КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ. Н.И. ЛОБАЧЕВСКОГО
Кафедра компьютерной математики и информатики

М.Ф. НАСРУТДИНОВ

АЛГЕБРАИЧЕСКАЯ ТЕОРИЯ КОДИРОВАНИЯ И КРИПТОГРАФИЯ В SAGE

Учебное пособие

Казань
2024

УДК 512

Рецензент: доцент кафедры цифровой аналитики и технологий искусственного интеллекта Института информационных технологий и интеллектуальных систем КФУ, кандидат физико-математических наук Е.К. Липачев

Насрутдинов М.Ф. Алгебраическая теория кодирования и криптография в Sage: Учебное пособие/ М.Ф. Насрутдинов. – Казань: Казанский федеральный университет, 2024. – 78 с.

В пособии рассматриваются приложения абстрактной алгебры в теории кодирования и криптографии. Основной целью пособия является демонстрация возможностей использовании системы компьютерной алгебры (СКА) Sage в решении задач курса. Кроме того при помощи СКА можно быстро проверять свои гипотезы на небольших объектах прежде чем пытаться доказать их в общем виде.

Оглавление

Предисловие

В пособии рассматриваются приложения абстрактной алгебры в теории кодирования и криптографии. Основной целью пособия является демонстрация возможностей использовании системы компьютерной алгебры (СКА) Sage в решении задач курса. Не пытаясь заменить обстоятельные учебники, мы хотели продемонстрировать возможности Sage при работе с алгебраическими объектами. Применение СКА позволяет студентам 'не утонуть' в больших и зачастую рутинных выкладках и сконцентрироваться на основных идеях. Кроме того при помощи СКА можно быстро проверять свои гипотезы на небольших объектах прежде чем пытаться доказать их в общем виде.

Глава 1
Введение в теорию кодирования

Рассматривается задача обнаружения и исправления ошибок при передачи сообщений по каналам связи.

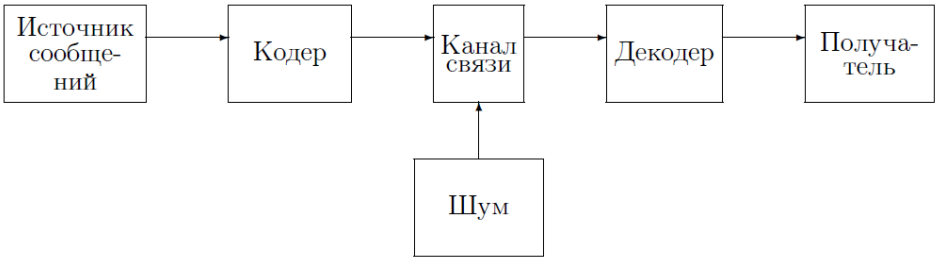

Рис. 1. Система связи по каналу с шумом.

Мы рассматриваем передачу сообщений блоками размера n над некоторым алфавитом X.

Общая схема работы подразумевается следующей:

- Источник формирует сообщение $u = (u_1, u_2, \ldots, u_k)$ длины k.
- Кодер добавляет проверочные символы к сообщению (само сообщение тоже может изменятся).
- Сообщение передается по каналу связи, в котором возможны помехи.
- Декодер проверяет наличие ошибок, при возможности исправляет их и предает декодированное сообщение получателю.

Пример 0.1. Код с повторениями. Пользователь хочет передать сообщение 1 или 0 (информационное сообщение, $k = 1$). Для уменьшения ошибки символ дублируется 5 раз. По каналу связи будет передано либо 00000, либо 11111.
Если будет получено, например, сообщение 01000, то значит в момент передачи произошла ошибка (обнаружение ошибки).
Если вероятность ошибки в каждом символе меньше 0.5, то вероятнее всего было передано сообщение 00000 (исправление ошибки).

Пример 0.2. Код проверки на четность. Пользователь передает сообщение $(u_1, u_2, \ldots, u_k)$, состоящее из 0 и 1. Кодер добавляет один дополнительный символ $u_{k+1} \in \{0, 1\}$, так чтобы сумма $u_1 + u_2 \ldots + u_k + u_{k+1}$ была четная. Если

в канале произойдет одна ошибка (или любое нечетное количество ошибок), то сумма станет нечетной и мы зафиксируем наличие ошибки.
В этом случае восстановить сообщение мы не сможем. Такие коды используют в системах, где вероятность ошибки очень маленькая, например, в вычислительных машинах для контроля передач информации между регистрами и для контроля считываемой информации в оперативной памяти.

Пример 0.3. ISBN-код. Международный стандартный книжный номер (англ. International Standard Book Number, сокращённо – англ. ISBN) – уникальный номер книжного издания, необходимый для распространения книги в торговых сетях и автоматизации работы с изданием. Пример ISBN-кода: ISBN 3-88053-002-5.
Изначально ISBN имел длину из 10 символов (сейчас 13), символы состоят из цифр от "0"до "9"и буквой "X". Между информационными символами стоят дефисы, которые нужны для удобства восприятия длинного числа, буква "X"служит для обозначения числа 10 и может стоять только на последней позиции.
Код состоит из четырех частей (между которыми располагается дефис): идентификаторы группы, издателя, книги для издателя, и контрольная цифра. Идентификатор группы используется для обозначения страны, географического региона, языка и прочее.Четвертая, заключительная часть (контрольная цифра), используется в коде алгоритме другими цифрами для получения поддающегося проверке ISBN. Количество цифр, содержащееся в первых трех частях, может быть различным, но контрольная цифра всегда содержит один символ (расположенный между "0"и "9"включительно, или "X"для величины 10), а само ISBN в целом имеет длину тринадцать символов (десять чисел плюс три дефиса, разделяющих три части ISBN).
Последний символ – проверочный. Если $x_1, x_2, \ldots, x_9$ – информационные символы, то проверочный символ x_{10} выбирается из условия

$$\sum_{i=1}^{10} i \cdot x_i = 0 \mod 11.$$

§1. Блоковые коды. Расстояние Хэмминга

В нашем курсе мы рассматриваем только блоковые коды. Это означает, что сообщения передаются блоками длины n над некоторым алфавитом F, состоящим из q элементов.

В качестве алфавита берут обычно какой-либо алгебраический объект, в приложениях это чаще всего конечное поле. Есть исследования, в которых алфавит это конечные кольца, группы и полугруппы, лупы.

Определение 1.1. Кодом длины n называется любое непустое подмножество $C \in F^n$. Если количество элементов $|C| = M$, то говорят, что C является (n, M)-кодом.
Элементы кода называются кодовыми словами.

Величина $k = \log_q M$ называется размерностью (или информационной длиной) кода C. Информационная длина показывает долю полезных (информационных) символов, которые передаются по каналу связи.

Фактически декодер проверяет принадлежит ли принятое слово множеству C или нет. Если слово не принадлежит C, то значит произошла ошибка. Далее будем решать следующие задачи:

1. Как задать C, чтобы проверка принадлежности была простой.

2. Если произошла ошибка, то можно ли восстановить исходное сообщение.

3. Как увеличить долю информационных символов, сохраняя корректирующие возможности кода.

Определение 1.2. *Расстояние Хэмминга* на множестве F^n определяется следующим образом:

$$d(x, y) = |\{i | x_i \neq y_i\}|,$$

где $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, $x, y \in X^n$.

Теорема 1.3. *Расстояние Хэмминга задает на F^n структуру метрического пространства.*

Определение 1.4. *Минимальное расстояние (кодовое расстояние)* кода определяется следующим образом:

$$d(C) = \min d(x, y),$$

где минимум берется по всем парам элементов $x, y \in C$, $x \neq y$.

Теорема 1.5. *Если кодовое расстояние $d(C) = 2t + 1$, то код может исправлять t и обнаруживать $2t$ ошибок.*

Если кодовое расстояние $d(C) = 2t$, то код может исправлять $t - 1$ и обнаруживать $2t - 1$ ошибку.

Чтобы вычислить расстояние Хэмминга между двумя векторами, можно пройтись циклом и подсчитать количество несовпадающих позиций.

Далее мы будем рассматривать линейные коды, для которых вводится понятие веса Хэмминга. Вес Хэмминга $wt(x)$ – это количество ненулевых координат вектора. С помощью Sage вес Хэмминга и расстояние можно вычислить следующим образом.

In [1]:
```
# Создадим два вектора над полем GF(2)
x=vector(GF(2), [1, 0, 1, 1, 0]);
y=vector(GF(2), [0, 1, 0, 1, 1]);
# Вес Хэмминга вектора x
print('Вес Хэмминга вектора x=', x.hamming_weight())
# Расстояние Хэмминга между векторами x и y
print('Расстояние Хэмминга между x и y=',
      (x-y).hamming_weight())
```

Out [1]: Вес Хэмминга вектора x= 3
Расстояние Хэмминга между x и y= 4

Пример. Найдите все слова, которые находятся на расстоянии 3 от слова 1010.

In [2]:
```
# Создадим вектор над полем GF(2)
x=vector(GF(2), [1, 0, 1, 0]);
# Найдем все слова перебором
for k in range(2):
    for l in range(2):
        for i in range(2):
            for j in range(2):
                y=vector(GF(2), [k, l, i, j])
                if (x-y).hamming_weight()==3:
                    print('y=', y)
```

Out [2]: y= (0, 0, 0, 1)
y= (0, 1, 0, 0)
y= (0, 1, 1, 1)
y= (1, 1, 0, 1)

§2. Упражнения

В качестве алфавита рассматривается множество $F = GF(2) = \{0, 1\}$.

1.1. Вычислите $d(11001, 01110)$.

1.2. Вычислите $d(0000, 0110)$.

1.3. Для данного множества $C \subset F^n$ найти (минимальное) кодовое расстояние. Для каждого из кодов найти: число ошибок, которые код обнаруживает и исправляет.

1) $C = \{101010, 010110, 000001\}$,

2) $C = \{01101010, 11000110, 00011001, 10101100\}$.

1.4. Для данного множества $C \subset F^n$ найти (минимальное) кодовое расстояние. Для каждого из кодов найти: число ошибок, которые код обнаруживает и исправляет.

1) $C = \{11000, 10101, 01110\}$, 2) $C = \{111100, 110011, 001111\}$.

1.5. Найдите все слова, которые находятся на расстоянии 3 от слова 1010 в F^4.

1.6. Найдите все слова, которые находятся на расстоянии 3 от слова 10101 в F^5.

1.7. Пусть $C = \{000000, 100110, 010101, 001011, 101101, 011110, 110011, 111000\}$. Учитывая, что это $[6, 8, 3]_2$ код, восстановите сообщения 000001.

1.8. Пусть $C = \{000000, 100110, 010101, 001011, 101101, 011110, 110011, 111000\}$. Учитывая, что это $[6, 8, 3]_2$ код, восстановите сообщения 011110.

Глава 2
Линейные коды

Для удобства кодирования и декодирования обычно рассматриваются коды с какой-либо алгебраической структурой. Наиболее важными с практической точки зрения являются линейные коды.

Далее в качестве алфавита берется конечное поле $F = GF(q)$, состоящее из q элементов.

Определение 0.1. *Линейным кодом* C называется подпространство векторного пространства F^n над полем F.

Размерность пространства C называется размерностью кода. Если k – размерность линейного пространства C и d – минимальное расстояние кода C, то говорят, что C линейный $[n, k, d]$-код над C.

Применяют также запись $[n, k]$-код и $[n, k, d]_q$-код для указания количества элементов в поле F.

Из определения ясно, что в C ровно q^k элементов.

Определение 0.2. *Весом Хэмминга* $wt(x)$ слова (вектора) $x \in F^n$ называется число ненулевых компонент x.

Теорема 0.3. *Расстояние Хэмминга для слов выражается через вес Хэмминга следующим образом:*

$$d(x, y) = wt(x - y).$$

Если код C линеен, то

$$d(C) = \min_{x \in C, x \neq 0} wt(x).$$

Линейный код как векторное подпространство можно задать либо через систему порождающих векторов (с помощью порождающей матрицы), либо как решение систем линейных уравнений (с помощью проверочной матрицы).

§1. Порождающая и проверочная матрица

Мы можем задать (построить) линейное подпространство с помощью порождающей матрицы

$$C = \{uG \mid u = (u_1, u_2, \ldots, u_k),\ G - k \times n \text{ матрица}\}.$$

Строки порождающей матрицы образуют базис подпространства C.

Пример. Код проверки на четность

$$\{(u_1, u_2, \ldots, u_k, u_1 + u_2 + \ldots + u_k) = (u_1, u_2, \ldots, u_k)(E_k|A)\}.$$

$$(E_k|A) = \begin{pmatrix} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ . & . & . & \ldots & . \\ 0 & 0 & \ldots & 1 & 1 \end{pmatrix}$$

Удобно представлять матрицу G в виде $(E_k \mid A)$. В этом случае исходное сообщение $u = (u_1, u_2, \ldots, u_k)$ кодируется вектором (u, uA).

Для однозначного кодирования важно, чтобы из условия $u \neq u'$ всегда следовало, что

$$uG \neq u'G.$$

Это эквивалентно тому, что строки матрицы G линейно независимы. Таким образом, строки матрицы G образуют базис линейного пространства C.

Линейный код можно задать и как решение систем однородных линейных уравнений

$$C = \{x \in K^n \mid xH^t = 0,\ H - (n-k) \times n \text{ матрица}\}.$$

Определение 1.1. Матрица H называется *проверочной матрицей* кода C, если выполнено условие: $x \in C$ тогда и только тогда, когда

$$xH^t = 0.$$

Проверочная и порождающая матрицы связаны соотношением

$$GH^t = 0.$$

Если $G = (-A^t|E_k)$, то

$$H = (E_{n-k}|A).$$

§2. Кодовое расстояние линейного кода

Теорема 2.1. *Если любые $s \leq d-1$ столбцов проверочной матрицы H линейного (n,k)-кода линейно независимы, то минимальное расстояние кода равно по меньшей мере d. Если при этом найдутся d линейно зависимых столбцов, то минимальное расстояние кода равно d в точности.*

Теорема 2.2. *Если минимальное расстояние линейного (n, k)-кода равно d, то любые $l \leq d-1$ столбцов проверочной матрицы H линейно независимы и найдутся d линейно зависимых столбцов.*

§3. Декодирование линейного кода

Для линейных кодов проверка на отсутствие ошибок сводится к умножению на проверочную матрицу.

Исправление ошибок осуществляется следующим образом. Для принятого вектора x вычисляется

$$xH^t.$$

Пусть $e = (e_1, e_2, \ldots, e_n)$ – вектор ошибок, $c = (c_1, c_2, \ldots, c_n)$ – сообщение, которое было послано. Тогда $x = e + c$ и

$$xH^t = (e + c)H^t = eH^t.$$

Введем отношение эквивалентности на множестве K^n. Векторы x и y эквивалентны, если

$$x - y \in C.$$

Два вектора x и y эквивалентны, если

$$(x - y)H^t = 0.$$

Рассмотрим класс эквивалентных элементов $x + C = \{x + c \mid c \in C\}$. Синдромом этого класса называется элемент $s \in x + C$ наименьшего веса (их может быть несколько, тогда выбираем произвольный из них).

Если запомнить все синдромы и из произведение на матрицу H^t, то исправление ошибки сводится к следующей процедуре:

1. Вычислить $r = xH^t$.

2. Если $r = 0$, то ошибок не было. Иначе найти из таблицы синдром s, соответствующий r и вернуть $x - s$.

Пример. Код Хэмминга.

Рассматриваем бинарный код. Код с проверочной матрицей, у которой в качестве столбцов берутся все двоичные представления чисел от 1 до m называется кодом Хэмминга.

