МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ.Н.И. ЛОБАЧЕВСКОГО
КАФЕДРА АЛГЕБРЫ И МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Специальность: 010101 математика

Специализация: алгебра

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(дипломная работа)

Построение конечных кодов в системе компьютерной алгебры Mathematica

Работа завершена:

Студент 05–902 группы математического отделения

_____ 2014г. _____ (И.Ф. Масгутов)

Работа допущена к защите:

Научный руководитель

к.ф.-м.н., доцент

_____ 2014г. _____ (М.Ф. Насрутдинов)

Заведующий кафедрой

д.ф.-м.н., профессор

_____ 2014г. _____ (М.М. Арсланов)

Казань – 2014

Оглавление

Введение

Несмотря на название, коды были придуманы для исправления ошибок переданного по каналу сообщения, а не для шифрования (этим занимается криптография). Например, при передаче сообщения по каналу из Казани в Москву могут возникнуть ошибки и переданное сообщение будет неверным. Задача теории кодирования - восстановить переданное сообщение.

Внимание акцентировалось на реализацию математических методов в компьютерной среде Wolfram Mathematica. Для него выделил раздел, в котором описал возможности, которые он предостовляет. Отдельно реализованы методы линейных и циклических кодов для пользователя и для изучения эффективности тех или иных методов, путем сравнения их времени работы в полях разных размеров. В пользовательской программе элементы матрицы вводятся вручную, а в режиме "анализа"заполняются случайным образом из введенного поля. Представляя циклические коды через алгебраические полиномы, можно работать с кодами в среде Wolfram Mathematica, а линейные коды легко реализуются через матрицы. Инструменты алгебры хорошо подходят для таких целей.

В теории рассматриваются методы нахождения линейных, циклических кодов через матрицы, обнаружение и исправление ошибок с помощью образующего многочлена, построение таблицы синдромов и соответствующих лидеров смежных классов. После каждого раздела представлен код, который наглядно демонстрирует методы пройденной главы и имеются подробные объяснения, каким образом происходит построение.

Современные компьютеры имеют два и более ядер, но не все программы используют всю вычислительную мощь машины. Wolfram Mathematica по умолчанию работает только с одним ядром, но удалось задействовать все ядра процессора и распараллелить исполняемый код. Таким образом, было подсчитано затраченное время при работе на одном ядре и на всех одновременно. Рассмотрены аспекты распараллеливания процесса и в работе имеется подробное их описание. Помимо всего этого, были написаны программы для ручного построения линейных и циклических кодов, т.е. пользователь вручную задает поле, размеры матриц и их элементы. В отдельной главе описал использованные процедуры пакета Mathematica. Так же в работе имеется план работ, чтобы было четкое представление о происходящем.

До того как начнем изучение глав, нужно понимать для чего все это делается и нужен план.

Теория кодирования занимается исправлением ошибок в сообщениях, которые передаются по каналу. В работе реализованы методы передачи сообщения по сымитированному каналу,

приема этого сообщения, анализа на наличие ошибки и в случае обнаружения, ее исправление, а для этого начальное сообщение представляется в виде кода и длина кода больше, чем длина сообщения. Итак, в системе комьютерной алгебры Wolfram Mathematica нужно построить код. До описания методов построения приведен необходимый теоретический минимум. После каждого раздела пройденный материал проиллюстрирован в Mathematica. Т.е. все операции (построение, передача, прием...) происходят пошагово. В работе рассмотрены два кода: циклический и линейный.

План работ:

1. Построить код или представить сообщения в виде кода
2. Найти минимальный вес, чтобы определить возможность нахождения ошибки
3. Построить таблицу синдромов для исправления ошибки
4. Передать код по сымитированному каналу
5. Получить код и провести анализ пользуясь минимальным весом
6. Исправить полученный код используя таблицу синдромов

После, работа всех представленных процессов проанализирована, причем анализ произведен в двух режимах работы программы: "обычный"и режим "с включенным распараллеливанием". Все этапы отражены в графике и описаны.

Линейные коды

В области математики и теории информации линейный код — это важный тип блокового кода, использующийся в схемах определения и коррекции ошибок. Линейные коды, по сравнению с другими кодами, позволяют реализовывать более эффективные алгоритмы кодирования и декодирования информации.

Начнем с того, что X – конечное поле $F_q, q = p^l,$ где p – простое число. В этом случае X^n можно рассматривать как n – мерное пространство над полем F_q. Его мы будем обозначать через F_q^n.

Интересным является случай $q = 2$. В этом случае пространство X^n называется двоичным линейным пространством Хэмминга.

В то же время естественно рассматривать и более общий случай: X – это конечная группа или конечное кольцо. В частности, наиболее широко рассматривался случай, в котором X – кольцо вычетов по модулю 4 или в несколько более общем случае X – кольцо Галуа. Заметим, что почти все рассматриваемые далее определения (линейный и двойственный коды, вес и многие другие), очевидным образом могут быть введены и в подобных пространствах X.

Определение 1. *Линейный код длины n это векторное подпространство C векторного пространства $X^n = \mathbb{F}_q^n = \{(\alpha_1, ..., \alpha_n) \mid \alpha_i \in \mathbb{F}_q\}$, где $\mathbb{F}_q$ — конечномерное поле из q элементов.*

Через $k = dimC$ обозначим размерность линейного кода C. Пусть $w = \{w_1, ..., w_k\}$ – базис пространства C. В теории кодирования принято называть матрицу $G = G(C)$ строками которой являются векторы $\{w_1, ..., w_k\}$, порождающей матрицей кода C.

Любой вектор x кода C может быть представлен в виде

$$x = zG,$$

где $z - k$ – мерный вектор пространства F_q^k.

Линейный код длины n и размерности k вкладывается как подпространство в векторное пространство X^n. Таким образом, слова кодового пространства $\mathbb{F}_q^n$ есть векторы, поэтому можно сказать, что кодовые слова (т.е. элементы C) кодовые векторы.

Двойственный код

Скалярное произведение $\langle x, y\rangle$ в поле $\mathbb{F}_q$ векторов $x = (x_1, ..., x_n)$ и $y = (y_1, ..., y_n)$ линейного пространства $\mathbb{F}_q^n$ мы определим следующим образом

$$\langle x, y\rangle = x_1y_1 + ... + x_ny_n \text{ все операции в поле } \mathbb{F}_q$$

Два вектора x, y называются ортогональными, если $\langle x, y\rangle = 0$.

Определение 2. ***(Двойственный код)*** *Код $C^\perp$ образованный всеми векторами, которые являются ортогональными ко все векторам кода C, называется двойственыым к коду C.*

Очевидно, что $C^{\perp\perp} = C$.

Определение 3. ***(Проверочной матрицей кода C)*** *Порождающая матрица $(n-k)\times n$ – матрица H кода $C^\perp$ называется проверочной матрицей кода C.*

Подобное название объясняется тем, что для каждого вектора x кода C выполнено

$$xH^T = 0.$$

В частности, произведение HG^T, GH^T является нулевой $k \times k$ матрицей.

Соотношение $xH^T = 0$ в теории кодирования принято рассматривать как набор $n - k$ проверок наложенных на координаты вектора x. Каждая проверка, определяемая одной из строк матрицы H, является однородным линейным уравнением, связывающих координаты вектора x.

Множество решений уравнения $xH^T = 0$ совпадает с кодом C. Имея в виду этот факт, говорят, что код C определяется проверочной матрицей H.

Лемма 1. *Код $^\perp$ является линейным кодом над полем $\mathbb{F}_q$ и имеет размерность $n - k$, где $k = dimC$.*

Доказательство. Очевидно, что если векторы x, y ортогональны вектору a, то их сумма с коэффициентами из $\mathbb{F}_q$ состоит из векторов x, которые ортогональны каждой строке проверочной матрицы H кода C.

Другими словами, векторы $x \in C^\perp$ являются решениями линейной системы однородных уравнений

$$xH^T = 0,$$

где H^T – транспонированная матрица H.

Как хорошо известно, множество решений однородной системы линейных уравнений с n неизвестными (координаты вектора x) и k уравнениями представляет собой линейное пространство размерности $n - k'$, если k' – ранг матрицы H. Ранг матрицы $k \times n$ – матрицы H равен k, ибо ее строками, по определению, являются базисные векторы пространства C.$\square$

Примеры

Для лучшего представления рассмотрим примеры (все операции будут выполняться над полем $\mathbb{F}_2 = \{0, 1\}$).

