Теоркод без хуйни. Гайд на J

1 Введение в теорию чисел

В декодере нам предстоит жить в поле Галуа $GF(2^m)$. С полем $GF(2)$ вы скорее всего уже знакомы, там у нас просто есть два числа: $0, 1$, сложение работает как xor, а умножение как and.

Здесь получается более общая ситуация: в поле $GF(2^m)$ есть 2^m примитивных элементов: $\alpha^0, \alpha^1, ..., \alpha^{2^m-1}$. Это те самые коэффициенты которые стоят в многочленах, т.е какой-то многочлен в поле $GF(2^3)$ мог бы выглядеть как $x^3 + \alpha^7 x^2 + \alpha^1 x + \alpha^5$. (Напоминаю что сложение и умножение тут это одно и то же).

Что же такое эти альфы? Они порождаются порождающим многочленом p, который прибит к конкретному m. Про него можно сказать много интересного, но нам он дан в условии чтобы мы его сами не считали. Вооружившись им можно найти эти альфы следующим образом:

$$\alpha^0 = 1$$

$$\alpha^i = (\alpha^{i-1} \cdot x) \, mod \, p$$

Вычисление по модулю работает как обычное деление столбиком в GF(2), больше можно прочитать тут:
`https://en.wikipedia.org/wiki/Finite_field_arithmetic`
Там же есть и пример кода который позволяет делать эти вычисления по модулю. Обратите внимание что x который используется при подсчете этих альф не то же самое что x который в многочлене. На (рис. 1) - пример построения поля $GF(2^4)$. По сути, для того чтобы складывать и умножать эти альфы, нужно складывать (ксорить) битовые векторы в 3 колонке и умножать (столбиком) по модулю порождающего многочлена (все есть на википедии).

Таблица 6.1. Поле $GF(2^4)$, порожденное многочленом $p(x) = 1 + x + x^4$

Степень	Многочлен	Последовательность
$-\infty$	0	0000
0	1	0001
1	x	0010
2	x^2	0100
3	x^3	1000
4	$1 + x$	0011
5	$x + x^2$	0110
6	$x^2 + x^3$	1100
7	$1 + x + x^3$	1011
8	$1 + x^2$	0101
9	$x + x^3$	1010
10	$1 + x + x^2$	0111
11	$x + x^2 + x^3$	1110
12	$1 + x + x^2 + x^3$	1111
13	$1 + x^2 + x^3$	1101
14	$1 + x^3$	1001

Рис. 1: Построение GF(2^4)

2 Кодирование

Теперь можно пойти кодировать. Первое что для этого нужно - найти порождающий многочлен. Все просто (нет). Ищется он по следующей формуле, но все по порядку:

Определение 6.1. *Циклический код длины n над $GF(q)$ называется кодом БЧХ с конструктивным расстоянием d, если для некоторого $b \geq 0$ порождающий многочлен кода равен*

$$g(x) = HOK\left\{M_i(x), i = b, b+1, \ldots, b+d-2\right\}. \tag{6.11}$$

Рис. 2: Порождающий многочлен

b можно брать любым, но удобно брать $b = 1$, а d – конструктивное расстояние, нам дано. Осталось понять что такое $M_i(x)$, а это - минимальные многочлены. Искать их можно вот так:

$$M_i(x) = \prod_{j \in C_i} (x - \alpha^j)$$

C_i здесь - i-ый циклотомический класс. Что же это такое? Строится оно примерно так: берем какое-то число от 0 до $2^m - 2$, умножаем на 2 и берем по модулю $2^m - 1$ пока можем. Все числа разделятся на какие-то классы. Проще всего понять на примере (рис. 3):

Пример 6.2. Циклотомическими классами по модулю 15 являются множества

$$C_0 = \{0\};$$
$$C_1 = \{1, 2, 4, 8\};$$
$$C_3 = \{3, 6, 12, 9\};$$
$$C_5 = \{5, 10\};$$
$$C_7 = \{7, 14, 13, 11\}.$$

Рис. 3: Циклотомические классы для m=4

Значит, чтобы найти i-ый минимальный многочлен, нужно вычислить произведение, написанное выше для всех элементов α^j принадлежащему i-ому циклотомическому классу. (Нумеровать классы будем по минимальному элементу в нем для удобства). Реализовывать это не очень приятно, нужно реализовать по сути раскрывание скобок и преведение подобных. (Помните что альфа это числа, к иксам они не имеют отношение, операции над ними определены как я рассказывал в секции 1, и как по той ссылке в википедии). Есть теорема, которая говорит что в минимальном многочлене коэффициенты у вас будут либо 0 либо 1 после всех этих махинаций.