Пример.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

проверочная матрица для $[2^3 - 1, 2^3 - 1 - 3, 3] = [7, 4, 3]$ кода.

В случае одной ошибки синдром eH^t совпадает с номером позиции, в которой произошла ошибка.

§4. Построение линейных кодов в Sage

Построить линейные коды в Sage можно различными способами.

1. Задание с помощью порождающей матрицы.

In [3]:
```
#Определим порождающую матрицу над полем GF(2)
M = matrix(GF(2), [[1, 0, 1, 1, 0],\
[0, 1, 0, 1, 1]])
#Создание линейного кода с порождающей матрицей G
C = LinearCode(M)
#Вывод всех  элементов кода
C.list()
```

Out [3]: [(0, 0, 0, 0, 0), (1, 0, 1, 1, 0), (0, 1, 0, 1, 1), (1, 1, 1, 0, 1)]

В примере сначала вводится матрица над полем GF(2), а потом определяется линейный код с порождающей матрицей M.

Функции для работы с кодами находятся в пространстве имен codes. Создан объект C – линейный код, у которого через методы можно найти его параметры.

Можем вычислить порождающую и проверочную матрицу, кодовое расстояние, размерность длину кода.

In [4]:
```
print('Порождающая матрица: ')
print(C.generator_matrix())
print('Проверочная матрица: ')
print(C.parity_check_matrix())
print('Кодовое расстояние = ', C.minimum_distance())
print('Размерность = ', C.dimension())
print('Длина = ', C.length())
```

Out [4]: Порождающая матрица:
[1 0 0 1 0]
[0 0 1 1 1]

[0 1 0 1 1]
Проверочная матрица:
[1 0 0 1 1]
[0 1 1 0 1]
Кодовое расстояние = 2
Размерность = 3
Длина = 5

Иногда нам может понадобиться перебрать все элементы кода C. Мы можем для этого пройтись по всем элемента кода с помощью цикла.

В следующем фрагменте мы выводим все веса Хэмминга для кода C.

```
In [5]: for c in C:
                    print(c, "wt=", c.hamming_weight())
```

```
Out [5]: (0, 0, 0, 0, 0) wt= 0
         (1, 0, 0, 1, 0) wt= 2
         (0, 0, 1, 1, 1) wt= 3
         (1, 0, 1, 0, 1) wt= 3
         (0, 1, 0, 1, 1) wt= 3
         (1, 1, 0, 0, 1) wt= 3
         (0, 1, 1, 0, 0) wt= 2
         (1, 1, 1, 1, 0) wt= 4
```

2. Специальные коды

Наиболее популярные коды уже реализованы в Sage. В следующем примере задается $[7, 4]$- код Хэмминга.

```
In [6]: # Код Хэмминга порядка m=3. Длина кода 2^m-1,
        # размерность 2^m-1-m
        C = codes.HammingCode(GF(2),3);C
```

```
Out [6]: [7, 4] Hamming Code over GF(2)
```

Напишем скрипт, выводящий параметры кода.

```
In [7]: # Функция для вывода параметров кода
        def printParameters(C):
            print('Порождающая матрица: ')
            print(C.generator_matrix())
            print('Проверочная матрица: ')
```

```
    print(C.parity_check_matrix())
    print('Кодовое расстояние = ',
          C.minimum_distance())
    print('Размерность = ', C.dimension())
    print('Длина = ', C.length())
```

Выведем теперь параметры, полученного кода.

In [8]:
```
printParameters(H)
```

Out [8]: Порождающая матрица:
[1 0 0 0 0 1 1]
[0 1 0 0 1 0 1]
[0 0 1 0 1 1 0]
[0 0 0 1 1 1 1]
Проверочная матрица:
[1 0 1 0 1 0 1]
[0 1 1 0 0 1 1]
[0 0 0 1 1 1 1]
Кодовое расстояние = 3
Размерность = 4
Длина = 7

Еще один пример кода

In [9]:
```
# Двоичный (расширенный) код Голея
C=codes.GolayCode(GF(2)); C
```

Out [9]: [24, 12, 8] Extended Golay code over GF(2)

Полный список доступных кодов можно найти в документации.

3. Кодирование и декодирование линейных кодов

Каждому линейному коду можно сопоставить кодировщик (encoder) и декодировщик (decoder). Рассмотрим как работают эти объекты.

In [10]:
```
# Построим новый код по порождающей матрице
G = matrix(GF(2), [[1, 0, 1, 1, 0], [0, 1, 0, 1, 1]]);
C=codes.LinearCode(G);
C.list()
```

Out [10]: [(0, 0, 0, 0, 0), (1, 0, 1, 1, 0), (0, 1, 0, 1, 1), (1, 1, 1, 0, 1)]

Найдем минимальное расстояние кода.

In [11]:
```
C.minimum_distance()
```

Out [11]: 3

Зададим кодировщик линейного кода.

In [12]:
```
#encoder кода C
Encoder=C.encoder(); Encoder
```

Out [12]: Generator matrix-based encoder for [5, 2] linear code over GF(2)

Воспользуемся кодировщиком для получения кодового слова из информационного сообщения.

In [13]:
```
# Зададим информационное сообщение
message=vector(GF(2), [1,1]);
# Закодируем информационное сообщение = message*G
word=Encoder(message); word
```

Out [13]: (1, 1, 1, 0, 1)

Сымитируем возникновение одной ошибки в кодовом слове при передаче по каналу связи.

In [14]:
```
#Зададим вектор ошибок
err_vect = vector(GF(2), (0, 0, 0, 0, 1))
#Слово, в котором произошла она ошибка
word_err = word+err_vect;
word_err
```

Out [14]: (1, 1, 1, 0, 0)

Создадим декодер.

In [15]:
```
# decoder кода C
Decoder=C.decoder(); Decoder
```

Out [15]: Syndrome decoder for [5, 2] linear code over GF(2) handling errors of weight up to 2

Применим декодер к слову без ошибок.

In [16]:
```
# Декодирование вектора без ошибок
Decoder.decode_to_code(word)
```

Out [16]: (1, 1, 1, 0, 1)

Применим декодер к слову с одной ошибкой.

In [17]:
```
# Декдирование вектора с одной ошибкой
Decoder.decode_to_code(word_err)
```

Out [17]: (1, 1, 1, 0, 1)

Декодер восстановил правильное слово.

Теперь получим исходное информационное сообщение.

In [18]:
```
# Получение вектора информационных символов
Decoder.decode_to_message(word_err)
```

Out [18]: (1,1)

Можем получить полный список синдромов и лидеров смежных классов. Для этого получим сначала объект - декодер на основе таблицы синдромов и лидеров смежных классов.

In [19]:
```
# список синдромов и лидеров смежных классов
D=codes.decoders.LinearCodeSyndromeDecoder(C);D
```

Out [19]: Syndrome decoder for [5, 2] linear code over GF(2) handling errors of weight up to 2

Выведем таблицу в виде синдром - лидер смежного класса.

In [20]:
```
T=D.syndrome_table(); T
```

Out [20]: (0, 0, 0): (0, 0, 0, 0, 0),
(1, 0, 0): (1, 0, 0, 0, 0),

(0, 1, 0): (0, 1, 0, 0, 0),
(0, 0, 1): (0, 0, 1, 0, 0),
(1, 0, 1): (0, 0, 0, 1, 0),
(1, 1, 1): (0, 0, 0, 0, 1),
(1, 1, 0): (1, 1, 0, 0, 0),
(0, 1, 1): (1, 0, 0, 0, 1)

Для демонстрации выведем все смежные классы по линейному пространству C.

Сначала опишем вспомогательную функцию.

In [21]:
```
# Функция для вывода членов смежного класса
# вектора v по C
def cosetList(C, v):
    l=[]
    v=vector(GF(2), v)
    for x in C.list():
        l.append(v+x)
    return l
```

Применим эту функцию для вывода всех смежных классов

In [22]:
```
for s in T:
    print("Синдром: ", s)
    print("Лидер класса: ", T[s])
    print("Члены смежного класса: ",
           cosetList(C, T[s]), "\n")
```

Out [22]: Синдром: (0, 0, 0)
Лидер класса: (0, 0, 0, 0, 0)
Члены смежного класса: [(0, 0, 0, 0, 0), (1, 0, 1, 1, 0), (0, 1, 0, 1, 1), (1, 1, 1, 0, 1)]

Синдром: (1, 0, 0)
Лидер класса: (1, 0, 0, 0, 0)
Члены смежного класса: [(1, 0, 0, 0, 0), (0, 0, 1, 1, 0), (1, 1, 0, 1, 1), (0, 1, 1, 0, 1)]
...
Синдром: (1, 1, 1)
Лидер класса: (0, 0, 0, 0, 1)

Члены смежного класса: [(0, 0, 0, 0, 1), (1, 0, 1, 1, 1), (0, 1, 0, 1, 0), (1, 1, 1, 0, 0)]
...
Синдром: (0, 1, 1)
Лидер класса: (1, 0, 0, 0, 1)
Члены смежного класса: [(1, 0, 0, 0, 1), (0, 0, 1, 1, 1), (1, 1, 0, 1, 0), (0, 1, 1, 0, 0)]

В этом примере можно заметить, что в последней строке вес лидера смежного класса равен двум и внутри смежного класса два элемента с таким весом. Это согласуется с тем, что кодовое расстояние равно 3 и код может исправлять только одну ошибку.

§5. Упражнения

Все коды рассматриваются над полем $F = GF(2)$. Для вычислений используйте Sage.

2.1. Пусть $C = \{000000, 100110, 010101, 001011, 101101, 011110, 110011, 111000\}$. Зная, что это линейный код, найдите минимальное кодовое расстояние. Можете ли вы доказать, что это действительно линейный код? Найдите базисные векторы.

2.2. Найдите проверочную и порождающую матрицу линейного кода

$$C = \{(x_1, x_2, x_3, x_4) \mid x_1 + x_2 + x_3 + x_4 = 0\}.$$

Найдите кодовое расстояние кода из проверочной матрицы, число ошибок, которые код обнаруживает и исправляет. Найдите все синдромы кода и декодируйте $(1, 1, 1,)0$.

2.3. Пусть

$$C = \{000000, 101110, 001010, 110111, 100100, 011001, 111101, 010011\}$$

Найдите порождающую и проверочную матрицу линейного кода.

2.4. Выпишите все кодовые слова двоичного кода, заданного порождающей матрицей. Найдите проверочную матрицу линейного кода.

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

Найдите кодовое расстояние кода из проверочной матрицы, число ошибок, которые код обнаруживает и исправляет. Найдите все синдромы кода. Декодируйте 1111 и 0101.

2.5. Найдите проверочную матрицу линейного кода с порождающей матрицей

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

Найдите кодовое расстояние кода из проверочной матрицы, число ошибок, которые код обнаруживает и исправляет. Найдите все синдромы кода. Декодируйте 11111 и 01011.

2.6. Для данного множества $V \subset B^n$ найти (минимальное) кодовое расстояние. Для каждого из кодов найти: число ошибок, которые код обнаруживает и исправляет.

1) $V = \{11000, 10101, 01110\}$,

2) $V = \{111100, 110011, 001111\}$.

2.7. Найдите все слова, которые находятся на расстоянии 3 от слова 11000 в $GF(2^5)$.

2.8. Рассмотрим $(6, 8, 3)_2$ линейный код

$$C = \{000000, 100110, 010101, 001011, 110011, 101101, 011110, 111000\}$$

с порождающими $100110, 010101, 001011$.

Покажите, что кодовое расстояние равно трем. Восстановите сообщения $111100, 111011$.

2.9. Выпишите все элементы линейного кода с порождающей матрицей

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

Найдите кодовое расстояние кода, число ошибок, которые код обнаруживает и исправляет.

2.10. Выпишите все элементы линейного кода с порождающей матрицей

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

Найдите кодовое расстояние кода, число ошибок, которые код обнаруживает и исправляет.

2.11. Используя бинарный код Хэмминга $[7, 4, 3]_2$, декодируйте сообщения $y_1 = (1, 1, 0, 1, 1, 0, 0)$.

2.12. Используя бинарный код Хэмминга $[7, 4, 3]_2$, декодируйте сообщения $y_2 = (1, 1, 1, 1, 1, 1, 1)$, $y_3 = (1, 1, 1, 0, 0, 0, 0)$.

2.13. Пусть C_1 и C_2 линейные коды одинаковой длины n с порождающими матрицами G_1 и G_2 и размерностями $k_1 \geq k_2 > 0$. Определим следующие (не обязательно линейные) коды: $C_3 = C_1 \cup C_2$; $C_4 = C_1 \cap C_2$; $C_5 = C_1 + C_2 = \{x + y \mid x \in C_1, y \in C_2\}$; $C_6 = \{(x, y) \mid x \in C_1, y \in C_2\} \subset F^{2n}$.

1) Покажите, что коды C_4, C_5, C_6 – линейные.
2) При каких условиях код C_3 является линейным?
3) Докажите, что кодовое расстояние $d(C_4) \geq \max(d(C_1), d(C_2))$.
4) Выразите порождающую матрицу кода C_6 через матрицы G_1 и G_2.
5) Докажите, что кодовое расстояние $d(C_6) = \min(d(C_1), d(C_2))$.

Глава 3
Границы объемов кодов

В этой главе мы приведем некоторые простые оценки, связывающие параметры кода.

Пусть F – алфавит из q элементов, C – (n, M, d)-код. Напомним, что это означает, что длина кодовых слов равна n, число элементов в C равно M (мы также будем писать $|C|$), минимальное расстояние между словами равно d. В этом случае код C может обнаруживать $d-1$ ошибку и исправлять $t = [\frac{d-1}{2}]$ ошибок.

§1. Граница сферической упаковки

Обозначим через

$$B_r(x) = \{y \in F^n \mid d(x, y) \leq r\}$$

шар радиуса r.

Лемма 1.1. *Количество элементов в $B_r(x)$ равно*

$$|B_r(x)| = \sum_{k=0}^{r} C_n^k (q-1)^k.$$

Действительно. Пусть x – слово длины n. Подсчитаем сколько векторов y имеет расстояние $d(x, y) = k$. Существует ровно C_n^k способов выбрать позиции $i_1, i_2, \ldots, i_k$, в которых слова x и y отличаются. Если позиции зафиксированы, то существует $(q-1)^k$ способов поставить элементы отличные от $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$.

Отметим также, что количество элементов в шаре $B_t(x)$ не зависит от центра. Обозначим $|B_t| = |B_t(x)|$ для произвольного x.

Теорема 1.2. *(Граница Хэмминга, граница сферической упаковки) Пусть C – (n, M)-код, исправляющий t ошибок. Тогда*

$$|C| \leq \frac{q^n}{\sum\limits_{k=0}^{t} C_n^k (q-1)^k}.$$

Доказательство. Так как код C исправляет t ошибок, то шары радиуса t с центрами в точках из C не пересекаются. Таким образом $\bigcup\limits_{x\in C} B_t(x)$ состоит из различных элементов и количество этих элементов не превосходит q^n. Итак

$$\bigcup_{x\in C} B_t(x) = |C||B_t| \leq q^n.$$

Применяя лемму, получим

$$|C| \leq \frac{q^n}{|B_t|} = \frac{q^n}{\sum\limits_{k=0}^{t} C_n^k (q-1)^k}.$$

□

Определение 1.3. Код, для которого достигается равенство, называется *совершенным.*

Пример 1.4. Код Хэмминга является совершенным.

Действительно, код Хэмминга является $[2^m-1, 2^m-1-m, 3]_2$-кодом и исправляет одну ошибку. Количество элементов в коде равно 2^{2^m-1-m}. Найдем $B_1(x) = C_n^0 + C_n^1(2-1) = 1+n = 1+2^m-1 = 2^m$. Имеем $2^{2^m-1-m} = 2^{2^m-1}/2^m$.

