number of cosets  $(V_A + b)$  such that  $\sum_{z \in (V_A + b)} y_z = 1$

$u_A \leftarrow \mathbf{1} [\text{num1} \geq 2^{m-t-1}]$

**end**

$y \leftarrow y - \text{Eval} \left( \sum_{A \subseteq \{1, \dots, m\}, |A|=t} u_A x_A \right)$

$t \leftarrow t - 1$

**end**

$c \leftarrow \text{Eval} \left( \sum_{A \subseteq \{1, \dots, m\}, |A| \leq r} u_A x_A \right)$

**return**  $c$

---

Подсказка: «coset» — смежный класс.

В оригинале  $\mathbf{1}[\cdot]$  описана как «indicator function» (характеристическая функция), но для меня это несёт мало смысла в этом контексте. Впрочем, из доказательства понятно, что здесь должно иметься ввиду:

$$\mathbf{1} [\text{num1} \geq 2^{m-t-1}] = \begin{cases} 1, & \text{num1} \geq 2^{m-t-1} \\ 0, & \text{num1} < 2^{m-t-1} \end{cases}$$

## A.1 Дополнительные доказательства

Далее я подробно доказываю некоторые утверждения, которые не были мне совершенно очевидны, и которые я не смог доказать в четыре слова чтобы включить в основной текст.

**Лемма.** Для подпространства  $V_A$  (размерности  $|A|$  в пространстве  $\mathbb{F}_2^m$ ) существует  $2^{m-|A|}$  смежных класса вида  $V_A + b := \{z + b \mid z \in V_A\}$ , где фиксировано  $b \in \mathbb{F}_2^m$ .

*Доказательство.* Из теоремы Лагранжа, известно что  $|G| = |H| \cdot [G : H]$ , где  $H \subseteq G$ , а  $[G : H]$  — число различных смежных классов. В нашем случае,  $H = V_A, G = \mathbb{F}_2^m$ . Тогда  $|V_A| = 2^{\dim V_A} = 2^{|A|}$ . Таким образом получаем:

$$[G : H] = \frac{|G|}{|H|} = \frac{|\mathbb{F}_2^m|}{|V_A|} = \frac{2^m}{2^{|A|}} = 2^{m-|A|} \quad \square$$

**Лемма.**  $\text{Eval}_z(x_A) = 1$  если и только если  $z_i = 1 \forall i \in A$ , причём существует только один такой  $z \in (V_A + b)$ .

*Доказательство.* Во-первых,  $\text{Eval}_z(x_A) = \text{Eval}_z(x_{A_1}x_{A_2}\dots x_{A_k})$  по определению  $x_A$ . Конечно же, оно будет верно если и только если  $x_{A_1} = x_{A_2} = \dots = x_{A_k} = 1$ . Другими словами,  $\forall i \in A \quad z_i = 1$ , если подставить значения вектор  $z$  на место переменных  $x$ . Таким образом, первая часть доказана.

Напомним определение  $V_A$ :

$$V_A = \{z \in \mathbb{F}_2^m \mid z_i = 0 \forall i \notin A\}$$

Теперь докажем существование вектора. Пусть искомый вектор существует и равен  $z = v + b, v \in V_A$ . Требуется, чтобы  $z_i = 1 \forall i \in A$ . Т.е.  $v_i + b_i = 1$ , а значит  $v_i = 1 - b_i$  (при  $i \in A$ , конечно). Такой  $v$  действительно существует в подпространстве  $V_A$ , потому что определение никак не ограничивает элементы  $v_i, i \in A$ .

Единственность следует из того, что все остальные элементы  $v$  обязательно обнуляются по определению  $V_A$  ( $v_i = 0$ , если  $i \notin A$ ). Теперь можно сказать, что  $v_i = \begin{cases} 1 + b_i, & i \in A \\ 0, & i \notin A \end{cases}$  и никак иначе, из чего получаем единственность искомого  $z = v + b$ .  $\square$

**Лемма.** Размерность  $V_A$  равна  $|A|$ .

*Доказательство.* Это почти очевидное утверждение. Если рассмотреть каждый из векторов в  $V_A$ , то у него могут меняться только те координаты, которые не обнулены, и их ровно  $|A|$ . Получается по одному базисному вектору на каждый элемент из  $|A|$ .  $\square$

Следующая теорема необходима для эффективной реализации алгоритма Рида на нормальном языке программирования.