Первая часть кодового слова состоит из символов самого сообщения:

$$x_1 = u_1, x_2 = u_2, ..., x_k = u_k,$$

за которым следуют $n - k$ проверочных символов $x_{k+1}, ..., x_n$. Проверочные символы выбраны так, чтобы кодовые слова удовлетворяли уравнению

$$H \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = Hx^T = 0$$

где $((n-k) \times n)$ - матрица H, называемая проверочной матрицей кода, имеет вид

$$H = [A|I_{n-k}].$$

Здесь A - некоторая фиксированная $((n-k) \times k)$ - матрица из 0 и 1, а

$$I_{n-k} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

– единичная матрица размера $(n-k) \times (n-k)$. Все операции выполняются по модулю 2.

Пример 1. *Проверочная матрица*

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

определяет код с $k = 3$ *и* $n = 6$. *Для этого кода*

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Сообщение $u_1u_2u_3$ *кодируется в кодовое слово* $x = x_1x_2x_3x_4x_5x_6$, *которое начинается с самого сообщения* $x_1 = u_1; x_2 = u_2; x_3 = u_3$, *а последующих три проверочных символа* $x_4x_5x_6$ *выбираются так, чтобы выполнялось уравнение* $Hx^T = 0$, *т.е.*

$$\begin{cases} x_2 + x_3 + x_4 = 0 \\ x_1 + x_3 + x_5 = 0 \\ x_1 + x_2 + x_6 = 0 \end{cases}$$

Если сообщение $u = 011$, то $x_1 = 0; x_2 = 1; x_3 = 1$, и проверочные символы легко определяются:

$$x_4 = -1 - 1 = 1 + 1 = 2 = 0; x_5 = -1 = 1; x_6 = -1 = 1$$

так что кодовое слово $x = 011011$.

В коде, над полем $\mathbb{F}_2$, имеется 2^k кодовых слов, где k длина сообщения.

Свойства линейного кода

(i). $x = x_1...x_n$ является кодовым словом, если и только если $Hx^T = 0$.
(ii) Обычно проверочная матрица H – это $((n-k) \times n)$ матрица вида

$$H = [A \mid I_{n-k}]$$

(iii) Порождающая матрица. Если известно соощение $u = u_1...u_k$, то как найти соответствующщее кодовое слово $x = x_1...x_n$? Во-первых, $x_1 = u_1; ...x_k = u_k$ или

$$\begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = I_k \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix}$$

Где I_k единичная матрица.
Тогда получаем, что

$$[A \mid I_{n-k}] \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = 0;$$

$$\begin{pmatrix} x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = -A \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = -A \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix}$$

В двоичном случае $-A = A$.
Имеем

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \left[\frac{I_k}{-A} \right] \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix}$$

и транспонируя, получаем равенство

$$x = uG$$

где

$$G = \left[I_k \mid -A^T \right]$$

G называется порождающей матрицей.

Пример 2. *Выпишем порождающую матрицу*

$$G = \left[I_3 \mid -A^T\right] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Каждое из восьми кодовых слов имеет вид:

$$u_1 \cdot (\textit{строка 1}) + u_2 \cdot (\textit{строка 2}) + u_3 \cdot (\textit{строка 3})$$

Вновь убеждаемся, что кодовое слово, соответствующее сообщению $u = 011$, *равно*

$$x = uG = \textit{строка 2} + \textit{строка 3} = 010101 + 001110 = 011011$$

(iv) Параметры линейного кода. Говорят, что кодовое слово $x = x_1...x_n$ имеет дилну n. Здесь имеется в виду не длина вектора в обычном математическом смысле, а число символов в этом векторе, т.е. его размерность. Число n называют также блоковой длиной кода. Если H имеет $n-k$ линейно независимых строк, то имеется 2^k кодовых слов. Число k называется размерностью кода.

Мы назовем такой код $[n, k]$ кодом. Этот код использует n символов для передачи k символов сообщения, поэтому говорят, что код имеет скорость или эффективность $R = k/n$.

Определение 4. *Расстояние (Хемминга) между двумя векторами* $x = x_1...x_n$ *и* $y = y_1...y_n$ *равно числу позиций, в которых они различаются; оно обозначается* $dist(x, y)$.

Определение 5. *Вес (Хемминга) вектора* $x = x_1...x_n$ *равен числу ненулевых* x_i*; оно обозначается* $wt(x)$.

Третьим важным параметром кода C, кроме длины и размерности, является минимальное расстояние Хемминга между его кодовыми словами:

Лемма 2. ***(Минимальное расстояние кода)*** *Кодовое расстояние линейного кода* $C \subset \mathbb{F}_q^n$ *равно минимальному весу вектора в линейном подпространстве* C. *Другими словами,*

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

$$d(C) = min[dist(u, v)] = min[wt(u - v)]; u \in C; v \in C; u \neq v;$$

Доказательство непосредственно вытекает из равенства $d(u, v) = wt(c)$, где $c = u - v \in C$.

Таким образом, вместо изучения совокупности взаимных расстояний между парами векторов a, b (функции от двух аргументов) линейного кода достаточно рассмотреть совокупность весов ненулевых элементов линейного подпространства C (функции одного аргумента).

О системе Mathematica

Mathematica — система компьютерной алгебры, используемая во многих научных, инженерных, математических и компьютерных областях. Изначально система была придумана Стивеном Вольфрамом, в настоящее время разрабатывается компанией Wolfram Research.

Аналитические преобразования

- Решение систем полиномиальных и тригонометрических уравнений и неравенств, а также трансцендентных уравнений, сводящихся к ним.
- Решение рекуррентных уравнений.
- Упрощение выражения.
- Нахождение пределов.
- Интегрирование и дифференцирование функций.
- Нахождение конечных и бесконечных сумм и произведений.
- Решение дифференциальных уравнений и уравнений в частных производных.
- Преобразования Фурье и Лапласа, а также Z-преобразование
- Преобразование функции в ряд Тейлора, операции с рядами Тейлора: сложение, умножение, композиция, получение обратной функции и т. д.
- Вейвлет-анализ

Численные расчёты

- Вычисление значений функций, в том числе специальных, с произвольной точностью.
- Решение систем уравнений.
- Нахождение пределов.
- Интегрирование и дифференцирование.
- Нахождение сумм и произведений.
- Решение дифференциальных уравнений и уравнений в частных производных.
- Полиномиальная интерполяция функции от произвольного числа аргументов по набору известных значений.
- Преобразования Фурье и Лапласа, а также Z-преобразование.
- Расчет вероятностей

Теория чисел

- Определение простого числа по его порядковому номеру, определение количества простых чисел, не превосходящих данное.
- Дискретное преобразование Фурье.
- Разложение числа на простые множители, нахождение НОД и НОК.

Линейная алгебра

- Операции с матрицами: сложение, умножение, нахождение обратной матрицы, умножение на вектор, вычисление экспоненты, получение определителя.
- Поиск собственных значений и собственных векторов.

Разработка программного обеспечения

- Автоматическое генерирование C кода и его компоновка.
- Автоматическое преобразование компилируемых программ системы Mathematica в C код для автономного или интегрированного использования.
- Использование SymbolicC для создания, обработки и оптимизации C кода.
- Интеграция внешних динамических библиотек
- Поддержка CUDA и OpenCL.

Язык программирования Mathematica

Кроме того, Mathematica — это интерпретируемый язык функционального программирования. Можно сказать, что система Mathematica написана на языке Mathematica, хотя некоторые функции, особенно относящиеся к линейной алгебре, в целях оптимизации были написаны на языке C.

Mathematica поддерживает и процедурное программирование с применением стандартных операторов управления выполнением программы (циклы и условные переходы), и объектно-ориентированный подход. Mathematica допускает отложенные вычисления. Также в систему Mathematica можно задавать правила работы с теми или иными выражениями.

