# Код Рида-Маллера

Илья Коннов
Факультет компьютерных наук
Высшая Школа Экономики

13 февраля 2022 г.

Если вы смотрите презентацию, то на сером фоне справа иногда видны некоторые ценные комментарии, для которых поля слайда оказались слишком узки. Если вы читаете pdf-ку, то эти комментарии уже находятся в самом подходящем для них месте в тексте (а в внешних полях видны заголовки слайдов). Если вы смотрите мой доклад и видите этот текст, то что-то пошло серьёзно не так. Да, у этого одного файла есть три разные версии.

По любым вопросам: ReedMuller@sldr.xyz или t.me/iliago или vk.com/iliago.

## Содержание

**1 Введение** **2**

**2 Кодирование** **3**

**3 Свойства и параметры кода** **4**
3.1 Конструкция Плоткина . . . . . 6
3.2 Минимальное расстояние . . . . . 7

**4 Декодирование** **8**
4.0.1 Пара слов о синдромах . . . . . 8
4.1 Алгоритм Рида . . . . . 9

**5 Домашнее задание** **10**

**6 Источники** **10**

**A Reed's Algorithm: Unique decoding up to half the code distance** **11**
A.1 Дополнительные доказательства . . . . . 13
A.2 Реализация алгоритма . . . . . 16

# 1 Введение

***Введение***

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

Обозначаются как $\mathrm{RM}(r, m)$, где $r$ — ранг, а $2^m$ — длина кода. Кодирует 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. $\mathbb{Z}_2$.

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

***Булевы функции и многочлен Жегалкина***

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 |

И при помощи многочлена Жегалкина:

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

***Многочлены Жегалкина***

В общем случае, многочлены будут иметь следующий вид:

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

Например, для $m = 2$: $f(x_1, x_2) = c_1 \cdot x_1 x_2 + c_2 \cdot x_1 + c_3 \cdot x_2 + c_4 \cdot 1$

Всего $n = 2^m$ коэффициентов для описания каждой функции.

***Функции небольшой степени***

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

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

Каждую можно записать следующим образом:

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

В каждом произведении используется не больше $r$ переменных.

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

Сколько тогда всего коэффициентов используется?

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

Если говорить несколько проще, то для составления многочленов мы сложим сначала одночлены $(x+y+z+...)$, затем произведения одночленов $(xy+yz+xz+...)$ и т.д. вплоть до $r$ множителей (поскольку мы работаем в поле $\mathbb{Z}_2$, 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 до всех $r$

# 2 Кодирование

***Идея кодирования***

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 переменных.
Их $2^m$, поскольку рассматриваем многочлены только над $\mathbb{Z}_2$ 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 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

$$\Longrightarrow \text{Eval}(f) = (1 \quad 0 \quad 0 \quad 0)$$

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.

***Пример***

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

- $r = 1$ (степень многочлена), $m = 2$ (переменных).
  Это $\text{RM}(1, 2)$.
- Тогда наш многочлен: $f(x, y) = c_1x + c_2y + c_3$.
- Сообщение: `101`, тогда $f(x, y) = x + 0 + 1$.
- Подставим всевозможные комбинации:

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

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

***Декодирование когда потерь нет***

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

- Мы получили код: `1100`
- Представим таблицу истинности.

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

- Подстановками в $f(x,y) = c_1x + c_2y +$ получим СЛАУ.

$$\begin{cases} c_3 & & & & c_3 & = & 1 \\ & & c_2 & + & c_3 & = & 1 \\ c_1 & + & & & c_3 & = & 0 \\ c_1 & + & c_2 & + & c_3 & = & 0 \end{cases}$$

- $c_1 = 1, c_2 = 0, c_3 = 1$, исходное сообщение: `101`.

***Коды 0-го порядка***

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

Для случая $\mathrm{RM}(0, m)$ 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 $\deg f \leq 0$.