Но вернемся к построению порождающиего многочлена (рис. 2). Очевидно, для элементов одного циклотомического класса минимальные многочлены одинаковые. Более того, существует теорема, которая говорит, что для разных циклотомических классов, минимальные многочлены взаимнопросты, а значит наш НОК превращается в перемножение минимальных многочленов для классов от 1 до $d - 1$. (В примере выше с $n = 15$ и $d = 5$, мы бы перемножили M_1 и M_3).

Дело за малым - закодировать. Пусть $c(x)$ - искомый закодированный вектор, а $v(x)$ - то, что нас попросили закодировать, тогда:

$$c(x) = v(x) \cdot x^r + (v(x) \, mod \, g(x))$$

Где $r = n - k$

3 Декодирование

По сути, весь алгоритм есть в учебнике и на скрине ниже: В этом алгоритме

Алгоритм 7.2. Алгоритм Берлекэмпа–Месси

```
Вычисление коэффициентов многочлена локаторов ошибок
по алгоритму Берлекэмпа-Месси
Input: Синдромный многочлен S(x)
Output: Многочлен локаторов Λ(x)
1. Инициализация
L = 0; % Текущая длина регистра
Λ(x) = 1; % Многочлен локаторов
B(x) = 1; % Многочлен компенсации невязки
2. Основной цикл
for r = 1 to d − 1 do
    Δ = Σ_{j=0}^{L} Λ_j S_{r−j}; % Невязка
    B(x) = xB(x); % Сдвиг
    if Δ ≠ 0 then
        % ЛРОС модифицируется
        T(x) = Λ(x) − ΔB(x); % Вспомогательный многочлен
        if 2L ≤ r − 1 then
            % Длина увеличивается
            B(x) = Δ^{−1}Λ(x);
            L = r − L;
        end
        Λ(x) = T(x);
    end
end
3. Формирование результата
if deg Λ(x) == L then
    Многочлен локаторов = Λ(x);
end
else
    Число ошибок больше t;
end
```

Рис. 4: Месси занялся чем-то не тем

нужно быть очень аккуратно с индексами и выходами за границы массивов, но в целом он рабочий. За ним стоит очень много теории, с ней вы можете попытаться ознакомиться в учебнике, но вы ничего не поймете.

Нужно понять как по входному вектору получить синдромный многочлен и как по многочлену локаторов получить индексы, в которых произошла ошибка.

Про синдромы: всего их $d-1$ (от 1 до $d-1$). Считаются они подстановкой соответствующей альфы во входной многочлен (рис. 5). Считать это можно (и нужно) схемой Горнера. Можно и быстрее, но вряд ли нужно, зайдет и так.

Про места ошибок: проверить что ошибка произошла в i-ом символе

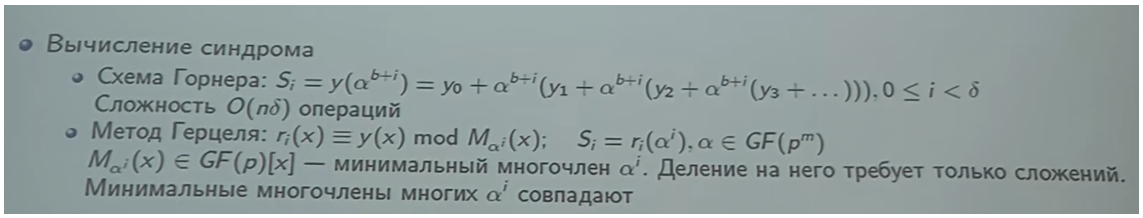

Рис. 5: Вычисление синдромов

очень просто, надо просто проверить условие:

$$\Lambda(\alpha^{-i}) = 0$$

4 Интересные факты

- В команаде $Simulate\,a\,b\,c$, b и c как в витерби, а a - вероятность с которой надо инвертировать бит в сгенерированном векторе.
- Говорят, можно словить ТЛ если каждый раз умножать альфы тем алгосом который я кидал выше. Избежать этого можно предподсчитав таблицу умножнния.