Построение линейного кода в Wolfram Mathematica 9.0

Теперь применив эту теорию в системе Mathematica необходимо построить линейный код (я постараюсь продемонстрировать эту теорию с помощью пакета Mathematica). Для начала нужно ввести необходимые данные и на их основе построить код C, найти минимальный вес, построить таблицу синдромов и т.д. Т.о. можно изучить эффективность теории в компьютерной системе. Как известно, современные компьютеры имеют два и более ядер. В системе

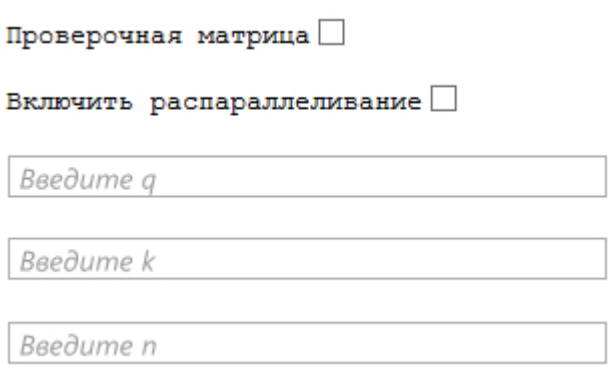
Mathematica все вычисления производятся на одном ядре, сколько бы их не было, но специальными инструментами можно подключить все ресурсы компьютера, в том числе, ресурсы видеокарты (но только для видеокарт NVidia). Сначала все вычисления будут производиться на одном ядре, а потом задействовав все ресурсы повторно произведем построение.

Итак, для построения линейного кода C нужна проверочная или порождающая матрица (система позволяет выбрать пользователю, с помощью какой матрицы будет осуществляться построение кода), а чтобы построить матрицу необходимы ее размеры и поле над которым будут производиться все вычисления и построения. Сразу хочу оговориться, элементы матрицы система будет строить без участия пользователя (над полем, который он задаст), чтобы каждый раз не приходилось заполнять элементы матриц, т.к. они будут больших размеров. Код буду приводить небольшими кусками, чтобы не загромождать текст и не нарушать восприятие.

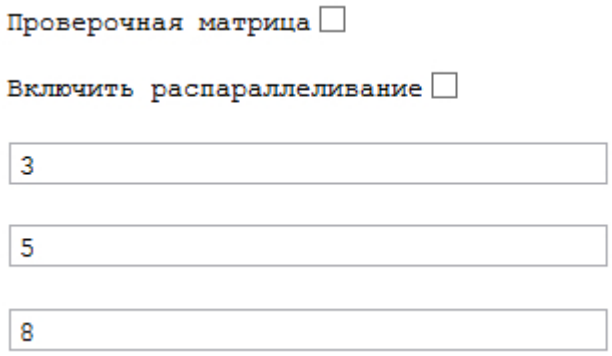
Запускаем следующий код:

```
"Проверочная матрица" Checkbox[Dynamic[ProverMatrica]]
"Включить распараллеливание" Checkbox[Dynamic[ParallelOn]]
InputField[Dynamic[q], Number, FieldHint -> "Введите q"]
InputField[Dynamic[k], Number, FieldHint -> "Введите k"]
InputField[Dynamic[n], Number, FieldHint -> "Введите n"]
```

на экране отобразиться



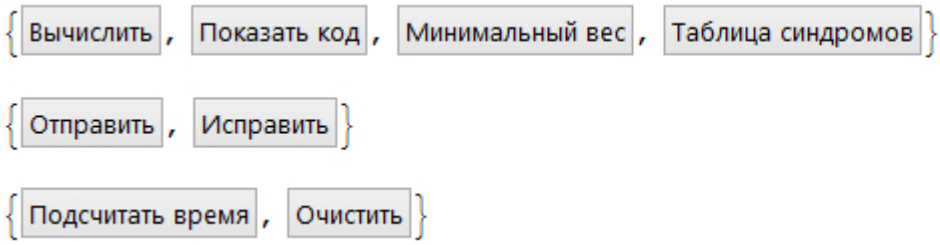
Разберем более подробно. Первая строка включает или отключает функцию распараллеливания. Во второй строке можно указать с помощью какой матрицы (порождающей или проверочной) система должна строить код. В следующем поле (где q) указывается множество в котором производится построение. В поле (где k) указывается число строк матрицы, а в поле (где n) число столбцов матрицы. Заполняем эти поля:



Итак, теперь имеются все необходимые данные для построения порождающей или проверочной матрицы. Для продолжения необходим следующий функционал.

```
Button["Вычислить",
 DoMatrix[];
 CalculateMatrices[];
 Print[Column[{"Проверочная матрица " MatrixForm[H],
    "Порождающая матрица " MatrixForm[G]}]];
 ]
```

здесь представлен код, который строит графический элемент "кнопка". Таких кнопок будет несколько и у каждой свой функционал.



Приведенный код рисует кнопку "Вычислить". При нажатии вызываются методы DoMatrix и CalculateMatrices. Первый метод строит матрицу с произвольными элементами над заданным полем. Второй находит либо порождающую, либо проверочную матрицу. Напомпню, что матрицы взаимосвязаны соотношением $GH^T = 0$ или $HG^T = 0$.

```
DoMatrix[] := Module[{i, j},
   If[Head[k] == Symbol || Head[n] == Symbol,
    Print["Введите размеры матрицы!"]];
   If[k >= n, Print["k не может быть больше n!"]]
    If[Head[k] == Integer && Head[n] == Integer && k < n,
     M = Array[Subscript[a, ##] &, {n - k, n}];
      If[ProverMatrica,
       For[i = 1, i <= n - k, i++,
        For[j = 1, j <= n, j++,
         M[[i, j]] = RandomInteger[{0, q - 1}];
          ]
        ]
       ,
       M = IdentityMatrix[{k, n}];
       For[i = 1, i <= k, i++,
        For[j = k + 1, j <= n, j++,
         M[[i, j]] = RandomInteger[{0, q - 1}];
          ]
        ]
```

```
      ]
    ]
  ]
```

В этом коде сначала строится пустая матрица размера $n-k \times n$, если выбрана проверочная матрица или $k \times n$, если выбрана порождающая матрица. Затем, в цикле заполняются элементы матрицы. После этих операций матрицы выводятся на экран. Результат работы этого кода следующий:

Проверочная матрица $\begin{pmatrix} 1 & 1 & 2 & 0 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 1 & 2 & 0 & 0 & 1 \end{pmatrix}$

Порождающая матрица $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 1 \end{pmatrix}$

Теперь известны данные на основе которых система построит код. Эту функцию выполняют следующие конструкции:

```
Button["Показать код",
 DoLCod[];
 Print[LCodTable];
 ]
```

Здесь представлен второй графический элемент кнопка с вызовом модуля, отвечающий за построение линейного кода. Останавливаться на нем не буду, т.к. от предыдущей конструкции ничем не отличается. Более детально нужно разобрать модуль, который вызывется при нажатии этой кнопки. Для начала приведу его:

```
DoLCod[] := Module[{u, Cod},
  LCodTable = {};
  (* формирование линейного кода *)
  For[i = 0, i <= q^k - 1, i++,
   u = IntegerDigits[i, q, k];
   Cod = u.G;
   LCodTable = Append[ LCodTable, Mod[Cod, q] ];
   ];
  ]
```

Как вы могли заметить, код небольшой и всё благодаря мощным инструментам, которые предоставляет система математика. Если набрать метод с аналогичным функционалом в другой компьютерной системе, напимер в C++, поверьте, код будет намного длиннее и сложнее по структуре. Вкратце опишу работу этого блока. Известно, что общее число всех сообщений не больше q^k, пользуясь этим правилом в цикле строятся все возможные сообщения над полем, который был задан и каждое полученное сообщение умножается на порождающую матрицу, т.к. каждый элемент линейного кода удовлетворяет равенству $c = uG$, где u - сообщение. Таким образом, получится линейный код C.

```
{{0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 2, 2, 1}, {0, 0, 0, 0, 2, 1, 1, 2},
 {0, 0, 0, 1, 0, 0, 2, 2}, {0, 0, 0, 1, 1, 2, 1, 0}, {0, 0, 0, 1, 2, 1, 0, 1},
 {0, 0, 0, 2, 1, 2, 0, 2}, {0, 0, 0, 2, 2, 1, 2, 0}, {0, 0, 1, 0, 0, 1, 0, 1},
 {0, 0, 1, 0, 2, 2, 1, 0}, {0, 0, 1, 1, 0, 1, 2, 0}, {0, 0, 1, 1, 1, 0, 1, 1},
 {0, 0, 1, 2, 0, 1, 1, 2}, {0, 0, 1, 2, 1, 0, 0, 0}, {0, 0, 1, 2, 2, 2, 2, 1},
 {0, 0, 2, 0, 1, 1, 2, 0}, {0, 0, 2, 0, 2, 0, 1, 1}, {0, 0, 2, 1, 0, 2, 2, 1},
 {0, 0, 2, 1, 2, 0, 0, 0}, {0, 0, 2, 2, 0, 2, 1, 0}, {0, 0, 2, 2, 1, 1, 0, 1},
 {0, 1, 0, 0, 0, 2, 0, 1}, {0, 1, 0, 0, 1, 1, 2, 2}, {0, 1, 0, 0, 2, 0, 1, 0},
 {0, 1, 0, 1, 1, 1, 1, 1}, {0, 1, 0, 1, 2, 0, 0, 2}, {0, 1, 0, 2, 0, 2, 1, 2},
 {0, 1, 0, 2, 2, 0, 2, 1}, {0, 1, 1, 0, 0, 0, 0, 2}, {0, 1, 1, 0, 1, 2, 2, 0},
 {0, 1, 1, 1, 0, 0, 2, 1}, {0, 1, 1, 1, 1, 2, 1, 2}, {0, 1, 1, 1, 2, 1, 0, 0},
 {0, 1, 1, 2, 1, 2, 0, 1}, {0, 1, 1, 2, 2, 1, 2, 2}, {0, 1, 2, 0, 0, 1, 0, 0},
 {0, 1, 2, 0, 2, 2, 1, 2}, {0, 1, 2, 1, 0, 1, 2, 2}, {0, 1, 2, 1, 1, 0, 1, 0},
```