Замечание. Можно показать, что совершенные двоичные линейные коды исчерпываются кодами Голея, кодами Хэмминга и кодом с повторением.

§2. Граница Синглтона

Теорема 2.1. *(Граница Синглтона) Для произвольного кода $C \subseteq F^n$ над алфавитом из q элементов с минимальным расстоянием d выполнено*

$$|C| \leq q^{n-d+1}.$$

В частности, если код C линейный и $k = \dim C$, то

$$d \leq n - k + 1.$$

Доказательство. Для любого подмножества $C \subseteq F^n$ имеем $|C| \leq q^n$. Рассмотрим

$$C' = \{(c_d, c_{d+1}, \ldots, c_n) \mid \exists (c_1, c_2, \ldots, c_{d-1}, c_d, c_{d+1}, \ldots, c_n) \in C\}.$$

Так как минимальное расстояние между словами в C равно d, то все элементы в C' различны. Поэтому $|C| = |C'| \leq q^{n-(d-1)} = q^{n-d+1}$. □

Определение 2.2. Код, для которого выполняется равенство в оценке Синглтона, называется *кодом с максимально достижимым расстоянием* (МДР-кодом).

Пример 2.3. Код Хэмминга не является МДР-кодом.

Пример 2.4. Код с повторениями и код проверки на четность являются МДР-кодами.

Пример 2.5. (обобщенный код Рида-Соломона) Пусть $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ различные элементы поля $F = GF(q)$. Обобщенным кодом Рида-Соломона назовем $[n, k, d]$ код с проверочной матрицей

$$H = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \ldots & \alpha_n^{n-k-1} \end{pmatrix}.$$

Естественно, $n < q$ и $k \leq n - 2$.

В этой матрице любые $n - k$ столбцов линейно независимы. Поэтому $d \geq n - k + 1$. С другой стороны из границы Синглтона $d \leq n - k + 1$. Таким образом, $d = n - k + 1$.

§3. Граница Варшамова-Гилберта

Обозначим через

$$B_q(n, t) = \sum_{k=0}^{t} C_n^k (q-1)^k$$

количество элементов в шаре радиуса t в n-мерном пространстве.

Теорема 3.1 (Граница Варшамова-Гилберта). *Пусть $F = GF(q)$, n, k, d – положительные целые числа, для которых выполнено условие*

$$B_q(n-1, d-2) < q^{n-k}.$$

Тогда существует линейный $[n, k, d]_q$-код.

Доказательство. Построим $n-k \times n$ матрицу H, у которой любые $d-1$ столбец линейно независимы. Это даст нам линейный код с требуемым расстоянием d.

Первые $n-k$ столбцов $h_1, h_2, \ldots, h_{n-k}$ матрицы H составляют единичную матрицу. Остальные столбцы будем строить итеративно.

Пусть уже построены столбцы $h_1, h_2, \ldots, h_{s-1}$, удовлетворяющие свойству, что любые $d-1$ столбец линейно независимые. Необходимо найти вектор $h_s \in F^{n-k}$, такой, что в построенном наборе снова любые $d-1$ столбец линейно независимые.

Рассмотрим множество всех векторов вида

$$\{\alpha_1 h_1 + \alpha_2 h_2 + \ldots + \alpha_{s-1} h_{s-1}\},$$

где $\alpha_i \in F$, по крайней мере один из $\alpha_i = 0$.

Это те векторы, которые 'не подходят' для построения следующего столбца $h_s \in F^{n-k}$. Их количество не превосходит числа векторов вида

$$a = (\alpha_1, \alpha_2, \ldots, \alpha_{s-1}),$$

имеющих вес Хэмминга $wt(a) \leq d-2$.

Так как

$$|\{a \mid wt(a) \leq d-2\}| = B_q(s-1, d-2) \leq B_q(n-1, d-2) < q^{n-k},$$

то существует вектор h_s, удовлетворяющий условию. □

Упражнения.

3.1. Рассмотрите линейный код с порождающей матрицей

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Является ли эти код совершенным или МДР-кодом?

3.2. Рассмотрите линейный код с порождающей матрицей

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Является ли эти код совершенным или МДР-кодом?

3.3. Напишите функцию isPerfect() на языке Python, которая принимает на вход линейный код и проверяет является ли он совершенным.

Глава 4
БХЧ-коды и коды Рида-Соломона

В этой главе мы рассмотрим два примера кодов: БХЧ-коды и коды Рида-Соломона. Эти примеры служат обоснованием введения понятия циклических кодов, которые рассматриваются в следующей главе.

§1. Коды Боуза-Хоквингема-Чоудхури

Построим код, который умеет исправлять заданное число ошибок, и количество проверочных символов будет порядка $\log(n)$, где n – длина кодовых слов.

Пусть в качестве алфавита выбрано поле $F = GF(q)$. Зафиксируем m и построим код с минимальным расстоянием $d = 2t + 1$ длины $n = q^m - 1$.

Рассмотрим расширение E поля F порядка m. Пусть β – примитивный элемент поля E. Тогда все элементы

$$\beta, \beta^2, \beta^3, \ldots, \beta^{2t}$$

различны. Обозначим

$$h_1, h_2, \ldots, h_{2t}$$

минимальные многочлены этих элементов над полем $GF(q)$.

Пусть теперь

$$g(x) = \operatorname{lcm}(h_1, h_2, \ldots, h_{2t})$$

наименьшее общее кратное этих многочленов.

Будем кодировать сообщения по следующей схеме:

$$(u_0, u_1, \ldots, u_{k-1}) \to u_0 + u_1 x g(x) + \ldots + u_{k-1} x^{k-1} g(x) \mod (x^n - 1)$$

Здесь мы используем соответствие между векторами вида $(c_0, c_1, \ldots, c_{n-1})$ и многочленами $c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$.

Таким образом, мы можем закодировать слова (многочлены) степени $k \leq n - \deg g$. Отметим, что степени h_i не превышают m, поэтому $k \geq n - 2tm$.

Теорема 1.1. *Для построенного кода C кодовое расстояние больше или равно d.*

Доказательство. Слово $c \in C$ тогда и только тогда, когда $c(x)$ аннулируется элементами $\beta, \beta^2, \beta^3, \ldots, \beta^{2t}$.

Запишем это в матричном виде

$$\begin{pmatrix} 1 & \beta & \beta^2 & \ldots & \beta^{(n-1)} \\ 1 & \beta^2 & \beta^{2(2)} & \ldots & \beta^{(n-1)2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \beta^{2t} & \beta^{2(2t)} & \ldots & \beta^{(n-1)(2t)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = 0.$$

Предположим, что $w = wt(c) \leq d - 1$. Это означает, что существует индексы $a_1, a_2, \ldots, a_w$, такие, что $c_i \neq 0$, только для $i \in \{a_1, a_2, \ldots, a_w\}$.

Тогда выделяя подходящие столбцы матричное равенство можно записать в виде

$$\begin{pmatrix} \beta^{a_1} & \beta^{a_2} & \ldots & \beta^{a_w} \\ \beta^{a_1(2)} & \beta^{a_2(2)} & \ldots & \beta^{a_w(2)} \\ \vdots & \vdots & \vdots & \vdots \\ \beta^{a_1(2t)} & \beta^{a_2(2t)} & \ldots & \beta^{a_w(2t)} \end{pmatrix} \begin{pmatrix} c_{a_1} \\ c_{a_2} \\ \vdots \\ c_{a_w} \end{pmatrix} = 0.$$

Так как по предположению $w \leq d-1$, то получим СЛУ с квадратной матрицей и определитель этой матрицы не равен нулю (надо вынести из каждого столбца $\beta_{a_i} l$ и получим определитель Вандермонда). Значит система может иметь только нулевое решение – противоречие. □

Коды, построенные таким способом, называются кодами Боуза-Хоквингема-Чоудхури (БХЧ-кодами).

§2. Код Рида-Соломона

Пусть $F = GF(q)$ – конечное поле, $\alpha_1, \alpha_2, \ldots, \alpha_n \in F$ и $k \leq n$.

Определение 2.1. Кодом Рида-Соломона называется линейный код $C \subseteq F^n$ с порождающей матрицей

$$G = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^k & \alpha_2^k & \ldots & \alpha_n^k \end{pmatrix}.$$

Если $u = (u_0, u_1, \ldots, u_{k-1})$ – информационное сообщение и $u(x) = u_0 + u_1 x + \ldots + u_{k-1}x^{k-1}$ соответствующий ему многочлен, то кодовое слово вычисляется по формуле

$$uG = (u(\alpha_1), u(\alpha_2), \ldots, u(\alpha_n)).$$

Так как многочлен степени не выше $k-1$ имеет не больше $k-1$ корней, то различные кодовые слова могут имеют расстояние $d \geq n-(k-1)$. Учитывая оценку Синглтона, получим, что код Рида-Соломона является МДР кодом, то есть

$$d(C) = n - k + 1.$$

Пусть $d = 2t+1$, то есть код может исправлять t ошибок, тогда $n = k+2t$.

Покажем, что проверочная матрица кода Рида-Соломона может быть выбрана в виде

$$H = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \ldots & \alpha_n^{n-k-1} \end{pmatrix} \begin{pmatrix} v_1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & v_2 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & v_{n-1} & 0 \\ 0 & 0 & 0 & \ldots & 0 & v_n \end{pmatrix}$$

Найдем неизвестные $v_1, v_2, \ldots, v_n$ из условия

$$A = HG^t = 0.$$

$$a_{ij} = \sum_{s=1}^{n} H_{is}G_{js} = \sum_{s=1}^{n} \alpha_s^{i-1} v_s \alpha_s^{j-1} = \sum_{s=1}^{n} \alpha_s^{i+j-2} v_s,$$
$$i = 1, 2, \ldots, n-k; \qquad j = 1, 2, \ldots, k.$$

Получим однородную систему линейных уравнений относительно $v_1, v_2, \ldots, v_n$

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-2} & \alpha_2^{n-2} & \ldots & \alpha_n^{n-2} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = 0.$$

Эта система имеет ненулевое решение. Если рассматривать решения этой СЛУ как линейный код, то этот код имеет кодовое расстояние n, так как любые $n-1$ столбец линейно независимы. Поэтому можем найти набор $v_1, v_2, \ldots, v_n$, для которого все $v_i \neq 0$.

Итак, мы получили, что кодовые слова кода Рида-Соломона это в точности те $(c_0, c_1, \ldots, c_{n-1})$, для которых

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \ldots & \alpha_n^{n-k-1} \end{pmatrix} \begin{pmatrix} v_1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & v_2 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & v_{n-1} & 0 \\ 0 & 0 & 0 & \ldots & 0 & v_n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = 0$$

Декодирование кода Рида-Соломона.

Опишем основную идею декодирования кодов Рида-Соломона.

Пусть $u(x) = u_0 + u_1 x + \ldots + u_{k-1} x^{k-1}$ полином степени меньше k и передана таблица значений $u(\alpha_1), u(\alpha_2), \ldots, u(\alpha_n)$, в которых может произойти не более чем t ошибок. Требуется восстановить $u(x)$. Обозначим полученные значения $\widetilde{u}(\alpha_1), \widetilde{u}(\alpha_2), \ldots, \widetilde{u}(\alpha_n)$

Отметим, что если бы ошибок не было, то взяв любые k значений

$$u(\alpha_{i_1}), u(\alpha_{i_2}), \ldots, u(\alpha_{i_k})$$

можно восстановить $u(x)$ построив интерполяционный многочлен по k точкам.

Предположим ошибки произошли в точках $\beta_1, \beta_2, \ldots, \beta_s$, $s \leq t$ (отметим, что в реальности нам эти позиции не известны). Существует многочлен $D(x)$ степени t, старший коэффициент которого равен 1 и который обращается в ноль в точках $\beta_1, \beta_2, \ldots, \beta_s$. Например, это может быть многочлен вида $(x - \beta_1)(x - \beta_2) \ldots (x - \beta_s) x^{t-s}$.

Рассмотрим многочлен

$$Q(x) = u(x) D(x).$$

Он будет обладать следующим свойством:

$$Q(\alpha_i) = \widetilde{u}(\alpha_i) D(\alpha_i) = u(\alpha_i) D(\alpha_i).$$

Лемма 2.2. *Существуют многочлены* $\widetilde{Q}(x)$ *и* $\widetilde{D}(x)$, *такие что* $\deg \widetilde{D}(x) \leq t$, $\deg \widetilde{Q}(x) < k + t$ *и*

$$\widetilde{Q}(\alpha_i) = \widetilde{u}(\alpha_i) \widetilde{D}(\alpha_i), \qquad i = 1, 2, \ldots, n.$$

Доказательство. Имеем систему однородных линейных уравнений с менее чем $t+k+t = n$ неизвестными (коэффициентами многочленов) и n уравнениями. Разрешая эту систему и выбирая ненулевое решение, получим требуемые многочлены. □

Лемма 2.3. *Пусть $\widetilde{Q}(x)$ и $\widetilde{D}(x)$ – многочлены, условиям предыдущей леммы. Тогда $\widetilde{Q}(x)$ делится на $\widetilde{D}(x)$ и*

$$\widetilde{Q}(x)/\widetilde{D}(x) = u(x).$$

Доказательство. Многочлены $u(x)\widetilde{D}(x)$ и $\widetilde{Q}(x)$ имеют степень меньше $t+k$ и совпадают в не менее чем $n-t$ точках. Так как $n-t = k+2t-t = k+t$, то $u(x)\widetilde{D}(x) = \widetilde{Q}(x)$. □

§3. Sage

Построение кода Боуза-Чоудхури-Хоквингема по определению.

In [23]:
```
#Зададим поле из q элементов
q=2; F=GF(q)
# Зададим параметр m и количество ошибок,
# которые будет исправлять код
m=5; t=5
#Расширение поля F
E.<a>=GF(q^m);E
#Длина блока n
n=q^m-1
#Для работы с полиномами над полем F
R.<x> = PolynomialRing(F,'x')
#S=[a^k for k in range(1,t+1)];
# НОК минимальных многочленов для степеней
# примитивного элемента
g=lcm([s.minimal_polynomial()
       for s in [a^k for k in range(1,2*t+1)]])
#Циклический код
C = codes.CyclicCode(generator_pol=g, length=q^m-1)
```

Выведем параметры построенного кода.

```
In [24]: C
```

```
Out [24]: [31, 11] Cyclic Code over GF(2)
```

```
In [25]: C.minimum_distance()
```

```
Out [25]: 11
```

Зададим кодировщик сообщений.

```
In [26]: # Кодер (энкодер), кодирующий сообщение в код.
         # На вход подаются сообщения длины k
         Encoder=C.encoder()
         # исходное сообщение, соответствует полиному 1+x+x^2
         message=vector(GF(2), [1,1,1,0,0,0,0,0,0,0,0])
         #Кодирование сообщения
         Encoder(message)
```

```
Out [26]: (1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
          0, 0, 0, 0)
```

```
In [27]: # Порождающий многочлен кода C
         gen=C.generator_polynomial();gen
```

```
Out [27]: x^20 + x^18 + x^17 + x^13 + ...
```

Убедимся, что кодирование это просто умножение на порождающий многочлен.

```
In [28]: gen*(x^2+x+1)
```

```
Out [28]: x^22 + x^21 + x^17 + x^15 + x^14 + ...
```

Построение БХЧ-кодов, используя библиотечные функции.

```
In [29]: C1=codes.BCHCode(GF(2), 31, 11); C1
```

```
Out [29]: [31, 11] BCH Code over GF(2) with designed distance 11
```

```
In [30]: C1.generator_polynomial()
```

```
Out [30]: x^20 + x^18 + x^17 + x^13 + x^10 +... + 1
```

Под названием кодов Рида-Соломона понимается семейство кодов.

```
In [31]: #Классический код Рида-Соломона (циклический)
         C4=codes.ReedSolomonCode(GF(64,'a'), 9, 4); C4
```

```
Out [31]: [9, 4, 6] Reed-Solomon Code over GF(64)
```

Этот код является циклическим и его можно перевести в циклический и найти средствами Sage порождающий полином:

```
In [32]: Ccyc4 = codes.CyclicCode(code=C4); Ccyc4
```

```
Out [32]: [9, 4] Cyclic Code over GF(64)
```

```
In [33]: Ccyc4.generator_polynomial()
```

```
Out [33]: x^5 + a^3*x^4 + (a^5 + a^4 + a^3 + a^2 + a + 1)*x^3 +
          (a^4 + a^2 + a)*x^2 + a^3*x + a^3 + a^2 + a + 1
```

В то же время обобщенный код Рида-Соломона не является циклическим кодом.

```
In [34]: #Обобщенный код Рида-Соломона (не циклический)
         F = GF(59)
         n, k = 40, 12
         C5 = codes.GeneralizedReedSolomonCode(F.list()[:n], k);
         C5
```

```
Out [34]: [40, 12, 29] Reed-Solomon Code over GF(59)
```

Попытка перевести этот код в циклический, посредством Ccyc5 = codes.CyclicCode(code=C5), приведет к ошибке.

