Московский государственный университет имени М. В. Ломоносова

Факультет Вычислительной математики и Кибернетики
Кафедра Математических Методов Прогнозирования

Практикум на ЭВМ.

Отчет по заданию №8:

Коды БЧХ.

Выполнил:
студент 3 курса 317 группы
Таскынов Ануар

Содержание.

1 Постановка задания.

В данном задании необходимо было реализовать два модуля. Модуль gf.py, в котором реализованы все стандартные действия над многочленами в поле $\mathbb{F}_2^q$ и модуль bch.py с реализацией БЧХ-кодов.

Также в модуль gf.py были добавлены такие функции как normal_poly и polyadd. Первая функция возвращает полином без "ведущих нулей"полинома, а вторая реализует сложение двух полиномов в поле $\mathbb{F}_2^q$.

Все эксперименты приведены в соответствующем Ipython Notebook.

2 БЧХ-код.

2.1 Кодирование.

Сообщение $u \in \{0,1\}^k$ можно представить в виде многочлена $u(x) = u_{n-1}x^{n-1} + \dots + u_1 x + u_0$.

Для того, чтобы построить (n,t) [1] БЧХ-код сначала нужно построить поле $\mathbb{F}_2^q$. К заданию прилагался список примитивных многочленов, из него выбирался многочлен $h(x)$ со степенью равной q. Порождающим многочленом для (n,t) БЧХ-кода является минимальный многочлен для набора $\alpha, \alpha^2, \dots, \alpha^{2t}$, где α - это стандартный примитивный элемент. Обозначим порождающий многочлен за $g(x)$. Тогда систематическое кодирование для многочлена $u(x)$ записывается в виде:

$$v(x) = x^m u(x) + mod(x^m u(x), g(x))$$

2.2 Декодирование.

Пусть $w(x)$ - принятое слово. Тогда его можно представить в виде $w(x) = v(x) + e(x)$, где $e(x) = x^{j_1} + \dots + x^{j_\nu}$ - *полином ошибок*, а $j_1, \dots, j_\nu$ - позиции, которых произошли ошибки. Пусть $s_i = w(\alpha^i)$, $i = 1, \dots, 2t$ [2]. Обозначим за $\Lambda(z)$ - *полином локаторов ошибок*:

$$\Lambda(z) = \prod_{i=1}^{\nu}(1 + \alpha^{j_i} z) = \Lambda_\nu z^\nu + \dots + \Lambda_1 z + 1.$$

Введем следующие обозначения:

$$b = \left[s_{\nu+1}, s_{\nu+2}, \dots, s_{2\nu}\right]^T$$

$$A = \begin{bmatrix} s_1 & s_2 & \dots & s_\nu \\ s_2 & s_3 & \dots & s_{\nu+1} \\ \dots & \dots & \dots & \dots \\ s_\nu & s_{\nu+1} & \dots & s_{2\nu-1} \end{bmatrix}$$

[1] $n = 2^q - 1$

[2] Синдромы

Тогда $\Lambda = \left[\Lambda_\nu, \ldots, \Lambda_1\right]^T$, будет удовлетворять СЛАУ: $A\Lambda = b$. Таким образом можно найти коэффициенты локатора ошибок.

Существуют два декодера:

1. PGZ.

2. Euclid.

PGZ-метод подразумевает под собой непосредственное решение СЛАУ. Если при данном ν СЛАУ не удается решить, то $\Lambda_\nu = 0$ и ν уменьшается на единицу. Иначе находятся все коэффициенты локатора ошибок и далее декодирование становится очевидным. Если ни на одной итерации не было найдено решения СЛАУ, то выдается отказ от декодирования. Также если синдромы $\hat{v}(x)$ не равны нулю, то выдается отказ.

Euclid-метод основан на расширенном алгоритме Евклида. Введем *синдромный многочлен* $S(z) = s_{2t}z^{2t} + s_{2t-1}z^{2t-1} + \cdots + s_1 z + 1$, где s_i - вычисленные ранее синдромы.