Код очень большой, больше 200 элементов, поэтому представил только часть. Теперь имеются матрицы и линейный код, т.е. все необходимые данные для нахождения минимального веса, также на основе этих легко построить таблицу синдромов с лидерами смежных классов. Напомню, с помощью минимального веса можно узнать, какое количество ошибок может найти и исправить код. Забегая вперед, таблица синдромов предназначена для определения вектора ошибки.

Найдем минимальный вес:

```
Button["Минимальный вес",
 DoMDist[];
 Print["Минимальный вес кода равен ", LCodMin];
 Print["Вектор: ",
  LCodTable[[ Position[MnojestvoMinim, LCodMin][[1, 1]] + 1]]];
 ]
```

Кнопка, для вызова метода нахождения и сам метод:

```
DoMDist[] := Module[{Minim},
  MnojestvoMinim = {};
  LCodMin = {};
  For[i = 1, i <= q^k - 1, i++,
   Minim = Count[LCodTable[[i]], Except[0]];
   If[Minim != 0,
    MnojestvoMinim = Append[MnojestvoMinim, Minim];
    ];
   ];
  LCodMin = Min[MnojestvoMinim];
  ]
```

Основополагающий принцип нахождения - это ненулевой вектор и число ненулевых позиций этого вектора. Обходим все множество C и для каждого элемента находим количество ненулевых позиций. Если полученное число ненулевое добавляем в локальное множество и после завершения обхода находим минимальный элемент локального множества.

```
Минимальный вес кода равен 3
Вектор: {0, 0, 0, 1, 0, 0, 2, 2}
```

Далее будем строить таблицу синдромов, изучив следующую главу.

Таблица синдромов

Для построения таблицы синдромов понадобится понятие смежного класса.

Определение 6 (Смежный класс)**.** *Пусть C является $[n, k]$ линейным кодом над полем из q элементов. Для любого вектора a множество*

$$a + C = \{a + x : x \in C\}$$

называется смежным классом кода C.

Любой вектор b находится в некотором смежном классе. Два вектора a и b лежат в одном и том же смежном классе, тогда и только тогда, когда $(a - b) \in C$.

Предложение 1. *Два смежных класса либо не пересекаются, либо совпадают.*

Доказательство. Если смежные классы $a + C$ и $b + C$ пересекаются, то возьмем $v \in (a+C) \cap (b+C)$. Тогда $v = a+x = b+y$, где $x, y \in C$. Следовательно, $b = a+x-y = a+x\prime$ $(x\prime \in C)$ и поэтому $b + C \subset a + C$. Аналогичным образом получается, что $a + C \subset b + C$ и значит, что $a + \ = b + C$.

Синдром. Имеется простой способ, как определить, в каком смежном классе находится y: надо вычислить Hy^T, который называется синдромом y.

Два ветора находятся в одном и том же смежном классе кода C, если и только если они имеют один и тот же синдром.
Имеется взаимно однозначное соответствие между синдромами и смежными классами.

Схема обнаружения ошибок, как раз состоит в сравнении синдрома с нулем. Для этого мы еще раз вычисляем проверочные символы, используя принятые информационные символы, и смотрим совпадают ли они с принятыми проверочными символами.

Построение таблицы синдромов и лидеров смежных классов

Теперь необходимо построить таблицу синдромов и лидеров смежных классов. Модули, выполняющие эту задачу, выглядят следующим образом:

```
Графический модуль
Button["Таблица синдромов",
 DoSkTable[];
 Print["Таблица синдромов и соответствующих лидеров смежных классов \
```

```
", Grid[{{Grid[Sindrom], Grid[LiderSK]}}, Frame -> All], ";"];
 ]

 Модули для нахождения лидеров и смежных классов
 getWeight[vector_] := Module[{i, res},
   res = 0;
   For[i = 1, i <= Length[vector], i++,
    If[ vector[[i]] != 0, res++ ];
    ];
   res
    ];

DoSkTable[] := Module[{AllLiders, SortLiders, i, j, l, Q},
  LiderSK = {};
  Sindrom = {};

  AllLiders = {};
  For[i = 0, i <= q^n - 1, i++,
   AllLiders = Append[AllLiders, IntegerDigits[i, q, n]];
   ];

  SortLiders = {};
  For[i = 0, i <= n, i++,
   Q = Select[AllLiders, getWeight[#] == i &];
   For[j = 1, j <= Length[Q], j++,
    SortLiders = Append[SortLiders, Q[[j]]];
    ];
   ];

  l = 1;
  While[Length[LiderSK] != q^n/q^k,
   If[Position[Sindrom, Mod[H.SortLiders[[l]], q]] == {},
    Sindrom = Mod[Append[Sindrom, H.SortLiders[[l]]], q];
    LiderSK = Append[LiderSK, SortLiders[[l]]]
    ];
   l++;
   ];

  ]
```

Работа начинается с модуля, который называется DoSkTable, модуль getWeight - вспомогательный. Сначала строятся все возможные лидеры смежных классов над указанным полем и каждый элемент заполняет локальное множество AllLiders. Далее, это множество сортируется в порядке возрастания веса элементов. На выходе получается множество состоящее из векторов вида $\{\{0, ..., 0, a\}, \{0, ..., a, 0\}, ..., \{b, ..., b\}\}$, где a – минимальный элемент поля, а b – максимальный. Отсортированное множество хранится в переменной SortLiders. Потом каждый элемент найденного множества умножается на проверочную матрицу и получится синдром. Напомню, количество синдромов с их лидерами равно q^n/q^k. Каждый полученный синдром помещается в таблицу с его лидером смежного класса. Таким образом, формируется таблица синдромов.

Таблица синдромов и соответствующих лидеров смежных классов

0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1
0	0	2	0	0	0	0	0	0	0	2
0	1	0	0	0	0	0	0	0	1	0
0	2	0	0	0	0	0	0	0	2	0
1	0	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	0	2	0	0
1	1	2	0	0	0	0	1	0	0	0
2	2	1	0	0	0	0	2	0	0	0
0	1	1	0	0	0	1	0	0	0	0
0	2	2	0	0	0	2	0	0	0	0
2	0	2	0	0	1	0	0	0	0	0
1	0	1	0	0	2	0	0	0	0	0
1	0	2	0	1	0	0	0	0	0	0
2	0	1	0	2	0	0	0	0	0	0
1	2	0	1	0	0	0	0	0	0	0
2	1	0	2	0	0	0	0	0	0	0
0	1	2	0	0	0	0	0	0	1	2
0	2	1	0	0	0	0	0	0	2	1
1	1	0	0	0	0	0	0	1	1	0
2	2	0	0	0	0	0	0	2	2	0
1	1	1	0	0	0	0	1	0	0	2
1	2	2	0	0	0	0	1	0	1	0
2	1	2	0	0	0	0	1	1	0	0
2	2	2	0	0	0	0	2	0	0	1
2	1	1	0	0	0	0	2	0	2	0
1	2	1	0	0	0	0	2	2	0	0

Для чего все эти манипуляции с таблицей синдромов? Ответ на этот вопрос описан в следующей главе.

Декодирование линейного кода

Предположим, что сообщение $u = u_1...u_k$ закодировано в кодовое слово $x = x_1...x_n$, которое потом передается по каналу. Так как канал с шумом, принятый вектор $y = y_1, ...y_n$ может отличаться от x. Удобно ввести вектор ошибок

$$e = y - x = e_1...e_n$$

Декодер должен на основании y решить, какое сообщение u или какое кодовое слово x было передано.