§4. Упражнения

4.1. Рассмотрите расширение $GF(8)$ поля $GF(2)$. Постройте БХЧ код, исправляющий одну, две и три ошибки.

4.2. Постройте код Рида-Соломона над полем $GF(2^7)$, используя конструкцию из лекций:

$$u(x) \to (u(\alpha), u(\alpha^2) \ldots, u(\alpha^s)),$$

где $\deg u(x) < k$, $s > k$, α – примитивный элемент поля $GF(2^7)$. Вычислите минимальное расстояние кода, проверьте, что это МДР код.

4.3. Пусть $GF(4) = GF(2)[\alpha]$ поле из четырех элементов с примитивным

элементом α, который является корнем многочлена $x^2 + x + 1$. Пусть $P = \{0, 1, \alpha, \alpha^2 = 1 + \alpha\}$. Рассмотрите код длины 4 размерности 2:

$$C = RS(4; 2; P) = \{(f(0), f(1), f(\alpha), f(\alpha^2) \mid f(x) \in GF(4)[x]_{<2}\}.$$

Выпишите все слова C. Сколько различных слов в C? Найдите порождающую и проверочную матрицы кода C. Убедитесь, что все ненулевые слова C имеют вес не меньше 3.

4.4. Пусть $C = RS(4; 2; P)$. Декодируйте сообщения $(1, 1, \alpha, \alpha^2)$, $(\alpha, \alpha, \alpha, 1)$, $(1, \alpha, \alpha, 0)$.

4.5. Рассмотрим поле $GF(5)$ и $P = (1; 3; 2; 4; 0)$. Постройте код $C = RS(5; 3; P)$. Найдите его порождающую матрицу. Возьмите произвольный вектор $x \in C$, изменяя в нем одну координату, получите вектор y. Убедитесь, что $y \notin C$, исправьте ошибку при помощи алгоритма декодирования.

Глава 5
Циклические коды

Циклические коды обладают дополнительной алгебраической структурой. Это позволяет организовать кодирование и декодирование более простыми способами, чем общая конструкция линейного кода. Среди циклических кодов достаточно много кодов с хорошими свойствами.

Определение 0.1. Линейный код $C \subseteq F^n$ называется циклическим, если из того, что

$$c = (c_0, c_1, \ldots, c_{n-2}, c_{n-1})$$

принадлежит коду C, следует, что вектор (слово в алфавите F)

$$(c_{n-1}, c_0, c_1, \ldots, c_{n-2})$$

также принадлежит коду C. Это кодовое слово называется *циклическим сдвигом* слова c.

Пример 0.2. Код $C_1 = \{(0,0,0,0), (0,1,0,1), (1,0,1,0), (1,1,1,1)\}$ является циклическим, а код $C_2 = \{(0,0,0,0), (1,0,0,1), (0,1,1,0), (1,1,1,1)\}$ не является.

Пример 0.3. Код с повторением и двоичный код проверки на четность – циклические.

Циклические коды можно рассматривать как идеалы в кольце $F[x]/(x^n - 1)$ или идеалы в групповой алгебре FC_n над циклической группой C_n.

§1. Идеалы и циклические коды

Векторное пространство F^n можно представлять в разных формах. Например, это пространство естественным образом изоморфно пространству V_n всех многочленов из $F[x]$, степени которых строго меньше n. Изоморфизм осуществляется следующим образом:

$$c = (c_0, c_1, \ldots, c_{n-1}) \leftrightarrow c(x) = \sum_{i=0}^{n-1} c_i x^i.$$

В дальнейшем мы будем по мере необходимости переходить от одного из этих пространств к другому, используя описанный выше изоморфизм.

Еще одним представлением пространства F^n является факторкольцо кольца многочленов по идеалу, порожденному многочленом $x^n - 1$.

Отождествим множество полиномов степени строго не выше n с факторкольцом

$$S = F[x]/(x^n - 1)$$

кольца многочленов $F[x]$ по идеалу, порожденному $x^n - 1$.

Будем обозначать через $\overline{f(x)}$ образ элемента $f(x) \in F[x]$ в S при естественном эпиморфизме.

Наше отождествление F^n и V_n может быть продолжено до отождествления F^n и S.

$$v = (v_0, v_1, \ldots, v_{n-1}) \leftrightarrow \overline{v} = \sum_{i=0}^{n-1} v_i \overline{x}^i.$$

Легко видеть, что умножение полинома в S на $\overline{x}$ эквивалентно циклическому сдвигу соответствующего вектора в F^n.

Учитывая вышесказанное, справедливо еще одно определение циклического кода:

Определение 1.1. Код C (как подмножество в S) называется циклическим, если C является идеалом в кольце S.

Замечание. С точки зрения вычислений факторкольцо $F[x]/(x^n - 1)$ выглядит как множество многочленов степени не выше чем $n - 1$ с обычной операцией сложения многочленов, а умножение производится по модулю многочлена $x^n - 1$, то есть сначала надо перемножить многочлены обычным способом, а потом взять остаток от деления на $x^n - 1$.

Далее мы будем отождествлять $\overline{g(x)}$ и его прообраз в $F[x]$ для обозначения элементов из S и опускать значок верхнего подчеркивания.

Итак, циклические коды находятся во взаимно-однозначном соответствии с идеалами факторалгебры $F[x]/(x^n - 1)$ и могут быть описаны на языке многочленов.

Так как кольцо многочленов $F[x]$ является кольцом главных идеалов, то все идеалы S являются главными и порождаются многочленами $\overline{g(x)}$, где $g(x)$ делит $x^n - 1$.

Более точно строение порождающих многочленов описывается следующей теоремой.

Теорема 1.2. *Пусть C – циклический код в $F[x]/(x^n-1)$. Тогда существует (и единственный) многочлен $g(x)$, порождающий идеал C, такой, что его степень $r < n$ минимальна и коэффициент при старшей степени равен 1.*

Многочлен $g(x)$ делит x^n-1 и любой элемент из C может быть записан в виде произведения $g(x)a(x)$, где степень $a(x)$ меньше $n-r$.

Пример 1.3. Циклические бинарные коды длины 7.

Рассмотрим все простые делители многочлена x^7-1 в кольце многочленов $GF(2)[x]$. Используя Sage, получим разложение многочлена на простые множители.

```
In [35]: #Определим кольцо многочленов над конечным полем
         R = PolynomialRing(GF(2),'x')
         #Нужно определить переменную x как объект этого кольца
         x = R.gen()
         #Запишем многочлен, который будем раскладывать
         f = x^7-1
         #Операция разложения на неприводимы множители
         f.factor()
```

```
Out [35]: (x + 1) * (x^3 + x + 1) * (x^3 + x^2 + 1)
```

Итак,

$$x^7-1 = (x+1) * (x^3+x+1) * (x^3+x^2+1).$$

Перебирая все делители многочлена x^7-1 мы можем построить $C_3^1+C_3^2 = 3+3 = 6$ кодов.

Выпишем, например, циклический код, порожденный $(x^3+x+1) * (x^3+x^2+1) = x^6+x^5+x^4+x^3+x^2+x+1$:

$$C = \{0, x^6+x^5+x^4+x^3+x^2+x+1\}$$

Этот код имеет размерность 1 и состоит из двух кодовых слов.

Теорема 1.4. *Пусть C – циклический код, порожденный многочленом $g(x) = g_0+g_1x+\cdots+g_rx^r$. Тогда код C имеет порождающую матрицу размера $(n-r)\times n$*

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \dots & g_r & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & \dots & g_r & 0 & \dots & 0 \\ 0 & 0 & g_0 & \dots & \dots & \dots & g_r & \dots & 0 \\ & & & \ddots & & & & \ddots & \\ 0 & 0 & 0 & \dots & g_0 & \dots & \dots & \dots & g_r \end{pmatrix}$$

и его размерность равна

$$\dim(C) = n - r$$

Доказательство. Заметим, что g_0 отличен от нуля. Так как иначе вектор $(0, g_1, \ldots, g_r, 0, \ldots, 0)$ содержится в C, следовательно и вектор $(g_1, \ldots, g_r, 0, \ldots, 0)$, полученный сдвигом, принадлежит C.

Учитывая, соответствие векторов многочленам в S, получим, что мы нашли ненулевой многочлен в C меньшей степени чем $g(x)$. Это противоречит минимальности степени образующего многочлена.

Таким образом, ранг матрицы G равен $n-r$. Осталось показать, что G – порождающая матрица.

Если $c(x) = a(x)g(x) = (a_0 + a_1x + a_2x^2 + \ldots + a_{n-r-1}x^{n-r-1})(g_0 + g_1x + \cdots + g_rx^r)$ кодовый многочлен в C, то

$$c(x) = a_0g(x) + a_1xg(x) + a_2x^2g(x) + \ldots + a_{n-r-1}x^{n-r-1}g(x).$$

Многочлену $x^kg(x)$ соответствует $k+1$ строка матрицы G. Таким образом, все элементы C являются линейной комбинацией строк матрицы G.

Определение 1.5. Пусть C — циклический код с порождающим многочленом $g(x)$ и

$$x^n - 1 = g(x)h(x),$$

тогда $h(x)$ называется *проверочным многочленом* кода C.

Теорема 1.6. *Пусть C есть циклический код с проверочным многочленом*

$$h(x) = h_0 + h_1x + \ldots + h_kx^k,$$

Тогда проверочная матрица кода C имеет вид

$$H = \begin{pmatrix} h_k & h_{k-1} & h_{k-2} & \ldots & h_0 & 0 & 0 & \ldots & 0 \\ 0 & h_k & h_{k-1} & h_{k-2} & \ldots & h_0 & 0 & \ldots & 0 \\ 0 & 0 & h_k & h_{k-1} & h_{k-2} & \ldots & h_0 & \ldots & 0 \\ & & & \ddots & & & & \ddots & \\ 0 & 0 & 0 & \ldots & h_k & h_{k-1} & h_{k-2} & \ldots & h_0 \end{pmatrix}$$

Упражнение 1. Докажите утверждение.

Пример 1.7. Пусть C – бинарный циклический код длины 7 с порождающим многочленом $g(x) = x^3+x+1$. Тогда $h(x) = (x^7-1)/(x^3+x+1) = x^4+x^2+x+1$ проверочный многочлен.

Порождающая матрица

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

Проверочная матрица

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Заметим, что это $[7, 4, 3]$ код Хэмминга.

§2. Примеры циклических кодов

В этом модуле мы покажем, что коды Хэмминга и обычный код Рида-Соломона – циклические. Отметим, что коды БХЧ тоже являются циклическим. Это сразу следует из способа построения этих кодов, приведенному в лекциях.

Справедлива следующая теорема.

Теорема 2.1. *Двоичный код Хэмминга эквивалентен циклическому коду*

Доказательство. Рассмотрим неприводимый многочлен $p(x)$ степени m над полем $\mathbb{F}_2 = GF(2)$.

Факторкольцо $\mathbb{F}_2[x]/p(x)$ полиномов по модулю $p(x)$ является полем и содержит 2^m элементов. Из теории полей известно, что существует элемент α поля $GF(2^m) = \mathbb{F}_2/p(x)$ такой, что

$$\mathbb{F}_2[x]/p(x) = \{0, 1, \alpha, \alpha^2, \ldots \alpha^{2^m-2}\}.$$

Этот элемент называется примитивным элементов и его порядок равен $2^m - 1$.

Рассмотрим матрицу

$$\widetilde{H} = (1, \alpha, \alpha^2, \ldots \alpha^{2^m-2})$$

над полем $\mathbb{F}_2[x]/p(x)$.

Так как поле $\mathbb{F}_2[x]/p(x)$ является m-мерным векторным пространством над полем $\mathbb{F}_2$ с базисом $1, x, \ldots, x^{m-1}$, то элементы α^i представимы в виде

$$\alpha^i = a_{i,0} + a_{i,1}x + \ldots + a_{i,m-1}x^{m-1},$$

где $a_{ik} \in \mathbb{F}_2$.

Обозначим $n = 2^m - 1$ и рассмотрим матрицу

$$H = \begin{pmatrix} a_{0,0} & a_{1,0} & a_{2,0} & \ldots & a_{n-1,0} \\ a_{0,1} & a_{1,1} & a_{2,1} & \ldots & a_{n-1,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{0,m-1} & a_{1,m-1} & a_{2,m-1} & \ldots & a_{n-1,m-1} \end{pmatrix}.$$

H это $m \times (2^m - 1)$ матрица над полем $\mathbb{F}_2$ и она является в некотором смысле представлением матрицы $\widetilde{H}$.

Рассмотрим теперь линейный код C с проверочной матрицей H.

Сначала убедимся, что это код Хэмминга. Действительно, так как все элементы α^i различны, то все столбцы матрицы H различные и ненулевые. Так как их $2^m - 1$, то это в точности все ненулевые столбцы высоты m.

Покажем, что C циклический код. Элемент $c = (c_0, c_1, \ldots, c_{n-1}) \in C \leftrightarrow Hc^t = 0$.

Матричное равенство $Hc^t = 0$ эквивалентно тому, что полином $c(x)$ обращается в ноль при подстановке $x = \alpha$ внутри $\mathbb{F}_2[x]/p(x)$. Итак получили критерий принадлежности слова коду C:

$$C = \{(c_0, c_1, \ldots, c_{n-1}) \mid c_0 + c_1\alpha + \ldots c_{n-1}\alpha^{n-1} = 0\} =$$

$$= \{c(x) \in \mathbb{F}_2[x]/(x^n - 1) \mid c(\alpha) = 0 \in \mathbb{F}_2[x]/p(x)\}.$$

Теперь мы можем доказать, то C – идеал в $\mathbb{F}_2[x]/(x^n - 1)$, а значит C – циклический код. Пусть $c(x) \in$, $f(x) \in \mathbb{F}_2[x]/(x^n - 1)$. Очевидно, что

$$c(\alpha)f(\alpha) = 0$$

и, поэтому, $c(x)f(x) \in C$. □

Покажем, что специальный вид кодов Рида-Соломона, так называемый обычный код Рида-Соломона, является циклическим.