Алгоритм Евклида запускается для пар $(z^{2t+1}, S(z))$:

$$z^{2t+1}A(z) + S(z)\Lambda(z) = r(z)$$

.

Здесь $r(z)$ - некоторый многочлен, степень которого не превышает t. Таким образом находится $\Lambda(z)$ и коэффициенты локатора ошибок. Если количество корней Λ не совпадает с ν, то выдается отказ от декодировании.

3 Эксперименты и их результаты.

3.1 Скорость кода.

В этом эксперименте необходимо было найти зависимость между скоростью кода и максимальным числом исправляемых ошибок, t. Результаты приведены на Рис.

b

Как видно из графиков, скорость кода "ступенчато"приближается к асимптоте, равной 0. Скорость кода определяет избыточность информации, поэтому альтернативное t можно выбрать как $n/5$.

Рис. 1: Зависимость скорости кода от максимального количества исправляемых ошибок.

3.2 Минимальное расстояние.

Для линейного кода, каким является БЧХ-код расстояние можно найти как минимальный хэммингов вес. Соответствующая функция, которая реализует приведена в классе BCH модуля bch.py.

В подпункте нужно было привести пример БЧХ-кода, расстояние которого больше $2t+1$. Таким оказался БЧХ-код с $n = 15$, $t = 4$, расстояние которого равно 15.

3.3 Сравнение времени работы методов.

В этом пункте необходимо было сравнить скорость работы методов. В таблицах 1 и 2 приведены результаты эксперимента.

Эксперименты были запущены сначала на количестве ошибок $r = t$. Как видно из таблицы, в этом случае PGZ-метод работает быстрее, так как при первой же итерации находит решение СЛАУ. Также было запущено на количестве ошибок $r = 1$. В данном случае оба метода работают примерно одинаково.

n	t	Euclid, время	PGZ, time
7	1	0.580945	0.327488
15	3	2.067368	1.005466
31	4	3.535710	1.681343
63	5	5.545853	3.128824
63	8	10.280127	4.736844
127	8	12.705602	7.585843
127	11	20.582913	9.850387

Таблица 1: Сравнение скорости работы при $r = t$.

n	t	Euclid, время	PGZ, time
7	1	0.588281	0.338834
15	3	1.169207	0.909572
31	4	1.739895	1.578275
63	5	3.003512	2.793846
63	8	3.958691	3.926208
127	8	6.288400	6.462221
127	11	8.696548	8.459977

Таблица 2: Сравнение скорости работы при $r = 1$.

3.4 Сравнение методов на основе количества допущенных ошибок.

Необходимо было провести эксперименты для того, чтобы выявить долю правильно декодированных, неправильно декодированных сообщений и долю отказа от декодирования в зависимости от количества допущенных ошибок. Результаты приведены на таблице 3.

Эксперимент проводился следующим образом: для количества ошибок $1 <= r <= t$ генерировалось матрица U с сообщениями, количество сообщений указано в таблице. Для кажой такой матрицы считалось соответственно все эти три доли. Как видно из таблицы, код БЧХ в точности исправляет до t ошибок.

Также было запущено для $r = t + 1$ и, как видно, в этом случае код БЧХ дает отказ от декодирования.

Message count	n	t	Correct, r<=t	Incorrect, r<=t	Decode error, r<=t	Correct, r>t	Incorrect, r>t	Decode error, r>t
10	7	1	1.0	0.0	0.0	0.0	1.000	0.000
50	15	3	1.0	0.0	0.0	0.0	0.400	0.600
100	31	4	1.0	0.0	0.0	0.0	0.000	1.000
200	63	5	1.0	0.0	0.0	0.0	0.045	0.955
200	63	8	1.0	0.0	0.0	0.0	0.000	1.000
500	127	8	1.0	0.0	0.0	0.0	0.000	1.000
500	127	11	1.0	0.0	0.0	0.0	0.000	1.000

Таблица 3: Оценка доли правильно раскодированных, неправильно раскодированных сообщений и отказов от декодирования.