Теорема 1. *Предположим, что любые $d-1$ столбцов проверочной матрицы B линейно независимы над полем $\mathbb{F}_q$.*

Тогда кодовое расстояние $d(C)$ кода C, определяемого прочерочной матрицей B, не меньше, чем d.

Если в дполонение выше высказанного условия существует линейно-зависимый комплект из d столбцов проверочной матрицы B, то $d(C) = d$.

Наоборот. Если код имеет кодовое растояние не меньше, чем d, то любые $d-1$ столбцов его прочерочной матрицы B являются линейно-независимыми.

Доказательство. В виду последней леммы достаточно показать, что вес любого ненулевого вектора x кода C не меньше d.

Предположим обратное, т.е. предположим, что существует кодовый вектор x, вес которого меньше d. Так как $xB^T = 0$, то комплект столбцов матрицы , номера которых совпадают с номерами ненулевых координат вектора x, является линейно-зависимым. Это противоречит условию теоремы. Поэтому $d(C) \leq d$. Последнее и предпоследнее утверждения теоремы очевидны.

Теорема 2. *Код с минимальным расстоянием d может исрпавлять $(d-1)/2$ ошибок. Если d четное, то код может одновременно исправлять $(d-2)/2$ ошибок и обнаруживать $d/2$ ошибок.*

Доказательство. Предположим, что $d = 3$. Шар радиуса r с центром в точке u состоит из всех векторов v таких, что $dist(u, v) \leq r$. Если вокруг каждого кодового слова описать шар радиуса 1, то эти шары не пересекаются. Если передавалось кодовое слово u и произошла одна ошоибка, то принятый вектор a находится внутри шара, проведенного вокруг u, и поэтому a ближе к u, чем к любому другому кодовому слову v. Таким обрзом, декодирование в ближайшее кодовое слово исправит эту ошибку. Подобным образом, если $d = 2t + 1$, то шары радиуса t, описанные около каждового кодового слова, не пересекаются и код может исправлять t ошибок.

Теперь предположим, что d четное. Сферы радиуса $(d-2)/2$, проведенные вокруг кодовых слов, не пересекаются, и поэтому код может исправлять $(d-2)/2$ ошибок. Но если произошло $d/2$ ошибок, то принятый вектор a может находиться посреди между двумя кодовыми словами. В этом случае декодер может только обнаруживать, что произошло $d/2$ ошибок.

Если произошло больше чем $d/2$ ошибок, то принятный вектор будет ближе к некторому другому кодовому слову, а не к верному. Тогда декодер ошибется и выдаст неправильное кодовое слово. Это называется ошибкой декодирования.

Имитация канала с шумом и декодирование

Итак, декодирвоние! Весь смысл всех этих операций является в декодировании. Ведь нужно передать сообщение, получить её, проверить на ошибку и исправить ее, если это возможно.

Этот функционал также был реализован в математике. Чтобы продемонстрировать декодирование (исправление ошибки при ее возникновении) необходимо будет симитировать канал передачи. Передать вектор, проверить на наличие ошибки и если возможно исправить ее - исправить.

Имитация канала:

```
Button["Отправить",
 Clear[Otpravlen, GetVector];
 Otpravlen = RandomChoice[LCodTable];
 Print["Отправлен: ", Otpravlen];
 SendVector[Position[LCodTable, Otpravlen][[1, 1]]];
 Print["Получен: ", GetVector];
 ],
Button["Исправить",
 RepairVector[];
 Print["Был отправлен: ", GetVector];
 ]
```

Имеется два графических элемента, которые называются Отправить и Исправить. При активации первой кнопки программа случайно выбирает из линейного множества вектор и передает модулю, который представлен ниже:

```
SendVector[Poziciya_] := Module[{i, ErrorVector},
 ErrorVector = RandomChoice[{0}, n];
 l = LCodMin;
 If[EvenQ[LCodMin], l = LCodMin/2, If[l != 1, l = (LCodMin - 1)/2]];
 For[i = 1, i <= l, i++,
  ErrorVector[[RandomInteger[{1, Length[ErrorVector]}]]] =
   RandomInteger[{0, q - 1}]
  ];
 GetVector = Mod[LCodTable[[Poziciya]] + ErrorVector, q];
 ]
```

Этот модуль имитирует канал с шумом, т.е. случайным образом генерирует вектор длины n и прибавляет к вектору, который был передан (вектор ошибки может быть нулевым). На выходе получается некоторый вектор и нам пока неизвестно - с ошибкой он или нет.

```
Отправлен: {0, 2, 1, 0, 2, 0, 1, 2}

Получен: {0, 1, 1, 0, 2, 0, 1, 2}
```

Возвращаемся к коду, где описаны графические элементы, а именно к кнопке Исправить. При ее активации вызывается модуль в котором описаны методы исправляющие ошибку.

```
RepairVector[] := Module[{},
 GetVector =
  Mod[GetVector -
```

```
    LiderSK[[Position[Sindrom, Mod[H.GetVector, q]][[1, 1]]]], q];
]
```

Весь метод в одну строку... Для исправления необходима таблица синдромов и проверочная матрица. Если умножить полученный вектор на проверочную матрицу, то получится синдром. По его позиции можно определить лидер смежного класса, потом сложить его с полученным вектором. Т.о. определили возникшую ошибку и исправили ее.

```
Был отправлен: {0, 2, 1, 0, 2, 0, 1, 2}
```

Построение кода в различных режимах

Теперь перейдем к изучению эффективности работы методов теории кодирования в системе Wolfram Mathematica 9.0. Рассмотрим время построения кода при различных режимах функционирования системы. К сожалению, в моем компьютере установлена видеокарта от AMD, поэтому ресурсы видеокарточки не получилось задействовать. При желании вы можете проделать это самостоятельно, подключив пакет CUDALink. Несмотря на это, распараллеливая процессы, тоже можно достичь определенных результатов.

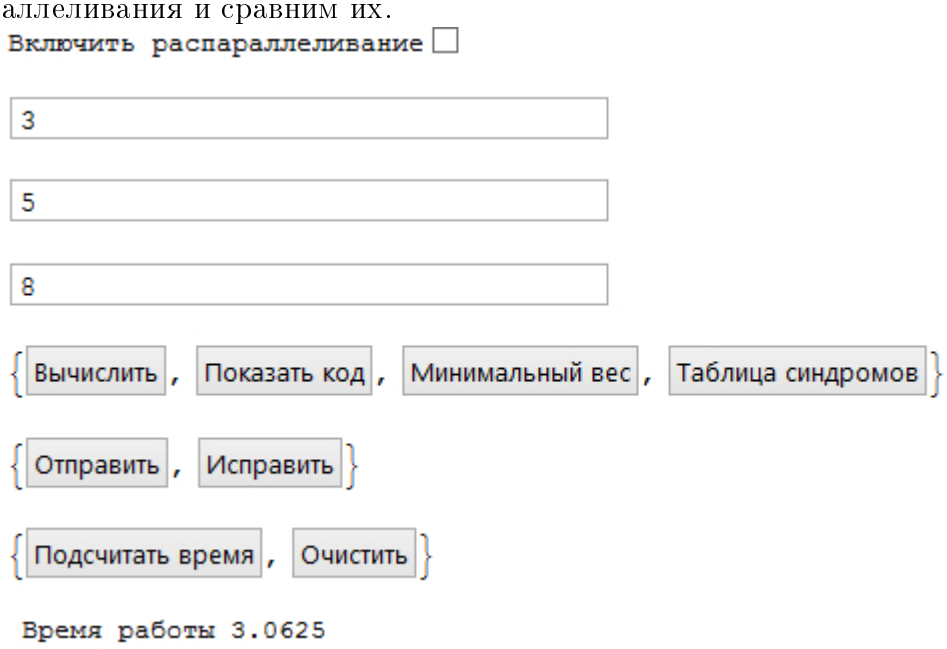
Функций распараллеливающих процессы в системе Wolfram Mathematica довольно много, я воспользовался функцией, которая называется ParallelDo[]. Почему ее? У этой функции простая конструкция, она полностью удовлетворяет моим запросам и эта функция была первая с которой я познакомился. Для подсчета времени воспользовался функцией Timing[].