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

Таблица истинности:

| | $x_1$ | $x_2$ | ... | $x_m$ | $f(x_1, ..., x_m)$ | $g(x_1, ..., x_m)$ |
|---|---|---|---|---|---|---|
| $2^m$ | 0 | 0 | $\cdots$ | 0 | 0 | 1 |
| | 0 | 0 | $\cdots$ | 1 | 0 | 1 |
| | | | $\ddots$ | | | |
| | 1 | 1 | $\cdots$ | 1 | 0 | 1 |

Здесь число строк, как и в любой другой таблице истинности, равно $2^m$, а колонки с 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 $\underbrace{\texttt{00...0}}_{2^m}$
- Сообщение 1 даст код $\underbrace{\texttt{11...1}}_{2^m}$

# 3 Свойства и параметры 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 ($\mathbb{Z}_2^k$) в пространство слов ($\mathbb{Z}_2^m$).

Для этого необходимо немного формализовать всё описанное раньше.

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

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

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 $p_{(x \oplus y)} = p_x + p_y$.

Напомню, 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 $p_{(x+y)} = p_x + p_y$. 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 растояние будет равно минимальному весу Хемминга среди всех кодов. Вес Хэмминга 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

***Конструкция Плоткина: многочлены***

Хотим понять как выглядят кодовые слова.

- Код — таблица истинности функции $f(x_1, ..., x_m) \in \mathrm{RM}(r, m)$, причём $\deg f \leq r$. Порядок очевидно не больше $r$, потому что это условие для включения в пространство кодов $\mathrm{RM}(r, m)$.
- Разделим функцию по $x_1$: $f(x_1, ..., x_m) = g(x_2, ..., x_m) + x_1 h(x_2, ..., x_m)$. 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: таблица истинности***

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

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

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

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

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

***Конструкция Плоткина: вывод***

Теперь собираем всё это в одно важное утверждение.

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

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

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

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

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

Тогда: $c = \mathrm{Eval}(f) \in \mathrm{RM}(r, m)$ (т.к. $\deg f \leq r$)
$u = \mathrm{Eval}(g) \in \mathrm{RM}(r, m-1)$ (т.к. $\deg g \leq r$)
$v = \mathrm{Eval}(h) \in \mathrm{RM}(r-1, m-1)$ (т.к. $\deg h \leq r-1$)

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

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

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

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

## 3.2 Минимальное расстояние

***Минимальное расстояние***

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

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

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

**База:** $\mathrm{RM}(0, m)$ — единственный бит потворён $2^m$ раз. Очевидно, $w(\underbrace{11\ldots1}_{2^m}) = 2^m = 2^{m-0} \geq 2^{m-r}$.

Случай $\mathrm{RM}(0, m)$ 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 (соответсвует функции $f(x_1, ..., x_m) = 0$ или $f(x_1, ..., x_m) = 1$) и повторяем его $2^m$ раз (в таблице истинности).

Замечу, что не рассматриваю второй случай $w(\mathtt{00\ldots0})$, поскольку он нам не нужен для расчёта минимального расстояния. Вариант с нулевым вектором явно выкидывается, см. определение $d$ выше.

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

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

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

Теперь немного объяснений.

Переход (1): $w((x \mid y)) = w(x) + w(y)$. Вес это всего лишь число ненулевых элементов, поэтому нет разницы как мы будем группировать части вектора.

Переход (2): $w(u \oplus v) \geq w(v) - w(u)$. 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 (IH): предположение индукции в чистом виде.

***Свойства и параметры***

Теперь можно подвести итоги исследования свойств.

Для бинарного кода $\mathrm{RM}(r, m)$:

- $r \leq m$

- 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 $t = \left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{2^{m-r}}{2} - \frac{1}{2} \right\rfloor = \left\lfloor 2^{m-r-1} - 0.5 \right\rfloor = 2^{m-r-1} - 1$
- 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 $\mathrm{RM}(m-r-1, m)$, но это я это доказывать не собираюсь. Но его можно найти в «Reed-Muller Codes: Theory and Algorithms», раздел Duality.