**Теорема.** Пусть  $\bar{A} = \{1, \dots, m\} \setminus A$ . Для фиксированного  $A$ , множество смежных классов  $\{V_A + b \mid b \in V_{\bar{A}}\}$  будет содержать их все, причём все различны.

*Доказательство.* Здесь используются верхние индексы, никакого возведения в степень.

Сначала докажем, что все эти смежные классы различны. Рассмотрим любые два:  $(V_A + b^1)$  и  $(V_A + b^2)$ , где  $b^1, b^2 \in V_{\bar{A}}$  и  $b^1 \neq b^2$ . Можно сказать, что векторы  $b^1$  и  $b^2$  отличаются хотя бы в одном бите, назовём его  $i$ -ым. Причём  $i \in \bar{A}$ , поскольку все другие биты в  $V_{\bar{A}}$  обнулены. Покажем, что

любые векторы  $x \in (V_A + b^1)$  и  $y \in (V_A + b^2)$  тоже будут отличаться в  $i$ -ом бите.

$$\begin{array}{lll} x = v^1 + b^1 & y = v^2 + b^2 & b^1 \neq b^2 \\ x_i = v_i^1 + b_i^1 & y_i = v_i^2 + b_i^2 & b_i^1 \neq b_i^2 \end{array}$$

Заметим, что  $v_i^1 = v_i^2 = 0$ , поскольку  $v_1, v_2 \in V_A$ , но  $i \notin A$ . Получается, что  $x_i = 0 + b_i^1$  и  $y_i = 0 + b_i^2$ , причём  $b_i^1 \neq b_i^2$ . Таким образом  $x \neq y$  для любых  $x \in (V_A + b^1), y \in (V_A + b^2)$ .

Теперь докажем, что мы перечислили все смежные классы. Как доказано ранее, их всего  $2^{m-|A|}$ . С другой стороны,  $|V_{\bar{A}}| = 2^{|\bar{A}|} = 2^{m-|A|}$ . Поскольку все элементы множества различны, то оно содержит все смежные классы.  $\square$