Код выглядит следующим образом:

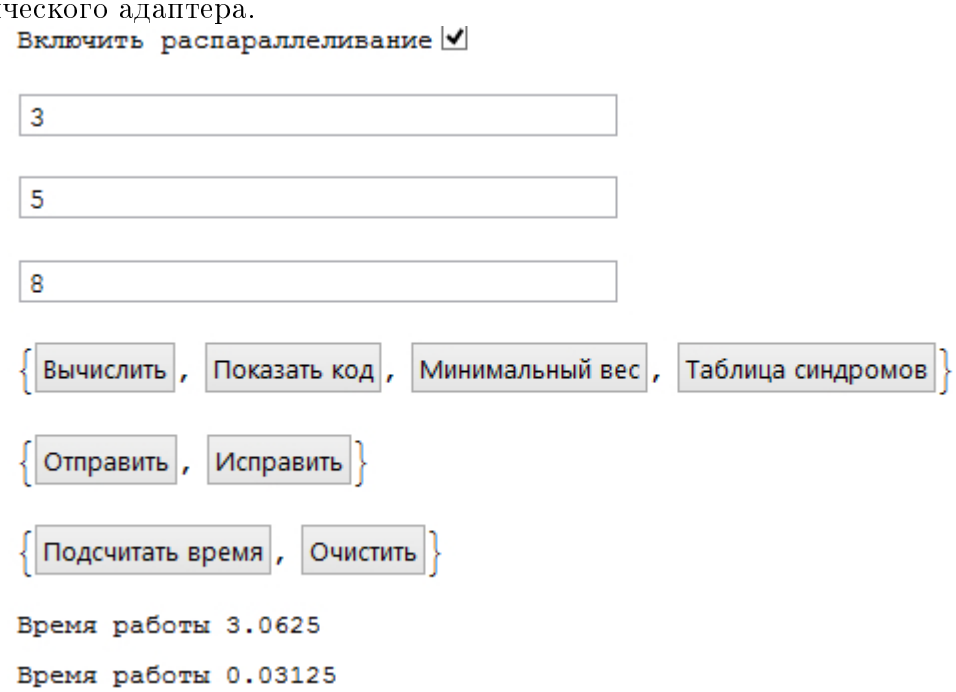
```
Button["Подсчитать время",
 Vremya = 0;
 If[ParallelOn,
  Vremya = Timing[ParallelDo[DoMatrix[], {1}]][[1]];
  Vremya += Timing[ParallelDo[CalculateMatrices[], {1}]][[1]];
  Vremya += Timing[ParallelDo[DoLCod[], {1}]][[1]];
  Vremya += Timing[ParallelDo[DoMDist[], {1}]][[1]];
  Vremya += Timing[ParallelDo[DoSkTable[], {1}]][[1]];
  Print["Время работы ", Vremya];
  ,
  Vremya = Timing[DoMatrix[]][[1]];
  Vremya += Timing[CalculateMatrices[]][[1]];
  Vremya += Timing[DoLCod[]][[1]];
  Vremya += Timing[DoMDist[]][[1]];
  Vremya += Timing[DoSkTable[]][[1]];
  Print["Время работы ", Vremya];
  ]
 ]
```

Разберем этот модуль. Первым делом осуществляется проверка на наличие флага для распараллеливания. Если флаг установлен срабатывают методы распараллеливания, если нет, то функции работают в обычном режиме. Далее, по очереди вызываются все модули, которые были описаны выше. Во время их работы подсчитывается время, затраченное на их исполнение. Пока работают модули время копится в переменной. После выполнения всех операций на выходе получается общее время.

Сначала запустим в обычном режиме и посмотрим на результаты, потом в режиме распараллеливания и сравним их.



Проверки проходили на компьютере с двухядерным процессором, без подключения графического адаптера.



Итак, результаты, как вы видите, впечатляющие. Функции отрабатывают очень быстро. Казалось бы, должно быть в два раза, но это не так. Так что даже на не самых мощных компьютерах, существенно можно повысить производительность, не говоря уже о серверных

мощностях. Однако не все так гладко, имеются свои подводные камни. Распараллеливание эффективно только для больших кодов. С небольшими кодами результат будет обратный и обусловлено это следующим: прежде чем запустить процесс распараллеливания, система распределяет нагрузку по ядрам и на это тоже нужно определенное время. Т.е. в случаях работы с маленьким кодом лучше не включать эту функцию.

На этом с линейными кодами закончим, изучим циклические коды и проделаем с ними ту же работу.

Циклические коды

Определение 7. *Код $\mathbb{C}$ называется циклическим, если любой циклический сдвиг кодового слова из кода C также является кодовым словом из кода C, т.е. если $(c_0, c_1, ..., c_{n-1}) \in \mathbb{C}$, то и $(c_{n-1}, c_0, ..., c_{n-2}) \in \mathbb{C}$*

Будем использовать следующее обозначение. Если $\mathbb{F}$ — поле, то $\mathbb{F}[x]$ — множество многочленов от x с коэффициентами из $\mathbb{F}$. Множество $\mathbb{F}[x]$ является кольцом многочленов.

Для алгебраического описания циклических кодов рассмотрим кольцо $R_n = \mathbb{F}[x]/(x^n - 1)$, состоящее из класса вычетов кольца $\mathbb{F}[x]$ по модулю многочлена $x^n - 1$. Все многочлены степени не больше $n-1$ попадают в различные классы вычетов. Мы выберем эти многочлены в качестве представителей этих классов.

Теорема 3. *В кольце R_n подпространство является циклическим подпростанством тогда и только тогда, когда оно является идеалом.*

Доказательство. Умножение на x соответствует циклическому сдвигу. Умножая в R_n многочлен $c(x)$ на x, получаем:

$$xc(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1} = c_{n-1} + c_0x + ... + c_{n-2}x^{n-1}$$

так как в R_n имеет место равенство $x^n = 1$. Этому многочлену будет соответствовать вектор $(c_{n-1}, c_0, ..., c_{n-2})$. По определению идеала, если I идеал, то произведение любого элемента из I на любой элемент из R_n будет давать элемент из I, и поскольку $xc(x)$ — циклический сдвиг вектора, то I — циклическое подпространство.

Пусть $g(x)$ — нормированный многочлен наименьшей степени, такой, что класс вычетов $\{g(x)\}$ принадлежит идеалу J. Если $f(x)$ — многочлен степени, меньше чем n, который делится на $g(x)$, то класс вычетов $\{f(x)\}$ принадлежит J, и, наобарот, если $\{f(x)\}$ принадлежит идеалу J, то многочлен $f(x)$ делится на многочлен $g(x)$. Кроме того, многочлен $x^n - 1$ делится на $g(x)$, и любой нормированный многочлен, на который делится $x^n - 1$, порождает свой идеал в R_n. Многочлен $g(x)$ называется многочленом, порождающим идеал (или образующим многчленом).

Теорема 4. *Пусть $f(x) = g(x)h(x)$, где $f(x)$ — многочлен степени n, а $h(x)$ — многочлен степени k. Тогда идеал, порожденный классом вычетов $\{g(x)\}$ в алгебре многочленов по модулю $f(x)$, имеет размерность k.*

Доказательство. В идеале, который является некоторым подпространством, векторы $\{g(x)\}, \{xg(x)\}, ..., \{x^{k-1}g(x)\}$ линейно независимы, ибо любая линейная комбинация этих векторов имеет вид $\{(a_0 + ... + a_{k-1}x^{k-1})g(x)\}$ и отлична от нуля, так как каждый класс вычетов содержит некоторый многочлен степени, меньшей n. Более того, если $\{s(x)\}$ принадлежит идеалу, многочлен $s(x)$ делится на $g(x)$, а если $s(x)$ — многочлен наименьшей степени в своем классе вычетов, то его степень меньше n. Таким образом,

$$s(x) = g(x)q(x) = g(x)(q_0 + q_1x + ... + q_{k-1}x^{k-1})$$

и

$$\{s(x)\} = q_0\{g(x)\} + q_1\{xg(x)\} + ... + q_{k-1}\{x^{k-1}g(x)\},$$

так что k векторов $\{g(x)\}, \{xg(x)\}, ..., \{x^{k-1}g(x)\}$ порождают идеал. Следовательно, размерность идеала равна k.

Итак, циклический код полностью задается многочленом $g(x)$, на который делится многочлен $x^n - 1$. С другой стороны, этот же код может быть полностью определен условием, что он является нулевым пространством идеала, порожденного многочленом $h(x) = (x^n - 1)/g(x)$. Если $g(x)$ — многочлен степени r, то по предыдущей теореме размерность кода равна $k = n - r$. Элемент $\{f(x)\}$ принадлежит коду тогда и только тогда, когда многочлен $f(x)$ делится на $g(x)$.