Обычный код Рида-Соломона задается проверочной матрицей

$$H = \begin{pmatrix} 1 & \alpha^b & \alpha^{2b} & \dots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \alpha^{2(b+1)} & \dots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \alpha^{b+d-2} & \alpha^{2(b+d-2)} & \dots & \alpha^{(n-1)(b+d-2)} \end{pmatrix},$$

где α – примитивный элемент поля, d – минимальное кодовое расстояние, b – произвольная степень.

Теорема 2.2. *Обычный код Рида-Соломона является циклическим.*

Доказательство. Слово $(c_0, c_1, \dots, c_{n-1})$ является кодовым тогда и только тогда, когда $c(\alpha^k) = 0$ для всех $k = b, b+1, \dots, b+d-2$. Множество многочленов, обращающихся в ноль на некотором множестве является идеалом, поэтому обычный код Рида-Соломона является циклическим. □

§3. Sage

Как мы установили в лекции все циклические коды длины n над полем F порождаются многочленами – делителями многочлена $x^n - 1$.

Разложение многочлена $x^n - 1$ над конечным полем F на неприводимые сомножители в Sage можно произвести с помощью функции factor.

Разложим, например, многочлен $f = x^4 + x^2 - 1$ над полем $GF(9)$.

In [1]:
```
#Определим GF(9)
S.<z>=GF(9)
# Определим кольцо многочленов над конечным полем
R = PolynomialRing(S,'x')
# Нужно определить переменную x как объект этого кольца
x = R.gen()
# Запишем многочлен, который будем раскладывать
f = x^4+x^2-1
# Операция разложения на неприводимы множители
f.factor()
```

Out [1]: $(x\hat{}2 + z) * (x\hat{}2 + 2 * z + 1)$

Здесь z элемент поля $GF(9)$, построенного как факторкольцо кольца многочленов $\mathbb{Z}_3[z]$ по неприводимому многочлену второй степени.

Пример. Найдите базис наименьшего циклического кода длины n=7, который содержит слово v = 1101000

In [2]:
```
n=7
w=[1,1,0,1,0,0,0]
P.<x> = PolynomialRing(GF(2),"x")
#v=vector(GF(2), w)
# Порождающий полином вида 1+x+0 x^2 + 1 x^3 + ...
g=1+x+x**3
# Наименьший циклический код
C = codes.CyclicCode(generator_pol=g, length=n); C
```

Out [2]: $[7, 4]$ Cyclic Code over $\mathrm{GF}(2)$

In [3]:
```
# Можно вывести все кодовые слова
C.list()
```

Out [3]: $[(0, 0, 0, 0, 0, 0, 0),$
$(1, 1, 0, 1, 0, 0, 0),$
$(0, 1, 1, 0, 1, 0, 0),$
$\ldots$
$(1, 0, 0, 1, 0, 1, 1)]$

In [4]:
```
#Сформируем пространство размерности n
V=VectorSpace(GF(2),n)
#Построим подпространство, содержащее все слова кода C
L=V.subspace(C);L
```

Out [4]:
```
Vector space of degree 7 and dimension 4
over Finite Field of size 2
Basis matrix:
[1 0 0 0 1 1 0]
[0 1 0 0 0 1 1]
[0 0 1 0 1 1 1]
[0 0 0 1 1 0 1]
```

§4. Упражнения

5.1. Рассмотрим линейные коды длины $n = 7$.
(a) Сколько циклических кодов можно построить с такой длиной слова? Найти все порождающие полиномы.
(b) Может ли $g(x) = 1+x^3+x^4$ быть порождающим полиномом циклического (7, 3) кода?
(c) Сформируйте систематическую порождающую матрицу для кода с порождающим полиномом $g() = 1 + x$.
5.2. Описать строение всех циклических $[7, 4]_2$ кодов. Указание: найти разложение многочлена $x^7 - 1$ над полем $GF(2)$. Найти среди них многочлены степени $3 = 7 - 4$.
5.3. Показать, что код с проверкой на четность является циклическим. Указание: рассмотреть многочлены $g(x) = x - 1$ и $h(x) = x^{n-1} + \ldots + x + 1$.
5.4. Закодировать в циклическом коде комбинации 1001, 1010, если образующий многочлен $g(x) = x^3 + x + 1$.
5.5. По заданному образующему многочлену $g(x) = x^4 + x^3 + 1$ построить образующую и проверочную матрицы.
5.6. Вычислите минимальное кодовое расстояние циклического кода длины $n = 23$, порожденного полиномом $x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$.
5.7. Найдите базис наименьшего циклического кода длины n, который содержит слово v.
a) $v = 1101000$, $n = 7$;
b) $v = 101010$, $n = 6$;
c) $v = 01010101$, $n = 8$.
5.8. Найдите порождающий многочлен циклического кода $C =< S >$, где $S = \{(0, 1, 0, 1), (1, 0, 1, 0), (1, 1, 0, 0)\}$.

Глава 6
Групповые коды

Эта глава посвящена групповым кодам. Подразумевается, что материал предназначен для работы на семинарах и будет разбираться студентами самостоятельно. Студентам будет необходимо разобрать статьи, выполнить вычисления в Sage. В конце главы приведены темы исследований, которые могут лечь в основу выпускной работы.

§1. Определение групповых кодов

Пусть $F = GF(q)$ – конечное поле, G – конечная группа. Групповой алгеброй FG называется множество формальных сумм

$$FG = \left\{ \sum_{g \in G} \alpha_g g \mid \alpha \in F, g \in G \right\}.$$

Сложение и умножение в FG определяется естественным образом:

$$\sum \alpha_g g + \sum \beta_g g = \sum (\alpha_g + \beta_g) g,$$

$$(\sum \alpha_g g)(\sum \beta_g g) = \sum_g (\sum_h \alpha_{gh^{-1}} \beta_h) g.$$

Элементы $\sum \alpha_g g$ и $\sum \beta_g g$ равны тогда и только тогда, когда $\alpha_g = \beta_g$ для всех $g \in G$.

Определение 1.1. Будем называть групповым кодом любой идеал групповой алгебры FG.

Определение 1.2. Если группа G является абелевой, то групповые коды называются абелевыми кодами.

Пример. Если группа $G =< a \mid a^n = 1 >$ является циклической порядка n, то в силу изоморфизма

$$FG \cong F[x]/ < x^n - 1 >,$$

групповые коды совпадают с циклическими.

Задача. Представляет интерес рассмотрение односторонних идеалов вместо двусторонних в качестве кодов.

Исследования групповых кодов начались с работ Бермана, опубликованных в 1967 году ([6], [7]), который рассматривал абелевы коды.

В последнее время активизировались исследования для некоммутативных групп, в частности, получены описания для групповых алгебр над группами диэдра (см., например, [1], [3]).

В этой главе мы рассмотрим алгебраическое описание групповых кодов в случае когда FG является классически полупростой алгеброй.

Напомним одну из основополагающих теорем для полупростых групповых алгебр.

Теорема 1.3. *(Машке) Пусть G – конечная группа, $|G| = n$, F – поле и его характеристика равна p. Групповая алгебра FG является классически полупростой тогда и только тогда, когда либо $p = 0$ либо $(n, p) = 1$.*

В этом случае $FG \cong \bigoplus_{i=1}^{k} M_{n_i}(T_i)$, где T_i – алгебры с делением над F.

Так как групповая алгебра FG конечна, то мы можем применить теорему Веддерберна.

Теорема 1.4. *(Веддерберн) Всякое конечное ассоциативное тело является полем.*

Таким образом, если характеристика поля F взаимно проста с порядком группы, то

$$FG \cong \bigoplus_{i=1}^{k} M_{n_i}(F_i),$$

где F_i – конечные расширения поля F.

Как хорошо известно, любой идеал в классически полупростой алгебре является суммой неприводимых идеалов. Эти идеалы порождаются примитивными центральными идемпотентами и допускают непосредственные вычисления для групп и полей малых порядков.

Пусть G – конечная группа, $|G| = n$, F – поле.

6.1. Докажите, что $\mathbb{C}G \cong \oplus_{i=1}^{k} Mat_{n_i}(\mathbb{C})$ и $n = n_1^2 + n_2^2 + \ldots + n_k^2$.

6.2. Докажите, что $\mathbb{R}G \cong \oplus_{i=1}^{k} Mat_{n_i}(T_i)$, T_i – это либо тело кватернионов, либо $\mathbb{C}$, либо $\mathbb{R}$.

6.3. Докажите, что если $F = GF(q)$, то $FG \cong \bigoplus_{i=1}^{k} M_{n_i}(F_i)$, где F_i – конечные расширения поля F.

6.4. Докажите, что если $F = GF(q)$ и G – абелева, то существуют такие центральные идемпотенты $e_1, e_2, \ldots, e_n$, такие что $FG \cong \bigoplus_{i=1}^{k}(FG)e_i \cong F_i e_i$, где $F_i \cong (FG)e_i$ – конечные расширения поля F.

6.5. Пусть $F = GF(q)$, $(n, q) = 1$. Докажите, что многочлен $x^n - 1$ не имеет кратных корней (в поле разложения).

6.6. Докажите следующую теорему.

Теорема 1.5. *Пусть $G =< g >$ циклическая группа порядка n над полем $GF(q)$ и $(n, q) = 1$. Пусть $x^n - 1 = (x-1)f_1(x)f_2(x)\ldots f_k(x)$ разложение на неприводимые сомножители $x^n - 1$ над полем $GF(p)$. Тогда имеет место изоморфизм:*

$$GF(p)G \cong GF(q) \oplus GF(q^{deg(f_1)}) \oplus GF(q^{deg(f_2)}) \ldots \oplus GF(q^{deg(f_k)}).$$

Примените эту теорему для описания строения следующих групповых алгебр.

Упражнение 1. Найти описание групповой алгебры FG над полем $F = GF(2)$ циклической группы G порядка $k = 15$.

Решение. Разложим многочлен $x^{15} - 1$ над полем F, применяя Sage.

$$x^{15} - 1 = (x+1) * (x^2+x+1) * (x^4+x+1) * (x^4+x^3+1) * (x^4+x^3+x^2+x+1).$$

Таким образом

$$FG \cong GF(2) \oplus GF(4) \oplus GF(16) \oplus GF(16) \oplus GF(16).$$

Упражнение 2. Найти описание групповой алгебры FG над полем $F = GF(3)$ циклической группы G порядка $k = 14$.

Упражнение 3. Найти описание групповой алгебры FG над полем $F = GF(5)$ циклической группы G порядка $k = 42$.

§2. Лабораторная работа

Эта лабораторная работа основана на статье Samir Assuena and César Polcino Milies *Good Codes From Dihedral Groups*, https://arxiv.org/abs/1506.03303, 2015

Рассмотрим группу диэдра

$$G = D_n =< a, b \mid a^n = 1, b^2 = 1, bab = a^{-1} >$$

и групповую алгебру FG над конечным полем F, состоящим из q элементов.

Нас будет интересовать частный случай: когда $n = p^m$, p – простое число, $(2p^m, q) = 1$.

Рассмотрим цепочку подгрупп

$$A = H_0 \supseteq H_1 \supseteq \ldots \supseteq H_m = \{1\},$$

где H_k подгруппа, порожденная a^{p^k}.

A также обозначает подгруппу, порожденную элементом a. Напомним, что подгруппа A – нормальная и G/A – циклическая группа порядка 2.

Пусть H – подгруппа группы G Обозначим через

$$\hat{H} = \frac{1}{|H|} \sum_{h \in H} h.$$

Упражнение 1. Покажите, что $\hat{H}$ – идемпотент.

Упражнение 2. Покажите, что если $B < C$ – подгруппы в G, то $\hat{C}\hat{B} = \hat{C}$.

Обозначим

$$e_0 = \hat{H}_0$$
$$e_k = \hat{H}_k - \hat{H}_{k-1} \quad (1 \leq k \leq m)$$

Справедлива следующая теорема (F. S. Dutra, R. A. Ferraz and C. Polcino Milies, Semisimple group codes and dihedral codes, Algebra and Disc. Math., 3 (2009), 28-48, Th 3.3)

Теорема 2.1. $\{\frac{1+b}{2}e_0, \frac{1-b}{2}e_0\} \cup \{e_k \mid 1 \leq k \leq m\}$ – *образуют множество всех центральных примитивных идемпотентов.*

Таким образом, мы можем перечислить все групповые коды. Напомним, что групповым годом мы назвали идеалы групповой алгебры.

Далее мы покажем (используя только вычисления в Sage), что односторонние идеалы могут дать коды с лучшими характеристиками.

Введем еще одно обозначение:

Пусть $e = e_k$ – один из идемпотентов $e_1, \ldots, e_m$. Обозначим

$$e_{11} = \left(\frac{1+b}{2}\right) e, \qquad e_{22} = \left(\frac{1-b}{2} e_0\right)$$

$$e_{12} = \left(\frac{1+b}{2}\right) a \left(\frac{1-b}{2}\right) e, \qquad e_{21} = 4[(a-a^{-1})e]^{-2} \left(\frac{1-b}{2}\right) a \left(\frac{1+b}{2}\right) e$$

$$f = e_{11} - e_{12}.$$

Справедлива следующая теорема (Prop. 2.5)

Теорема 2.2. *Множество*

$$\{f, af, a^2 f, \dots a^{\varphi(p^k)-1}\}$$

является базисом (одностороннего) идеала FGf (здесь φ – функция Эйлера).

Пример.

Рассмотрим $G = D_9$, $F = GF(11)$. Следующий код демонстрирует (не очень изящно, простым нахождением элементов с маленьким весом), что кодовые расстояния центральных кодов (то есть двусторонних идеалов) меньше 9

```
In [5]:
#Функция вычисления веса
def wt(x):
return len(str(x).split(sep='+'))
#Примитивные центральные идемпотенты
e0=H0
e11=(kG(1)+b)/R(2)*e0
e22=(kG(1)-b)/R(2)*e0
e1=H1-H0
e2=kG(1)-H1
#Некоторые веса элементов из минимальных
#двусторонних идеалов
print(wt(e11*(a**2)  - e11*a), wt(e22*(a**2)  - e22*a),
wt(e1), wt(e2))
```

```
Out [5]: 1 1 9 3
```

Обозначим $e = e_1 = H_1 - H_0$, $f = e_{11} - e_{22}$ и односторонний идеал

$$FGf$$

.

В этом случае идеал FGf порожден двумя базисными элементами $\{f, af\}$.

Вычислим минимальный вес этого идеала над полем $GF(11)$, используя Sage

In [6]:
```
#Групповая алгебра над группой диэдра
G = DihedralGroup(9)
R=GF(11)
kG = G.algebra(R)
#Порождающие группы a,b
[a, b] = G.gens()
```

Out [6]:

Вычислим H_0 и H_1.

In [7]:
```
#Сумма всех элементов <a>
H0=kG(1)
for k in range(1,9):
    H0=H0+a**k
#Так мы показываем, что это элемент из кольца z=R(9)
H0=H0/R(9)

#Сумма всех элементов <a^3>
H1=kG(1)
for k in range(1,3):
    H1=H1+a**(3*k)
H1=H1/R(3)
```

Out [7]:

Вычисляем идемпотенты

In [8]:
```
E1=H1-H0
E11=(kG(1)+b)/R(2)*E1
E12=(kG(1)+b)/R(2)*a*(kG(1)-b)/R(2)*E1
f=E11-E12
```

Out [8]:

Перебираем все элементы идеала FGf и записываем их в список s.

In [9]:
```
s=[]
```

```
for i in range(9):
    for j in range(9):
        s.append(kG(i)*f+kG(j)*a*f)
```

Out [9]:

Выведем элементы s.