## A.2 Реализация алгоритма

Замечание: этот код действительно реализует алгоритм Рида, но он использует не те же соглашения, что даны в презентации (можете проверить на примерах). Если вы собираетесь использовать его в ДЗ, убедитесь описать, чему соответствуют биты и как у вас выглядит таблица истинности.

```
import itertools, math
_all_ = ['encode', 'decode', 'code_info']

# Возвращает длину сообщения  $k = \sum_{i=0}^r C_m^i$ , корректирующую
# способность  $t = 2^{m-r-1} - 1$  и длину кода  $n = 2^m$ 
def code_info(r, m):
    ...

    >>> code_info(1, 2)
    {'k': 3, 'd': 2, 't': 0, 'n': 4}
    >>> code_info(2, 4)
    {'k': 11, 'd': 4, 't': 1, 'n': 16}
    ...

    return {'k': sum(math.comb(m, i) for i in range(0, r+1)),
            'd': 2**(m-r), 't': 2**(m-r-1) - 1, 'n': 2**m}

# Возвращает  $\{A \subseteq \{0, \dots, m-1\} : |A| = t\}$ 
def subsets(m, t):
    ...

    >>> [bin(i) for i in subsets(3, 1)]
    ['0b1', '0b10', '0b100']
    >>> [bin(i) for i in subsets(3, 2)]
    ['0b11', '0b101', '0b110']
    >>> [bin(i) for i in subsets(3, 3)]
    ['0b111']
    ...

    for i in itertools.combinations(range(0, m), t):
        # i содержит выбранные биты, ровно t штук.
        yield sum(1 << j for j in i)

# Возвращает  $\{A \subseteq \{0, \dots, m-1\} : |A| \leq r\}$ 
def all_subsets(r, m):
    ...

    >>> [bin(i) for i in all_subsets(3, 3)]
    ['0b0', '0b1', '0b10', '0b100', '0b11', '0b101', '0b110',
    ↪ '0b111']
    ...

    return itertools.chain.from_iterable(
        subsets(m, t) for t in range(0, r+1))

# Вычисляет  $\text{Eval}(\sum_{A \in \mathcal{A}_s} u_A x_A)$ 
def evaluate(get_As, m, u):
    ...

    f(x0, x1, x2) = x0x2 будет иметь вектор значений 00000101
    >>> bin(evaluate(lambda: [0b101], 3, {0b101: 1}))
    '0b101'

    f(x0, x1) = 1 + x, будет иметь вектор значений 1100
    >>> bin(evaluate(lambda: [0b00, 0b10], 2, {0:1, 2:1}))
    '0b1100'
    ...

    result = 0
    for z in range(2**m):
        summ = 0
        for A in get_As():
            # Вычисляю  $x_A = x_{A_1}x_{A_2}\dots x_{A_k}$ , подставляя в
            ↪ качестве  $x_i = z_i$ 
            # Это равно единице тогда и только тогда, когда
            ↪ все биты из A также стоят в z.
            xProduct = 1 if (z & A) == A else 0
            summ += u[A] * xProduct
        result = (result << 1) | (summ % 2)
    return result

# Кодировать сообщение u при помощи кода RM(r, m)
def encode(r, m, msg):
    ...

    >>> bin(encode(2, 4, [1,1,1,0,0,1,1,0,1,0,0]))
    '0b1000111010001110'
    ...

    u = [None] * (2**m)
    for i, A in zip(msg, all_subsets(r, m), strict=True):
        u[A] = i
```

```
    return evaluate(lambda: all_subsets(r, m), m, u)

# Возвращает все векторы из подпространства  $V \subseteq \mathbb{F}_2^n$ ,
# если даны базисные векторы для V.
def _subspace(basis):
    for i in range(2**len(basis)):
        result = 0
        for mask in basis:
            if (i & 1) == 1:
                result |= mask
            i >>= 1
        yield result

# Возвращает все векторы из подпространства  $V_A \subseteq \mathbb{F}_2^n$ 
def subspaceV_A(m, A):
    ...

    >>> [bin(i) for i in subspaceV_A(3, 0b10)]
    ['0b0', '0b10']
    >>> [bin(i) for i in subspaceV_A(3, 0b101)]
    ['0b0', '0b1', '0b100', '0b101']
    ...

    basis = []
    mask = 1
    while mask <= A:
        if (A & mask) != 0:
            basis.append(mask)
        mask <<= 1
    return _subspace(basis)

# Возвращает все векторы из подпространства  $V_{\bar{A}}$ 
def subspaceV_minusA(m, A):
    ...

    >>> [bin(i) for i in subspaceV_minusA(3, 0b10)]
    ['0b0', '0b1', '0b100', '0b101']
    >>> [bin(i) for i in subspaceV_minusA(3, 0b101)]
    ['0b0', '0b10']
    ...

    basis = []
    for i in range(m):
        mask = 1 << i
        if (A & mask) == 0:
            basis.append(mask)
    return _subspace(basis)

# Алгоритм Рида по псевдокоду, который был в статье
def decode(r, m, y):
    ...

    Попробуйте изменить здесь бит и запустить тесты снова!
    >>> decode(2, 4, 0b1000111010001110)
    ('0b1000111010001110', [1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0])
    ...

    u = [None] * (2**m)
    t = r
    while t >= 0:
        for A in subsets(m, t):
            num1 = 0
            for b in subspaceV_minusA(m, A):
                coset = (y + b for v in subspaceV_A(m, A))

                #  $s = \sum_{z \in (V_A + b)} y_z$ 
                s = sum((y >> z) & 1 for z in coset)
                if (s % 2) == 1:
                    num1 += 1

                u[A] = int(num1 >= 2**(m - t - 1))
            y = y ^ evaluate(lambda: subsets(m, t), m, u)
            t = t - 1

    c = evaluate(lambda: all_subsets(r, m), m, u)
    msg = [u[A] for A in all_subsets(r, m)]
    return bin(c), msg

# «Тесты»:
import doctest; doctest.testmod()
# Try: `python -i ReedMuller.py`
```