Многочлен $h(x)$ называется проверочным многочленом для кода C, порожденного многочленом $g(x)$. Поскольку $x^n - 1$ делится на $h(x)$, то многчлен $h(x)$ может быть использован в качестве многчлена, порождающего циклический код. Этот последний код эквивалентен коду, двойственному к коду C, и в теории циклических кодов его обычно называют просто кодом, двойственным к коду C.

Пример. Пусть задан многочлен $x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1)$ над полем Галуа $GF(2)$. Многочлен $g(x) = x^3 + x^2 + 1$ порождает циклический $(7, 4)$—код. Элементы

$$\{x^3g(x)\} = (1101000), \{x^2g(x)\} = (0110100), \{xg(x)\} = (0011010), \{g(x)\} = (0001101)$$

$$G = \begin{bmatrix} 1101000 \\ 0110100 \\ 0011010 \\ 0001101 \end{bmatrix}$$

можно выбрать в качестве базисных векторов, и, следовательно, матрицу G можно выбрать в качестве порождающей матрицы для этого кода. Этот код является нелевым пространством идеала, порожденного многочленом $h(x) = (x - 1)(x^3 + x + 1) = x^4 + x^3 + 1$

$$\{x^2h(x)\} = (1110100), \{xh(x)\} = (0111010), \{xh(x)\} = (0011101).$$

Поскольку, условия равенства нулю произведения многочленов и скалярного произведения соответствующих векторов не совпадают, то рассматриваемый код является нулевым

пространством матрицы H, образованный векторами $\{x^2h(x)\}, \{xh(x)\}, \{xh(x)\}$, компоненты которых записаны в обратном порядке:

$$P = \begin{bmatrix} 0010111 \\ 0101110 \\ 1011100 \end{bmatrix}.$$

Легко проверить, что $GH^T = 0$.

Матричное описание циклических кодов

Наиболее элементарный способ представления циклических кодов с помощью матриц был проиллюстрирован в предыдущем разделе. Если многочлен $g(x) = a_r x^r + a_{r-1}x^{r-1} + ... + a_0$ порождает код, то все векторы $\{x^{n-r-1}g(x)\}, \{x^{n-r-2}g(x)\}, ..., \{g(x)\}$ являются кодовыми векторами. Таким образом, кодовыми векторами являются все строки следующей матрицы:

$$G = \begin{bmatrix} a_r & a_{r-1} & \dots & a_0 & 0 & . & \dots & 0 \\ 0 & a_r & a_{r-1} & \dots & a_0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & a_r & a_{r-1} & \dots & a_0 \end{bmatrix}$$

Очевидно, что строки этой матрицы линейно независимы, а ее ранг, совпадающий с размерностью кода, равен $n - r$.

Условимся в следующем. В любом циклическом коде первые k символов, т.е. коэффициенты при $x^{n-1}, x^{n-2}, ..., x^{n-k}$, будут всегда выбираться в качестве информационных символов, а последние $n - k$ символов, т.е. коэффициенты при $x^{n-k-1}, x^{n-k-2}, ..., 1$, — в качестве проверочных символов.

Порождающая матрица любого циклического кода может быть следующим образом приведена к модифицированной приведенно-ступенчатой форме. Пусть $r_i(x)$ — остаток от деления x^i на многчлен $g(x)$:

$$x^i = g(x)q_i(x) + r_i(x).$$

Тогда многочлены

$$x^i - r_i(x) = g(x)q_i(x)$$

являются кодовыми векторами. Если эти многочлены при $i = n - 1, -2, ..., n - k$ выбрать в качестве строк порождающей матрицы, то

$$G = \Big[I_k, -R\Big],$$

где I_k — единичная матрица размерности $k \times k$, а $-R$ — матрица размерности $k \times (n-k)$, j-й строкой которой является ветор из коэффициентов многочлена $-r_{n-j}(x)$. Тогда код является также нулевым пространством матрицы

$$H = \Big[R^T, I_{n-k}\Big],$$

причем j-я строка матрицы H^T является вектором коэффициентов многочлена $r_{n-j}(x)$ даже при $j \leq n-k$.

Построение циклического кода в системе Wolfram Mathematica

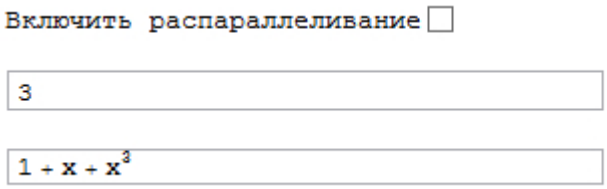
Как было описано, с циклическими кодами проще всего работать представив их в виде многочлена. По своей структуре циклический код проще, нежели линейный. Для построения достаточно знать поле и порождающий многочлен.

Приведу код для реализации:

```
"Включить распараллеливание" Checkbox[Dynamic[ParallelOn]]
InputField[Dynamic[q], Number, FieldHint -> "Введите q"]
InputField[Dynamic[g]]
```

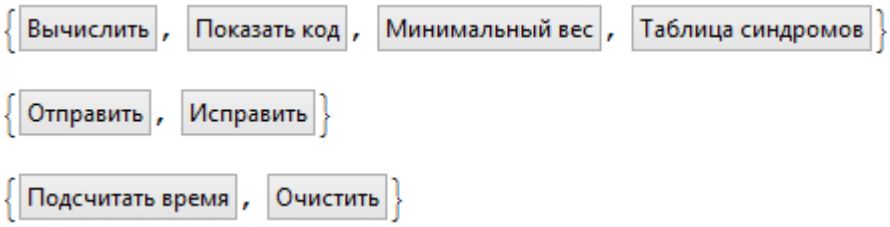
После выполнения этого кода отобразится "чекбокс"включающий или отключающий распараллеливание, поле для ввода "Поля"и поле для ввода порождающего многочлена.

Заполняем их. Этих данных достаточно для построения циклического кода.



Следующим шагом надо построить порождающую и проверочную матрицу. Код выполняющий эти функции выглядит следующим образом:

```
Button["Вычислить",
 DoMatrix[];
 Print["n = ", n];
 Print["Порождающая матрица имеет вид ",  MatrixForm[G]];
 H = Reverse[Mod[NullSpace[G], q]];
 k = Length[G];
 Print["Проверочная матрица имеет вид ", MatrixForm[H]];
 ]
```



При нажатии кнопки вычислить вызывается модуль, который строит порождающую матрицу:

```
DoMatrix[] := Module[{i, res},
  For[n = 1,
   Not[Length[
```

```
    CoefficientList[
     PolynomialRemainder[x^n - 1, g, x, Modulus -> q], x]] == 0], n++];
G = {};
res = Length[CoefficientList[g, x]];
For[i = 1, i <= n - Exponent[g, x], i++,
 v = RandomChoice[{0}, n];
 v[[i ;; res + i - 1]] = CoefficientList[g, x];
 G = Append[G, v];
 ]
]
```

Перво-наперво система находит длину кодовых слов циклического кода. Программа берет многочлен $x^n - 1$, где $n = 1$, раскладывает этот многочлен на множители таким образом, чтобы порождающий многочлен был одним из этих множителей, но при этом остаток должен быть равен нулю. Если же полином не раскладывается увеличивает n на единицу и так до тех пор, пока не будет многочлен $x^n - 1$ не будет разложен на множители без остатка. Далее создается пустое множество G, порождающий многочлен представляется в виде коэффициентов (получится вектор) и находится число элементов в векторе res. Эта информация вспомогательная, она необходима для построения строк порождающей матрицы.

Система строит нулевой вектор длины n и первые res элементов заменяет порождающим многочленом (в векторной форме). Полученный вектор будет первой строкой порождающей матрицы. Далее формирует вторую строку сдвинув порождающий многочлен на единицу вправо. Первый элемент вектора обнуляется. И так $n - r - 1$ раз, где $r = deg(g)$. В итоге, получаем порождающую матрицу размера $k \times n$, где $k = n - r - 1$ и проверочную матрицу.