In [10]:
```
s
```

Out [10]: [0, 4*() + 4*(2,9)(3,8)(4,7)(5,6) + 5*(1,2)(3,9)(4,8)(5,7) +
5*(1,2,3,4,5,6,7,8,9) + 2*(1,3)(4,9)(5,8)(6,7) +
2*(1,3,5,7,9,2,4,6,8) + 4*(1,4)(2,3)(5,9)(6,8) +
4*(1,4,7)(2,5,8)(3,6,9) + 5*(1,5)(2,4)(6,9)(7,8) +
5*(1,5,9,4,8,3,7,2,6) + 2*(1,6)(2,5)(3,4)(7,9) +
2*(1,6,2,7,3,8,4,9,5) + 4*(1,7)(2,6)(3,5)(8,9) +
4*(1,7,4)(2,8,5)(3,9,6) + 5*(1,8)(2,7)(3,6)(4,5) +
5*(1,8,6,4,2,9,7,5,3) + 2*(1,9,8,7,6,5,4,3,2) +
2*(1,9)(2,8)(3,7)(4,6),
...

Такое представление мы получим из-за того, что группа диэдра в Sage реализована как подгруппа группы перестановок. Это позволяет Sage эффективно производить вычисления.

В следующем фрагменте мы подсчитаем веса

In [11]:
```
l=[]
for t in s:
    l.append(len(str(t).split(sep='+')))
print(l)
```

Out [11]: [1, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 15, 15, 15, 18, 15, 15, 18,
18, 18, 15, 18, 15, 15, 15, 15, 18, 15, 15, 18, 15, 15, 18, 18, 18, 18, 15,
18, 15, 18, 15, 15, 18, 18, 18, 15, 15, 15, 15, 18, 18, 15, 18, 18, 18, 15,
18, 15, 18, 18, 15, 15, 18, 15, 15, 18, 18, 15, 15, 18, 15, 18, 15, 15, 18,
18, 18, 15, 15, 18]

Вычислим наименьший вес

In [12]:
```
from collections import Counter
cnt = Counter(l).most_common(10)
cnt
```

```
Out [12]: [(18, 44), (15, 36), (1, 1)]
```

Получили количество элементов различных весов.

Пара $(1, 1)$ здесь соответствует нулевому элементу. Такое представление обусловлено тем, как мы считаем веса. Нам пришлось представить элементы групповой алгебры как строки, а затем разбить их на отдельные элементы, используя в качестве разделителя знак '+'.

Таким образом, мы получили $[18, 2, 15]$-код.

Упражнение 3. Найдите минимальное расстояние в случае когда поле $GF(5)$, $GF(7)$, $GF(13)$.
Покажите, что если брать поле $GF(5)$ или $GF(7)$, то минимальное расстояние будет меньше 15.

Глава 7
Алгебраическая криптография

Появление и развитие интернета привело к массовому применению криптографии для защиты информации, передаваемой по открытым каналам связи. Само по себе использование криптографии насчитывает несколько тысяч лет, но широкое применение математики для построения и анализа стойкости криптосистем началось с работы К. Шеннона "Математическая теория криптографии" [11], опубликованной в 1949 г.

Криптосистемы разделяют на симметричные, то есть системы, в которых для шифрования и расшифрования применен один и тот же криптографический ключ, и асимметричные (криптосистемы с открытым ключом). В асимметричных системах используются открытый ключ, который передается по открытому каналу, и закрытый ключ, который применяется для расшифровки. Начало асимметричным шифрам было положено в работе У. Диффи и М. Хеллмана [12], опубликованной в 1976 г. Методы шифрования с отрытым ключом применяются как для собственно шифрования, так и для обмена ключами по открытым каналам связи для дальнейшего применения симметричного шифрования, или как средство аутентификации пользователей.

Одной из первых реализаций идеи асимметричного шифрования была криптосистема RSA (см., например, [13]), базирующаяся на вычислительной сложности задачи факторизации больших чисел. С тех пор ведутся поиски подходящих математических объектов для построения асимметричных криптосистем.

В конце 1990-х гг. сформировалось направление исследований, которое можно назвать алгебраической криптографией (см., например, [14], [15]). Особенностью алгебраической криптографии является то, что криптографические протоколы строятся с использованием таких абстрактных алгебраических структур (платформ), как, например, некоммутативные группы, полугруппы, кольца.

Исследования по алгебраической криптографии (так же как и по математической криптографии в целом) нацелены на построение тех или иных криптографических протоколов и на обоснование их криптостойкости, т.е. вычислительной сложности нахождения секретного ключа по тем или иным открытым данным (например, по открытому ключу).

Мы рассмотрим сначала некоторые примеры исторических шифров и их простые реализации в Sage, далее обсудим подходы построения асимметричных шифров.

§1. Криптосистемы симметричного шифрования

До второй половины 20-го в криптографии для шифрования и расшифрования использовали один и тот же криптографический ключ. Существует большое количество различных симметричных криптографических систем как современных так и представляющих исторический интерес. В этой главе мы приводим только простые примеры шифров в качестве иллюстрации методов, реализованных в Sage. Для нашего курса основной интерес представляют системы ассиметричного шифрования, которые более подробно рассматриваются в следующей главе.

§2. Исторические шифры

2.1. Шифр Цезаря

Традиционно начнем изложение с самого простого шифра – шифра Цезаря. Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется буквой находящейся на некоторое постоянное число позиций левее или правее него в алфавите. Шифр назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки со своими генералами.

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

$$y = (x + k) \mod n$$
$$x = (y - k + n) \mod n,$$

где x — символ открытого текста, y — символ шифрованного текста, n — мощность алфавита, а k — ключ.

Шифр Цезаря может быть легко взломан простым перебором всех вариантов.

В Sage нет библиотек, работающих с русским языком 'из коробки'. Далее приведен пример кода на Python, реализующий шифрование шифром Цезаря.

In [13]:
```
# Пример программы на Python для шифрования
# и дешифрования текста
# с использованием шифра Цезаря:
def encrypt(text, shift):
    encrypted_text = ""
#Переводим строку в верхний регистр
    text=text.upper()
      m=ord("Я")-ord("А")
    for char in text:
# Проверяем, является ли символ буквой
        if char.isalpha():
# Получаем ASCII-код символа
            ascii_code = ord(char)
# Определяем верхний или нижний регистр буквы
            is_upper = char.isupper()
# Применяем сдвиг к ASCII-коду символа
            shifted_ascii = (ascii_code - ord("А")
                             + shift
if is_upper else ascii_code - ord("Я") +
shift) % m
# Преобразуем ASCII-код обратно в символ
encrypted_char = chr(shifted_ascii + ord("А")
if is_upper else shifted_ascii + ord("Я"))
encrypted_text += encrypted_char
else:
# Если символ не является буквой,
# оставляем его без изменений
encrypted_text += char
return encrypted_text

def decrypt(text, shift):
    return encrypt(text, -shift)
```

Out [13]:

In [14]:
```
#Пример шифрования
text="аБВГД ЭЮЯ"
```

```
shift = 3
encrypted_text = encrypt(text, shift)
print("Зашифрованный текст:", encrypted_text)
decrypted_text = decrypt(encrypted_text, shift)
print("Расшифрованный текст:", decrypted_text)
```

Out [14]: Зашифрованный текст: ГДЕЖЗ БВГ
Расшифрованный текст: АБВГД ЭЮА

Если работать только с латинским алфавитом, то можно воспользоваться уже реализованными методами.

In [15]:
```
# AlphabeticStrings() для работы со строками,
# использующие латинский алфавит
# для русского языка в Sage не реализован
# plaintext/ciphertext alphabet
A = AffineCryptosystem(AlphabeticStrings()); A
```

Out [15]: Affine cryptosystem on Free alphabetic string monoid on A-Z

In [16]:
```
#Переводим все символы в верхний регистр,
#убирая не буквы
P = A.encoding("The affine cryptosystem generalizes
the shift cipher -999.");P
```

Out [16]: THEAFFINECRYPTOSYSTEMGENERALIZESTHESHIFTCIPHER

In [17]:
```
# аффинное шифрование, замена
# x -> a*x + b \mod m, a - должен быть обратим в Z_m
a, b = (9, 13)
C = A.enciphering(a, b, P); C
```

Out [17]: CYXNGGHAXFKVSCJTVTCXRPXAXKNIHEXTCYXTYHGCFHSYXK

In [18]:
```
# Шифр Цезаря, a =1
a, b = (1, 3)
C=A.enciphering(a, b, A.encoding("ABCDYZ"));C
```

Out [18]: DEFGBC

In [19]:
```
#Дешифровка
A.deciphering(a, b, C)
```

Out [19]: ABCDYZ

2.2. Шифры перестановки

Шифры перестановки — такие шифры, преобразования из которых приводят к изменению только порядка следования символов исходного сообщения.

Обычно открытый текст разбивается на отрезки равной длины и каждый отрезок шифруется независимо. Пусть, например, длина отрезков равна n и $\sigma \in S_n$ — взаимно-однозначное отображение множества $1, 2, \ldots, n$ в себя. Тогда шифр перестановки действует так: отрезок открытого текста $x_1, \ldots, x_n$ преобразуется в отрезок шифрованного текста $\sigma(x_1) \ldots \sigma(x_n)$.

В случае английского алфавита мы имеем 26! вариантов построения шифра. Современные вычислительные мощности позволяют применить простой перебор всех вариантов. Но и без современных компьютеров эти шифры достаточно легко взламывались частотным анализом.

2.3. Шифр Виженера

Шифр Виженера – метод полиалфавитного шифрования буквенного текста с использованием ключевого слова. Этот метод является простой формой многоалфавитной замены.

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Открытый текст разбивается на блоки некоторый длины n. Задается ключ – последовательность из n натуральных чисел: $a_1, a_2, \ldots, a_n$ (обычно это кодовое слово – для удобства запоминания). Затем в каждом блоке первая буква циклически сдвигается вправо по алфавиту на a_1 позиций, вторая буква – на a_2 позиций, ... , последняя – на a_n шагов. Часто используют уже готовую таблицу (Таблицу Виженера).

Реализация в Sage.

In [20]:
```
S = AlphabeticStrings()
# Второй параметр это длина блока
E = VigenereCryptosystem(S,14); E
```

Out [20]: Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 14

```
In [21]: #Ключ для шифрования, A - смещение на 0 позиций,
         # B - смещение на 1 позицию ...
         key = S('ABCDEFGHIJKLMN'); key
```

```
Out [21]: ABCDEFGHIJKLMN
```

```
In [22]: # Шифратор
         e = E(key);e
```

```
Out [22]: Cipher on Free alphabetic string monoid on A-Z
```

```
In [23]: M=e(S("THECATINTHEHAT"));M
```

```
Out [23]: TIGFEYOUBQOSMG
```

```
In [24]: #Дешифровка
         E.deciphering(key, M)
```

```
Out [24]: THECATINTHEHAT
```

Криптостойкость этого шифра выше чем криптостойкость шифров моноалфавитной замены, однако уже в середине 19-го века были разработаны методы взлома этой криптосистемы. С помощью Sage можно промоделировать взлом сообщений, но мы оставим эту тему для самостоятельного изучения студентов.

§3. Современные криптосистемы симметричного шифрования

Требования к современным криптографическим системам:

- зашифрованное сообщение должно поддаваться чтению только при наличии ключа;
- знание алгоритма шифрования не должно влиять на надежность защиты (принцип Керкгоффса);
- любой ключ из множества возможных должен обеспечивать надежную защиту информации;
- алгоритм шифрования должен допускать как программную, так и аппаратную реализацию.

Обычно требуется выполнение следующих свойств:

- Шифр должен быть стойким даже в случае если нарушителю известно достаточно большое количество исходных данных и соответствующих им зашифрованных данных.
- Число операций, необходимых для расшифровывания информации путем перебора всевозможных ключей должно иметь строгую нижнюю оценку и должно либо выходить за пределы возможностей современных компьютеров, с учетом возможности организации сетевых вычислений, или требовать создания дорогостоящих вычислительных систем.
- Незначительное изменение ключа или исходного текста должно приводить к существенному изменению вида зашифрованного текста. Надо сказать, что данное требование не выполняется по отношению практически ко всем шифрам донаучной криптографии.
- Структурные элементы алгоритма шифрования должны быть неизменными.
- Длина шифрованного текста должна быть раной длине исходного текста.
- Дополнительные биты, вводимые в сообщение в процессе шифрования, должны быть полностью и надежно скрыты в шифрованном тексте.
- Не должно быть простых и легко устанавливаемых зависимостей между ключами, последовательно используемых в процессе шифрования.

Существует множество алгоритмов симметричных шифров, существенными параметрами которых являются: стойкость, длина ключа, число раундов, длина обрабатываемого блока, сложность аппаратной/программной реализации, сложность преобразования.

Применение шифров регулируются государственными стандартами. Приведем некоторые названия таких шифров: AES (англ. Advanced Encryption Standard) – американский стандарт шифрования, ГОСТ 28147-89 — советский и российский стандарт шифрования, также является стандартом СНГ, DES (англ. Data Encryption Standard) — стандарт шифрования данных в США, RC4 (алгоритм шифрования с ключом переменной длины) и многие другие. В Sage реализованы примеры шифров Simplified DES, Mini-AES, DES, потоковые шифры (Stream Cryptosystems).

§4. Упражнения

7.1. Зашифруйте сообщение "the house is being sold tonight"("дом продан сегодня вечером"), используя шифр Виженера с ключом "dollars". Игнорируй-

те пространство между словами. Расшифруйте сообщение, чтобы получить исходный текст.

7.2. Зашифруйте сообщение "Life is full surprises"("Жизнь полна сюрпризов"), используя шифр Виженера с ключом: "health". Игнорируйте пространство между словами. Расшифруйте сообщение, чтобы получить исходный текст.

7.3. Ева перехватила следующий зашифрованный текст. Используя статистическую атаку, найдите исходный текст:

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHS
PPEVWMXMWASVX-LQSVILY-
VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW

7.4. Шифротекст ЮВЧКЛ*ИКТЕЬОЮМП*ТВЕРТ*ЬСЕ получен в результате следующего алгоритма шифрования: открытый текст был записан в таблицу по строкам, после чего переставлены столбцы. Найти открытый текст.

7.5. Используйте атаку грубой силы, чтобы расшифровать следующее сообщение. Предположите, что Вы знаете: шифр — аффинный и исходный текст "ab"зашифрован "GL":

XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS

7.6. Используйте атаку частоты отдельных букв, чтобы расшифровать следующее сообщение. Предположите, что Вы знаете, что оно зашифровано с применением моноалфавитного шифра подстановки:

ONHOVEJHWOBEVGWOCBWHNUGBLHGBGR

Глава 8
Асимметричное шифрование

Проблема распределения ключей.

В симметричной криптографии каждая из сторон должна иметь копию общего ключа, что создает сложную проблему управления ключами. В криптосистемах с открытым ключом используются два ключа – открытый и секретный ключ.

Шифрование с открытым ключом

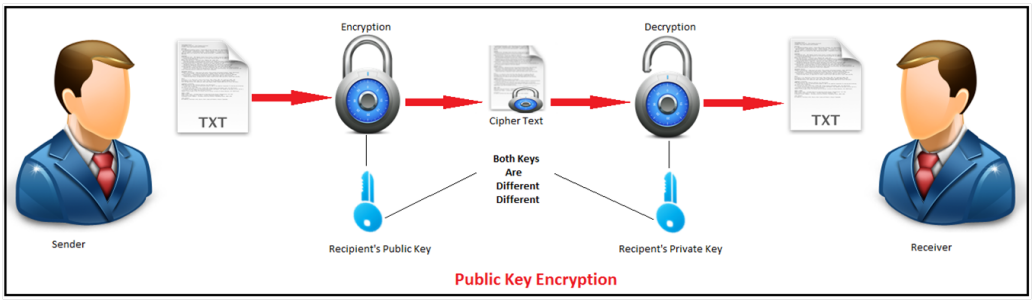

Рис. 1. Схема функционирования асимметричной криптосистемы.