# 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 ввиду, но о них было рассказано и без меня, так что я их пропущу.

### 4.0.1 Пара слов о синдромах

***Синдромы и как их использовать***

Я не стал включать это в презентацию, но вообще-то говоря метод полезный, так что пусть будет здесь.

Пусть у нас в полученном сообщении $r$ есть ошибка $e$. Тогда $r = v+e$, где $v$ — кодовое слово, которое крайне легко можно декодировать. Получается, что $s = rH^T = (v+e)H^T = vH^T + eH^T = eH^T$, поскольку $vH^T = 0$ (есть такое свойство). Мы можем перебрать всевозможные ошибки $(e)$, для каждой посчитать синдром и записать всё это в таблицу. Тогда чтобы восстановить сообщение, нужно посчитать синдром, по таблице найти ошибку и исправить её.

Источник: `https://ru.wikipedia.org/wiki/Линейный_код`

## 4.1 Алгоритм Рида

***Пример***

Теперь начинаем нормальный алгоритм декодирования, придуманный Ридом (тем самым). Именно из-за алгоритма декодирования Рида включили в соавторы кода Рида-Маллера.

Ранее: `101` кодируется как `1100` при помощи $\mathrm{RM}(1,2)$ (см. самый первый пример).

- $t = 1$

  –

- $t = 0$

  –

# 5 Домашнее задание

TODO

# 6 Источники

Бонусный раздел, который не включён в основную презентацию, но может быть очень полезен.

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` — кратко, чётко, понятно, но не описано декодирование.
4. `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 $\mathrm{RM}(r,m)$. Он исправляет любые ошибки, вес которых не превышает $2^{m-r-1}$, половину минимального расстояния кода.

Для подмножества $A \subseteq \{1...m\}$ определим обозначим моном $x_A = \prod_{i\in A} x_i$, где $x_i$ — аргументы булевой функции [напр., $x_{\{1,2\}} = x_1x_2$]. 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 $b \in F_2^m$ [доказательство далее]. Тогда для любого $A \subseteq \{1, ..., m\}$ и $b \in \mathbb{F}_2^m$ мы имеем

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

а для любых $A \not\subseteq B$,

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

Эти две суммы над $\mathbb{F}_2$ [т.е. $1 + 1 + 1 = 1$]. Первая сумма вытекает из того, что $\mathrm{Eval}_z(x_A) = 1$ 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 $\mathrm{Eval}_z(x_B)$. Отсюда, $\mathrm{Eval}_{z,z_i=0}(x_B) = \mathrm{Eval}_{z,z_i=1}(x_B)$, а значит все единички в этой сумме взаимоуничтожатся.

Предположим, что битовый вектор $y = (y_z \mid z \in \mathbb{F}_2^m)$ — зашумлённая версия кодового слова $\mathrm{Eval}(f) \in \mathrm{RM}(r,m)$, такого что $y$ и $\mathrm{Eval}(f)$ отличаются не более чем в $2^{m-r-1}$ позициях. Алгоритм Рида позволяет восстановить исходное кодовое слово из $y$, извлекая коэффициенты полинома $f$. Поскольку $\deg f \le r$, мы всегда это можем записать $f = \sum_{A\subseteq\{1,...,m\},|A|\le r} u_A x_A$, где $u_A$ — коэффициенты соответствующих мономов. Алгоритм Рида сначала извлекает все коэффициенты при монмах 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[1] среди этих $2^{m-r}$ сумм. Если там больше единиц, чем нулей, то восстаналиваем $u_A = 1$, иначе $u_A = 0$. Заметим, что если $y = \mathrm{Eval}(f)$, т.е. ошибки нет, то:

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

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

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

Это зашумленная версия кодового слова $\mathrm{Eval}(f - \sum_{B\subseteq\{1,\dots,m\},|B|=r} u_B x_B) \in \mathrm{RM}(r-1, m)$, и количество оошибок 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* $\mathrm{RM}(r, m)$ *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 прочитать алгоритм ниже и попытаться доказать это самостоятельно]

[1] В оригинале — «performs a majority vote»; я не смог придумать лучшего перевода.