Пусть пользователь Алиса опубликовал открытый ключ. Любой желающий написать Алисе может зашифровать сообщение с помощью ее открытого ключа. Алиса расшифровывает сообщение, применяя секретный ключ.

Впервые требования к криптосистемам с открытым ключом были опубликованы в работе Диффи и Хеллмана "Новые направления в криптографии" в 1976 году.

Требования к асимметричным системам(Диффи, Хеллман 1976 год)

- Вычисление пары ключей (открытого и закрытого) должно быть простым Отправитель, зная открытый (shared key) ключ sk легко вычисляет криптограмму $C = E_{sk}(M)$
- Получатель, используя закрытый (private key) ключ pk и криптограмму легко восстанавливает исходное сообщение $M = D_{pk}(C)$
- Противник, зная открытый ключ при попытке вычислить секретный наталкивается на непреодолимую проблему
- Противник, зная открытый ключ и криптограмму, при попытке восстановить исходное сообщение наталкивается на непреодолимую вычислительную проблему

Идея криптографии с открытым ключом очень тесно связана с идеей *односторонних функций*. Функция $f(x)$ называется односторонней, если она обладает следующим свойством: по известному x довольно просто[1] найти значение $f(x)$, тогда как определение x из $f(x)$ невозможно за разумный срок.

Сама по себе односторонняя функция бесполезна в применении: ею можно зашифровать сообщение, но расшифровать нельзя. Поэтому криптография с открытым ключом использует *односторонние функции с лазейкой*. Лазейка — это некий секрет, который помогает найти x, то есть существует такой y, что, зная $f(x)$ и y, можно вычислить x.

§1. Криптосистема RSA

Первым примером реализации криптосистемы с отрытым ключом стала система RSA (аббревиатура от фамилий Rivest, Shamir и Adleman), основанная на вычислительной сложности задачи разложения числа на простые сомножители. Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи.

Формирование открытого и секретного ключа

Алиса подбирает два больших простых числа p и q. Держа их в секрете, Алиса публикует

$$N = pq.$$

Кроме того Алиса выбирает шифрующую компоненту E, удовлетворяющую условию:

$$\text{НОД}(E, (p-1)(q-1)) = 1.$$

Пара (N, E) составляет **открытый ключ** Алисы.

Для выбора секретного ключа Алиса применяет расширенный алгоритм Евклида к паре $E, (p-1)(q-1)$, получая при этом расшифровывающую экспоненту d, такую, что

$$Ed = 1 \mod (p-1)(q-1).$$

d – **секретный ключ** Алисы.

Шифрование/Расшифрование

[1] обычно это означает, что $f(x)$ вычисляется за полиномиальное время от длины записи x

Боб хочет послать сообщение m Алисе. Сообщение – это некоторое число $1 \leq m \leq N$. Шифротекст, которое Боб формирует на основе открытого ключа:

$$C = m^E \mod N.$$

Алиса, получив сообщение C дешифрует его по правилу

$$D(C) = C^d \mod N.$$

Теорема 1.1.

$$(M^E)^d = M \mod N.$$

Пример 1.2. Пусть $p = 5$, $p = 11$. Тогда $N = 5 * 11 = 55$ и $\varphi(N) = (p - 1)(p - 1) = 4 * 10 = 40$.
Открытые ключи $N = 55$, $e = 7$. Секретный ключ: $d = 23$.
Шифровка сообщения $m = 6$:

$$C = m^7 = 6^7 \mod 55 = 63636 \mod 55 = 41 \mod 55$$

Дешифровка сообщения:

$$C^{23} = 41^{23} \mod 55 = \ldots = 6 \mod 55.$$

Реализация алгоритма RSA в Sage

In [25]:
```
# Случайные простые числа
p, q = random_prime(2^64), random_prime(2^64)
n = p*q
# Кольцо вычетов
ZZn = IntegerModRing(n)
print(n)
```

Out [25]: 173203645151859621695964915347765032049

In [26]:
```
# Функция Эйлера от n
r = (p-1)*(q-1)
ZZr = IntegerModRing(r)
# Поиск обратимого по модулю r элемента
# Можно выбрать случайное e = ZZ.random_element(r)
# В примере берем маленький показатель
# в качестве открытого ключа
E=5
while gcd(E, r) != 1:
```

```
E=E+1
#E = ZZ.random_element(r)
print("Шифрующая экспонента", E)
d = ZZr(e)^-1;
# d=~ZZr(e)
print("Дешифрующая экспонента", d)
```

Out [26]: Шифрующая экспонента 7
Дешифрующая экспонента 98973511515348355239688743849024820663

In [27]:
```
# Выберем случайное сообщение
m = ZZn.random_element(); print("message ", m)
# Закодируем его
s = m^E; print("encrypted message ", s)
print("decrypted message ", s^d)
s^d == m
```

Out [27]: message 884379467276786285557229122773394925 37
encrypted message 10613068981098049901656086569712344 8056
decrypted message 884379467276786285557229122773394925 37
True

§2. Алгоритмы цифровой подписи RSA

Криптосистема RSA позволяет реализовать цифровую подпись. Рассмотрим самую простую версию такой подписи.

Пусть Алиса хочет подписать документ m. Для этого она использует свой закрытый ключ d. Для создания подписи, обозначаемой s, Алиса вычисляет

$$s = m^d \text{ н.о.д. } n.$$

Пара (s, m) состоит из подписи и открытого текста.

Для проверки неизменности сообщения от Алисы с помощью электронной подписи Боб использует подпись и открытый ключ Алисы. Вычисляя

$$m' = s^E \mod n,$$

Боб может убедится, что сообщение послано Алисой и текст не менялся.

§3. Упражнения

8.1. Зашифруйте сообщение ATTACK, используя криптосхему RSA с $n = 43 * 59$ и $E = 13$. Каждая буква преобразуется в ее номер в алфавите (A-00, B-01, ..., Z-25).

8.2. Дешифруйте сообщение 0667 1947 0671, зашифрованное криптосхемой RSA с $n = 43 * 59$ и $E = 13$.

8.3. Дешифруйте сообщение 185 2038 2460 2550, зашифрованное криптосхемой RSA с $n = 53 * 61$ и $E = 17$.

8.4. Доказать, что в система RSA с модулем 21 и 35 все возможные ключи шифрования e совпадают с ключами дешифрования d.

8.5. Для электронной подписи используется криптосистема RSA. Секретный ключ Боба составляют числа p_B и q_B, а секретный ключ Алисы числа p_A и q_A. Пусть открытыми ключами Боба и Алисы являются пары $n_B = \{p_B q_B; eB\}$ и $n_A = \{p_A q_A; e_A\}$ соответственно. Необходимо передать Бобу секретное поручение m от Алисы, а также удостовериться в подлинности данного сообщения. Параметры криптосистемы: $p_A = 11, q_A = 23, e_A = 31, p_B = 7, q_B = 13, e_B = 5, m = 41$.

8.6. Для электронной подписи используется криптосистема RSA. Секретный ключ Боба составляют числа p_B и q_B, а секретный ключ Алисы числа p_A и q_A. Пусть открытыми ключами Боба и Алисы являются пары $n_B = \{p_B q_B; eB\}$ и $n_A = \{p_A q_A; e_A\}$ соответственно. Необходимо передать Бобу секретное поручение m от Алисы, а также удостовериться в подлинности данного сообщения. Параметры криптосистемы: $p_A = 7, q_A = 11, e_A = 7, p_B = 3, q_B = 5, e_B = 7, m = 13$.

§4. Алгоритм Диффи-Хеллмана распределения ключей

Алгоритм Диффи-Хеллмана не применяется для шифрования сообщений или формирования электронной подписи. Его назначение – в распределении ключей.

Он позволяет двум или более пользователям обменяться без посредников ключом, который может быть использован затем для симметричного шифрования.

Это была первая криптосистема, которая позволяла защищать информа-

цию без использования секретных ключей, передаваемых по защищенным каналам.

Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в мире шифрования, так как снимала основную проблему классической криптографии – проблему распределения ключей

Протокол Диффи-Хеллмана

Алиса и Боб выбирают группу G и элемент g, лежащий в этой группе. Каждый из них выбирает секретный параметр (натуральное число) x (Алиса) и y (Боб).

Далее они пересылают друг другу

$$A = g^x \text{ и } B = g^y$$

Алиса возводит B в степень x, Боб возводит A в степень y. Таким образом они получают общий ключ g^{xy} для обмена сообщениями.

Если в группе G вычисление x по g^x вычислительно сложная задача, то мы получим криптостойкую систему обмена ключами.

Изначально в схеме бралась группа $\mathbb{F}_p^*$.

§5. Криптосистема Эль-Гамаля

Эта система шифрования с открытым ключом опубликована в 1985 году Эль-Гамалем. Криптостойкость протокола основана на сложности решения задачи дискретного логарифмирования.

Пусть G – группа, $g \in G$. Сообщения M кодируется элементом из G. Пользователь(Алиса) выбирает секретный ключ x и публикует открытый ключ

$$< G, g >, \qquad y = g^x.$$

Сообщение, которое отправляется Алисе, шифруется следующим образом:

- Выбирается сессионный ключ k – случайное целое число.
- Вычисляются элементы $a = g^k$ и $b = y^k M$.

Пара чисел (a, b) является шифротекстом.

Расшифровка осуществляется на основе формулы

$$\frac{b}{a^x} = \frac{(g^x)^k M}{(g^k)^x} = M$$

5.1. Реализация на группе $\mathbb{Z}_p^*$

Изначально в качестве группы G бралась мультипликативная группа $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, где p – большое простое число, содержащее не менее 1024 двоичных разрядов. В группе $\mathbb{Z}_p^*$ выбирается элемент g достаточно большого порядка (генератор).

Процедура шифрования в криптосистеме Эль-Гамаля состоит из следующих операций.

1) **Создание пары из закрытого и открытого ключей стороной A.**
   1) Алиса выбирает простое случайное число p и генератор g.
   2) Алиса выбирает $x \in (1, p-1)$ с помощью генератора случайных чисел.
   3) Вычисляет $y = g^x \mod p$.
   4) Алиса создает закрытый и открытый ключи sk и pk:

$$sk = (p, g, x),\ pk = (p, g, y).$$

   Криптостойкость задается битовой длиной параметра p.
2) **Шифрование открытым ключом**
   1) Бобу известен открытый ключ Алисы $sk = (p, g, y)$.
   2) Сообщение представляется числом $m \in [0, p-1]$.
   3) Боб выбирает случайное число $r \in (1, p-1)$ и вычисляет:

$$\begin{aligned} a &= g^r \mod p, \\ b &= m \cdot y^r \mod p. \end{aligned}$$

   4) Боб посылает шифрованное сообщение в виде

$$C = (a, b)$$

   и посылает Алисе.
3) **Расшифрование**

   Получив сообщение (a, b) Алиса вычисляет:

$$m = \frac{b}{a^x} \mod p.$$

Шифрование корректно, так как

$$\begin{aligned} m' &= \tfrac{b}{a^x} = \tfrac{my^r}{g^{rx}} = m \mod p, \\ m' &\equiv m \mod p. \end{aligned}$$

Чтобы криптоаналитику получить исходное сообщение m из шифртекста (a, b), зная только открытый ключ получателя $sk = (p, g, y)$, нужно вычислить значение $m = b \cdot y^{-r} \mod p$. Для этого криптоаналитику нужно найти случайный параметр $r = \log_g a \mod p$, то есть вычислить дискретный логарифм. Такая задача является вычислительно сложной.

5.2. Реализация схемы Эль-Гамаля на эллиптических кривых в Sage

В качестве группы в схеме Эль-Гамаля можно выбрать различные группы. Одним из вариантов является использование группы эллиптических кривых над конечным полем.

В этом параграфе мы рассмотрим числовой пример, который базируется на одном из упражнений [16].

Алгоритм шифрования. В открытом доступе публикуются параметры группы эллиптической кривой, то есть $E_p(a, b) = \{(x, y) \mid y^2 = x^3 + ax + b, \quad x, y \in \mathbf{F}_p\}$ и точки-основания G на ней. Каждый из пользователей выбирает случайное целое число x, которое держит в секрете, затем вычисляет и делает общедоступной точку $x * G$.

Пусть $Pb = nb * G$ – открытый ключ Боба (nb – секретное число, придуманное Бобом).

Чтобы послать Бобу сообщение M, Алиса выбирает случайно целое число k и посылает пару точек $(k * G, M + k * Pb) = (C1, C2)$.

Чтобы прочитать сообщение, Боб умножает первую точку из полученной пары на свое секретное число nb и вычитает результат умножения из второй точки: $C2 - nb * C1 = M$.

Буквы кодируются элементами группы $E_p(a, b)$.

Упражнение. Используется кривая $E_{751}(-1, 1)$ и генерирующая точка $G = (-1, 1)$. Зная секретный ключ $nb = 29$, расшифровать сообщение M = [(440, 539), (128, 672)], [(489, 468), (282, 341)], [(489, 468), (45, 720)], [(72, 254), (227, 299)], [(188, 93), (251, 506)], [(72, 254), (319, 518)], [(745, 210), (129, 659)], [(286, 136), (515, 684)], [(568, 355), (395, 414)].

Решение.

In [28]:
```
#Задание группы эллиптической кривой
#над поле из 751 элемента
EC=EllipticCurve(GF(751),[-1,1]); EC
```

Out [28]: Elliptic Curve defined by y**2 = x**3 + 750*x + 1 over Finite Field of size 751

In [29]:
```
# Задание точек кривой
G = EC(0,1); G
```

Out [29]: $(0 : 1 : 1)$

In [30]:
```
# Символы алфавита закодированы элементами группы
# (кодирование задано в примере)
alpha={(228, 271):"а", (228, 480):"б",
(229, 151):"в", (229, 600):"г",
(234, 164):"д", (234, 587):"е",
(235, 19):"ж", (235, 732):"з",
(236, 39):"и", (236, 712):"й",
(237, 297):"к", (237, 454):"л",
(238, 175):"м", (238, 576):"н",
(240, 309):"о", (240, 442):"п",
(243, 87):"р",  (243, 664):"с",
(247, 266):"т", (247, 485):"у",
(249, 183):"ф", (249, 568):"х",
(250, 14):"ц", (250, 737):"ч",
(251, 245):"ш", (251, 506):"щ",
(253, 211):"ъ", (253, 540):"ы",
(256, 121):"ь", (256, 630):"э",
(257, 293):"ю", (257, 458):"я"}
```

Out [30]:

In [31]:
```
#Схема шифрования C = (C1, C2) = {kG, M + kPb}
def MyEncoder(G, Pb, k, M):
    return (k*G, M+k*Pb)
```

Out [31]: .

In [32]:
```
# Схема декодирования
# M = (M + kPb) - nb*(kG)  = C2 - nb*C1
def MyDecoder(C1, C2, nb):
     return C2-nb*C1
```

Out [32]: .

In [33]:
```
# Используется кривая E(751, -1,1)
# и генерирующая точка G = (-1, 1)
# Зная секретный ключ nb, расшифровать сообщение M
# Закрытый ключ
nb=29
# Шифртекст
Message=[ [(440, 539), (128, 672)],
[(489, 468), (282, 341)],
[(489, 468), (45, 720)],
[(72, 254), (227, 299)],
[(188, 93), (251, 506)],
[(72, 254), (319, 518)],
[(745, 210), (129, 659)],
[(286, 136), (515, 684)],
[(568, 355), (395, 414)]]
```

Out [33]: .