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

**Data:** Parameters $r$ and $m$ of the RM code, and a binary vector $y = (y_z \mid z \in F_2^m)$ of length $n = 2^m$

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

$t \leftarrow r$

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

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

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

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

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

  **end**

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

  $t \leftarrow t - 1$

**end**

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

**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:

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

## 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 $b \in F_2^m$.*

*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:

$$[G : H] = \frac{|G|}{|H|} = \frac{|F_2^m|}{|V_A|} = \frac{2^m}{2^{|A|}} = 2^{m-|A|} \qquad \square$$

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

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

Напомню 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 теорма необходима для эффективной реализации алгоритма Рида на нормальном языке программрования.

**Теорема.** *Пусть* $\bar{A} = \{1, ..., m\} \setminus A$. *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, а значит $b^1_i = b^2_i = 0$. Покажем, что любые векторы $x \in (V_A + b^1)$ и $y \in (V_A + b^2)$ тоже будут отличаться в $i$-ом бите.

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

Заметим, что $v^1_i = v^2_i = 0$, поскольку $v_1, v_2 \in V_A$, но $i \notin A$. Получается, что $x_i = 0 + b^1_i$ и $y_i = 0 + b^2_i$, причём $b^1_i \neq b^2_i$. 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

Битовые векторы храним как int. Нумеруются справа налево, нулевой элемент на самой правой позиции int. Тогда: $u + v =$ `u ^ v` и $v_i =$ `(v >> i) & 1` (нумерация здесь с нуля, $i \in \{0, ..., n-1\}$).

Множество $A$ также храним при помощи одного int. Если $i \in A$, то $A_i = 1$.

```python
import itertools, math

# Возвращает {A ⊆ {0,...,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 ⊆ {0,...,m−1} : |A| ≤ r}
def all_subsets(m, r):
    '''
    >>> [bin(i) for i in all_subsets(3, 3)]
    ['0b1', '0b10', '0b100', '0b11', '0b101', '0b110',
    ↪ '0b111']
    '''
    return itertools.chain.from_iterable(
        subsets(m, t) for t in range(1, r+1)
    )

# Возвращает все векторы из подпространства V ⊆ 𝔽₂,
# 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 ⊆ 𝔽₂^m
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_Ā
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)

# Возвращает все смежные классы вида (V_A + b)
def cosets(m, A):
    for b in subspaceV_minusA(m, A):
        yield (v + b for v in subspaceV_A(m, A))

# Вычисляет Eval(∑_{A∈As} u_A x_A)
def evaluate(As, m, u):
    '''
    f(x₀,x₁,,x₂) = x₀x₂ будет иметь вектор значений 00000101
    >>> bin(evaluate([0b101], 3, {0b101: 1}))
    '0b101'

    f(x₀,x₁) = 1 будет иметь вектор значений 1111
    >>> bin(evaluate([0], 2, {0: 1}))
    '0b1111'
    '''
    result = 0
    for z in range(2**m):
        val = 0
        for A in As:
            # Вычисляю x_A = x_{A_1} x_{A_2} ... 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

            val += u[A] * xProduct
        val %= 2
        result = (result << 1) | val
    return result

# Алгоритм Рида по псевдокоду, который был ранее
def Reed(r, m, y):
    # k = ∑_{i=0}^{r} C_m^r. TODO: хранить в int, а не массиве
    u = [None] * sum(math.comb(m,i) for i in range(r+1))
    t = r
    while t ⩾ 0:
        for A in subsets(m, t):
            num1 = 0
            for coset in cosets(m, A):
                # s = ∑_{z∈coset} 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(subsets(m, t), m, u)
        t = t - 1
    c = evaluate(all_subsets(m, r), m, u)
    return (c, u)

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