In [34]:
```
message=[]
for x in Message:
C2=EC(x[1])
C1=EC(x[0])
DM=MyDecoder(C1, C2, nb)
message.append(alpha[(DM[0], DM[1])])
print(DM[0], DM[1], alpha[(DM[0], DM[1])])
```

Out [34]: 228 480 б
228 271 а
243 87 р
249 568 х
228 271 а
247 266 т
238 576 н
253 540 ы
236 712 й

In [35]:
```
# Выведем в виде одной строки
"".join(message)
```

Out [35]: 'бархатный'

Дополнительные замечания.

Для построения новых примеров/упражнений надо уметь порождать группы эллиптических кривых, кодировать символы используемого алфавита элементами группы.

In [36]:
```
# Породим группу
EC0=EllipticCurve(GF(101),[2,3]);EC0
```

Out [36]: Elliptic Curve defined by y**2 = x**3 + 2*x + 3 over Finite Field of size 101

In [37]:
```
#Сформируем группу
Gr=[x for x in EC0]
```

Out [37]: .

In [38]:
```
# Формирование словаря
# Нумеруем буквы, начиная с 4-го элемента группы
rus_alph="абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
alpha0={}
for k in range(4,37):
#print(EC0(Gr[k]))
D=Gr[k]
alpha0[(D[0],D[1])]=rus_alph[k-4]
```

Out [38]: .

In [39]:
```
print(alpha0)
```

Out [39]: (3, 95): 'а', (5, 21): 'б', (5, 80): 'в', (9, 12): 'г',
(9, 89): 'д', (10, 35): 'е', (10, 66): 'ё', (11, 12): 'ж',
(11, 89): 'з', (13, 2): 'и', (13, 99): 'й', (17, 1): 'к',
(17, 100): 'л', (18, 35): 'м', (18, 66): 'н', (20, 8): 'о',
(20, 93): 'п', (21, 32): 'р', (21, 69): 'с', (23, 46): 'т',
(23, 55): 'у', (25, 15): 'ф', (25, 86): 'х', (27, 34): 'ц',
(27, 67): 'ч', (30, 46): 'ш', (30, 55): 'щ', (35, 15): 'ъ',
(35, 86): 'ы', (40, 7): 'ь', (40, 94): 'э', (41, 15): 'ю',
(41, 86): 'я'

Глава 9
Приложения

§1. Конечные поля

Определение 1.1. Пусть F есть множество с двумя бинарными операциями $+$ и $\cdot$. F называется полем, если
1) F есть абелева группа по сложению $+$.
2) $F^* = F \setminus \{0\}$ есть абелева группа по умножению $\cdot$
3) Выполняется дистрибутивность для всех a, b и c из F

$$a(b+c) = ab + ac$$

$$(a+b)c = ac + bc$$

Если число элементов F конечно, то F называется конечным полем.

Теорема 1.2. *Кольцо вычетов $\mathbb{Z}_p$ поле тогда и только тогда, когда p – простое число.*

Определение 1.3. Поле E называется расширением поля F, если F подполе E. Обозначения: $F < E$, E/F.

Определение 1.4. Если E – расширение F, то поле E можно рассматривать как векторное пространство над F. Степенью расширения называется размерность этого векторного пространства $[E : F]$.

Пример.

$$F = \mathbb{Q}(\sqrt{2}) = \{a + b\sqrt{2} \mid a, b \in \mathbb{Q}\}$$

Теорема 1.5. *Пусть F – поле, $F[x]$ – кольцо многочленов, $p(x)$ – неприводимый многочлен. Тогда факторкольцо $F[x]/ < p(x) >$ является полем.*

Теорема 1.6. *Пусть F – поле. Факторкольцо $F[x]/ < f(x) >$ является полем ттогда $f(x)$ – неприводимый многочлен.*

Определение 1.7. Элемент $\alpha \in E$ называется алгебраическим, если $f(\alpha) = 0$ для некоторого ненулевого полинома $f(x) \in F[x]$. Говорят, что полином $f(x)$ аннулирует элемент α. Многочлен минимальной степени, аннулирующий α назывется минимальным ногочленом.

Теорема 1.8. *Пусть $\alpha \in E$ -алгебраический элемент и $f(x)$ его минимальный многочлен. Тогда $f(x)$ неприводим.*

Теорема 1.9 (Конструкция присоединения корня)**.** *Пусть F – поле и $f(x) \in F$ неприводимый многочлен. Тогда факторкольцо $E = F[x]/ < f(x) >$ – поле и F можно вложить в E при помощи отображения $\alpha \to \alpha + < f(x) >$.*

Учитывая вложение, поле E является конечным расширением поля F, его степень $[E : F] = \deg(f(x))$ и многочлен $f(x)$ имеет корень в E.

Определение 1.10. Наименьшее расширение поля F, содержащее все корни многочлена $f(x)$ называется полем разложения многочлена $f(x)$.

Теорема 1.11. *(без доказательства) Поле разложения многочлена $f(x)$ строится однозначно с точностью до F-изоморфизма.*

Пусть F – поле, $f(x) \in F[x]$ – многочлен степени n. Многочлен $f(x)$ называется сепарабельным, если он имеет ровно n корней (в своем поле разложения), то есть многочлен раскладывается в произведение n различных линейных множителей.

Расширение E/F называется сепарабельным расширением поля F, если каждый элемент из E является корнем сепарабельного многочлена из $F[x]$.

Теорема 1.12. *Пусть F – поле, $f(x) \in F[x]$. Многочлен $f(x)$ сепарабельный ттогда $(f, f') = 1$.*

§2. Строение конечных полей

Теорема 2.1. *Пусть F – конечное поле. Тогда характеристика поля F равна некоторому простому числу p.*

Теорема 2.2. *Пусть F – конечное поле характеристики p. Тогда количество элементов в F равно p^n для некоторого натурального n.*

Теорема 2.3. *Пусть F – конечное поле характеристики p. Тогда*

$$(a + b)^{p^n} = a^{p^n} + b^{p^n}.$$

Теорема 2.4. *Для каждого простого числа p и натурального n существует (и единственное с точностью до изоморфизма) конечное поле порядка p^n.*

Доказательство. Рассмотрим поле разложения многочлена $x^{p^n} - x$ над $\mathbb{Z}_p$.

Многочлен $x^{p^n} - x$ сепарабелен, то есть все его корни различны. Множество всех его корней образует подполе в поле разложения. Порядок этого подполя равен p^n. □

Теорема 2.5. *Если поле F состоит из q элементов, то каждый элемент поля F является корнем многочлена $x^q - x$.*

Доказательство. Очевидно, что ноль является корнем рассматриваемого многочлена. Рассмотрим ненулевые элементы. Так как мультипликативная группа поля F состоит из $q-1$ элемента, то по теореме Лагранжа $x^{q-1} = 1$ для любого ненулевого $x \in F$. □

Конечные поля обычно обозначают $GF(q)$.

Теорема 2.6. *Поле $GF(p^n)$ содержит $GF(p^m)$ в качестве подполя тогда и только тогда, когда m делит n.*

Доказательство. Если поле $GF(p^n)$ содержит $GF(p^m)$ в качестве подполя, то $GF(p^n)$ является линейным пространством над $GF(p^m)$, откуда следует, что p^n есть степень числа p^m. Отсюда следует, что m делит n.

Пусть, наоборот, m делит n. Тогда $xp^n - 1 = (p^m)^k 1 = (p^m - 1)t$ и, поэтому,

$$x^{p^n} - x = x(x^{p^n - 1} - 1) = x(x^{(p^m-1)t} - 1) = x(x^{(p^m-1)} - 1)T = (x^{p^m} - x)T$$

Таким образом, многочлен $x^{p^m} - x$ делит многочлен $x^{p^n} - x$ Если рассмотреть все элементы поля $GF(p^n)$, которые являются корнями многочлена $x^{p^m} - x$ (их ровно p^m), то они образуют подполе (изоморфное $GF(p^m)$). □

Теорема 2.7. *Мультипликативная группа конечного поля является циклической.*

Доказательство. Предположим, что у конечного поля F из $q = p^n$ элементов мультипликативная группа не является циклической.

F^* – это абелева группа. Если она не является циклической, то рассмотрим наибольший порядок s элементов из F^*. Все порядки элементов делят s, так как иначе мы можем построить элемент с большим порядком [Упражнение: $o(a) = m$, $o(b) = n$, тогда $o(ab) =$ наименьшее общее кратное n и m]. Итак, существует число $s < q - 1$ такое, что $x^s = 1$ для любого $x \in F^*$. Это означает, что все элементы поля F являются корнями многочлена $x^{s+1} - x$.

Таким образом, у многочлена степени $< q$ есть q корней, что невозможно. □

Определение 2.8. Примитивным элементом конечного поля $GF(p^m)$ называется всякий первообразный корень степени $p^m - 1$, то есть всякий порождающий элемент мультипликативной группы этого поля.

Теорема 2.9. *Каждое конечное расширение E конечного поля F является простым расширением, то есть существует $\alpha \in E$, такое что $E = F(\alpha)$.*

§3. Конечные поля в Sage

```
In [40]: # Конечное поле характеристики 7.
         # Будет построено как факторкольцо F[a]/p(a)
         F.<a> = GF(7^15); F
```

```
Out [40]: Finite Field in a of size 7**15
```

В Sage полином p(a) выбирается из полиномов Конвея (минимальный полином примитивного элемента).

```
In [41]: # Отображение полинома p(a)
         F.polynomial()
```

```
Out [41]: a^15 + 5*a^6 + 6*a^5 + 6*a^4 + 4*a^3 + a^2 + 2*a + 4
```

Выведем подином Конвея.

```
In [42]: conway_polynomial(7, 15)
```

```
Out [42]: x^15 + 5*x^6 + 6*x^5 + 6*x^4 + 4*x^3 + x^2 + 2*x + 4
```

Выведем примитивный элемент поля F, то есть такой элемент, что мультипликативная группа $F^* =< \alpha >$

```
In [43]: F.primitive_element()
```

```
Out [43]: a
```

Вывод всех элементов можно осуществить, используя конструкцию F.list(), но вычисления, скорее всего, будут происходить долго в силу большого количества элементов. Еще один способ это вывести все степени примитивного элемента.

```
In [44]: #Вывести степени примитивного элемента
         print([a**k for k in range(1,25)])
```

```
Out [44]: [a, a^2, a^3, a^4, ...]
```

Еще один способ задать поле – построить факторкольцо.

```
In [45]: #Кольцо полиномов над Z_7 от переменной z
         P.<z>=Integers(7)[];
         #Идеал, порожденный z^2+z+3
         K = P.ideal(z^2+z+3)
         #Факторкольцо по идеалу
         H.<a> = P.quotient(K); H
```

```
Out [45]: Univariate Quotient Polynomial Ring
          in a over Ring of integers modulo 7
          with modulus z^2 + z + 3
```

```
In [46]: #Проверка неразложимости полинома
         (z^2+z+3).is_irreducible()
```

```
Out [46]: True
```

```
In [47]: #Проверка является ли H полем
         H.is_field()
```

```
Out [47]: True
```

§4. Упражнения

9.1. Постройте поле порядка 25. Разложите многочлен $x^{25} - x$ на множители над этим полем. Прокомментируйте результаты (сколько неприводимых сомножителей, почему столько ...).

9.2. Пусть α корень многочлена $f(x) = x^3 + x^2 + 1$ над $\mathbb{Z}_2$. Постройте поле $GF(8)$. Покажите, что f разлагается на линейные множители в поле $\mathbb{Z}_2(\alpha)$.

9.3. Постройте поле $GF(9)$ с помощью неприводимого полинома $f(x) = x^2 + 2x + 2 \in \mathbb{Z}_3[x]$. Проверьте, будет порождающий элемент примитивным.

9.4. Покажите, что каждый элемент в конечном поле $GF(25)$ является суммой квадратов.

9.5. Покажите, что каждый элемент в конечном поле является суммой квадратов.

§5. Варианты тем дипломных работ

В этом параграфе приведены возможные темы для самостоятельных исследований студентов.

1. В статьях [2], [3], [8] описано (используя различные техники) строение полупростых алгебр над группами диэдра. Исследовать другие классы конечных групп.

2. Рассмотреть групповые коды в случае модулярных групповых алгебр, то есть когда характеристика поля делить порядок группы.

3. Мы рассматривали групповые коды как двусторонние идеалы. Рассмотреть односторонние идеалы в качестве кодов. Исследовать какие результаты можно получить в этом случае, в частности, можно ли получить коды с лучшими характеристиками. Начать можно с 'численных' экспериментов - вычислить характеристики в Sage и попробовать выдвинуть гипотезы (см. также Лабораторную работу).

4. Построить криптосистему, основанную на кодах МакЭллис (см. [10])

Литература

1. *Blake I.F.* The Mathematical Theory of Coding / I.F. Blake, R.C. Mullin. – New York: Academic Press, 1975. – 356 pp
2. *Ferraz Raul Antonio* Idempotents in group algebras and minimal abelian codes/Raul Antonio Ferraz, César Polcino Milies // Finite Fields and Their Applications. – Vol. 13, Issue 2. – 2007. – P. 382-393.
3. *Dutra F.* Semisimple group codes and dihedral codes/Flaviana S. Dutra, Raul A. Ferraz and C. Polcino Milies // Algebra and Discrete Mathematics. – 2009. N 3. – P. 28 – 48.
4. *Adams S.S.* Introduction to Algebraic Coding Theory With Gap. Fall 2006/ S.S. Adams – URL: https://pi.math.cornell.edu/~web3360/eccbook2007.pdf (дата обращения: 15.01.2024)
5. *Абызов А.Н.* Кольца и модули : монография/ А. Н. Абызов, А. А. Туганбаев. – Москва: ФЛИНТА, 2017. —- 258 с.
6. *Berman S.D.* Semisimple cyclic and Abelian codes. II/ S.D Berman // Cybern Syst Anal. – Vol. 3. – 1967. – P. 17–23.
7. *Berman S.D.* On the theory of group codes/ S.D Berman // Cybern Syst Anal. Vol. 3, 1967. – P. 25–31.
8. *Brochero F.E.* Structure of finite dihedral group algebra/ Martínez F.E. Brochero //Finite Fields and Their Applications. – Vol. 35, 2015. – P. 204-214.
9. *Assuena S.* Good Codes From Dihedral Groups /Samir Assuena and César Polcino Milies. – URL: https://arxiv.org/abs/1506.03303 (дата обращения: 15.01.2024)
10. *Roering Ch.* Coding Theory-Based Cryptopraphy: McEliece Cryptosystems in Sage /Roering, Christopher – URL: https://digitalcommons.csbsju.edu/honors_theses/17 (дата обращения: 15.01.2024)
11. *Shannon C. E.* A Mathematical Theory of Communication/ C. E. Shannon // Bell System Technical Journal. – V. 27, 1948. – P. 379-423.
12. *Diffie W.* New directions in cryptography / Diffie, W., Hellman, M. //IEEE Trans. Inf. Theory., 1976, V. 22, No. 6. – P. 644–654.
13. *Ященко В.В.* Введение в криптографию. 4-е изд., доп. / В.В. Ященко

– Москва : МЦНМО, 2014. – 347 с.

14. *Романьков В.А.* Алгебраическая криптография: монография / В.А. Романьков. – Омск: Изд-во Омск. Гос ун-та, 2013. – 136 с.
15. *Myasnikov A.* Non-commutative Cryptography and Complexity of Group-theoretic Problems / A. Myasnikov, V. Shpilrain, A. Ushakov. – AMS, Mathematical Surveys and Monographs 177., 2011. – 385 p.
16. *Жданов О.Н.* Эллиптические кривые: Основы теории и криптографические приложения/ О.Н. Жданов, В.А. Чалкин. – URSS. 2020. – 200 с.