Порождающая матрица имеет вид $\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$

Проверочная матрица имеет вид $\begin{pmatrix} 2 & 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 2 & 2 & 1 & 2 & 0 & 0 & 1 & 0 \\ 0 & 2 & 2 & 1 & 2 & 0 & 0 & 1 \end{pmatrix}$

Методы построения кода, таблицы синдромов и т.д. идентичны методам линейного кода, поэтому подробно описывать их не буду, но результаты приведу.

```
{{0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 1, 0, 1}, {0, 0, 0, 0, 2, 2, 0, 2},
 {0, 0, 0, 1, 2, 1, 1, 1}, {0, 0, 0, 1, 0, 2, 1, 2}, {0, 0, 0, 2, 2, 0, 2, 0},
 {0, 0, 0, 2, 1, 2, 2, 2}, {0, 0, 1, 1, 0, 1, 0, 0}, {0, 0, 1, 1, 1, 2, 0, 1},
 {0, 0, 1, 2, 2, 2, 1, 1}, {0, 0, 1, 2, 0, 0, 1, 2}, {0, 0, 1, 0, 2, 1, 2, 0},
 {0, 0, 2, 2, 0, 2, 0, 0}, {0, 0, 2, 2, 1, 0, 0, 1}, {0, 0, 2, 2, 2, 1, 0, 2},
 {0, 0, 2, 0, 0, 1, 1, 2}, {0, 0, 2, 1, 2, 2, 2, 0}, {0, 0, 2, 1, 0, 0, 2, 1},
 {0, 1, 1, 0, 2, 1, 0, 1}, {0, 1, 1, 0, 0, 2, 0, 2}, {0, 1, 1, 1, 2, 0, 1, 0},
 {0, 1, 1, 2, 0, 0, 2, 0}, {0, 1, 1, 2, 1, 1, 2, 1}, {0, 1, 1, 2, 2, 2, 2, 2},
 {0, 1, 2, 1, 0, 0, 0, 2}, {0, 1, 2, 2, 2, 1, 1, 0}, {0, 1, 2, 2, 0, 2, 1, 1},
 {0, 1, 2, 0, 1, 2, 2, 1}, {0, 1, 2, 0, 2, 0, 2, 2}, {0, 1, 0, 2, 1, 2, 0, 0},
 {0, 1, 0, 0, 2, 2, 1, 0}, {0, 1, 0, 0, 0, 0, 1, 1}, {0, 1, 0, 0, 1, 1, 1, 2},
 {0, 1, 0, 1, 2, 1, 2, 2}, {0, 2, 2, 0, 2, 0, 0, 0}, {0, 2, 2, 0, 0, 1, 0, 1},
 {0, 2, 2, 1, 1, 1, 1, 1}, {0, 2, 2, 1, 2, 2, 1, 2}, {0, 2, 2, 2, 1, 0, 2, 0},
 {0, 2, 0, 1, 2, 1, 0, 0}, {0, 2, 0, 1, 0, 2, 0, 1}, {0, 2, 0, 1, 1, 0, 0, 2},
Минимальный вес кода равен 3
```

Таблица синдромов и соответствующих лидеров смежных классов

0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1
0	0	2	0	0	0	0	0	0	2
0	1	0	0	0	0	0	0	1	0
0	2	0	0	0	0	0	0	2	0
1	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	2	0	0
2	0	2	0	0	0	0	1	0	0
1	0	1	0	0	0	0	2	0	0
1	2	1	0	0	0	1	0	0	0
2	1	2	0	0	0	2	0	0	0
1	1	2	0	0	1	0	0	0	0
2	2	1	0	0	2	0	0	0	0
0	2	2	0	1	0	0	0	0	0
0	1	1	0	2	0	0	0	0	0
2	2	0	1	0	0	0	0	0	0
1	1	0	2	0	0	0	0	0	0
0	1	2	0	0	0	0	0	1	2
0	2	1	0	0	0	0	0	2	1
1	0	2	0	0	0	0	1	0	2
1	2	0	0	0	0	0	1	2	0
2	0	1	0	0	0	0	2	0	1
2	1	0	0	0	0	0	2	1	0
2	2	2	0	0	0	1	0	2	0
1	1	1	0	0	0	2	0	1	0
1	2	2	0	0	0	1	0	0	1
2	1	1	0	0	0	2	0	0	2

;

```
Отправлен: {2, 0, 2, 2, 2, 0, 2, 2}

Получен: {2, 0, 0, 2, 2, 0, 2, 2}

Был отправлен: {2, 0, 2, 2, 2, 0, 2, 2}
```

Ошибки можно находить и исправлять так же как и в случае с линейными кодами. Но есть и другой способ сделать это, который описан в следующей главе.

Альтернативное обнаружение и исправление ошибок

Описание метода обнаружения и исправления ошибок циклических кодов.

Для декодирования циклического кода, необходимо принятую комбинацию $c(x)$ разделить

на образующий многочлен $g(x)$. Если остаток $R(x) \neq 0$, то при передаче возникла ошибка. В этом случае определяем вес w остатка. Если $w \leq t$, где t — число исправляемых ошибок, то комбинацию необходимо сложить с остатком. Полученная комбинация будет верным многочленом.

Если $w > t$, то циклически сдвигаем $c(x)$ влево и снова полученную комбинацию делим на $g(x)$. Сравниваем w и t. Если $w > t$ повторяем пока w станет не больше t. Через n сдвигов $w \leq t$ и когда это выполнится, необходимо сдвинуть комбинацию сложенную с остатком вправо на n позиций.

Пример. $c(x) = 1101110, g(x) = 1011, t = 1$.

$$\frac{c(x)}{g(x)} = r(x) = 111 => w = 3$$

Вес w остатка больше чем число исправляемых ошибок, поэтому сдвигаем $c(x)$ влево на 1 позицию.

$$w > t => \frac{1011101}{g(x)} = r(x) = 101 => w = 2$$

Еще раз:

$$\frac{0111011}{g(x)} = r(x) = 001 => w = 1$$

Вес равен числу исправляемых ошибок, поэтому $0111011 + 001 = 0111010 = u(x)$ свдигаем $u(x)$ вправо на 2 позиции и получаем верную комбинацию: 1001110

Обнаружение и исправление ошибки в Wolfram Mathematica

В случае с циклическими кодами имеется два способа исправления ошибки. Первый такой же как и с линейным кодом через таблицу синдромов. Рассматривать этот случай не будем. Более подробно разберем способ, который разбирали выше.

Код альтернативного способа декодирования.

```
InputField[Dynamic[q], Number, FieldHint → "Введите q (Zq)"]
InputField[Dynamic[t], Number, FieldHint → "Введите t (количество ошибок)"]
InputField[Dynamic[g]]
InputField[Dynamic[b]]
{
 Button["Декодировать",
  FindCod[];
]
}
```

2

1

$1 + x^3 + x^4$

$x^7 (x + x^2 + x^3 + x^6 + x^8 + x^{11} + x^{14})$

Прорисовывается форма для ввода данных. В первое поле вводим поле в котором система будет работать, для демонстрации ввел Z_2. Во второе поле – количество ошибок, которые нужно исправить, в третье образующий многочлен и в четвертое принятое "сообщение".

При активации кнопки декодировать срабатывает следующий модуль:

```
FindCod[] := Module[{},
   r = Exponent[g, x];
   For[n = 1,
    Not[Length[CoefficientList[PolynomialRemainder[x^n - 1, g, x, Modulus → q], x]] = 0],
    n++];
   Print["Многочлен x^n-1, где n = ", n, " раскладывается на ",
    Factor[x^n - 1, Modulus → q]];
   Print["r = ", r, ", n = ", n, " следовательно размер порождающей матрицы будет
 равен ", (n - r), "x", n];
   Gp = Array[f, n - r];
   a = PolynomialRemainder[x^(n - 1), g, x, Modulus → q];
   f[1] = Reverse[CoefficientList[x^(n - 1) - a, x, Modulus → q]];
   For[i = n - 2, i ≥ r, i--,
     a = PolynomialRemainder[x^i, g, x, Modulus → 2];
     f[n - i] = Reverse[CoefficientList[x^(n - 1) + x^i - a, x, Modulus → q]];
    ]
    For[j = 2, j ≤ n - r, j++,
     Gp[[j, 1]] = 0;
    ]
    w = 0; (* ves *)
   R = PolynomialRemainder[b, g, x, Modulus → q]; (* ostatok ot deleniya b na g *)
   Rc = CoefficientList[R, x];
   For[i = 1, i ≤ Length[Rc], i++, (* podschitivaem ves ostatka*)
     If[Not[Rc[[i]] = 0], w++];
    ]
    If[w > t, Print["При передаче возникла ошибка!"]]
```

```
  l = 0; (* chislo ciklicheskih sdvigov *)
 While[w > t,
   w = 0;
   l = l + 1;
   Expand[b = x * b]; (* ciklicheskii sdvig *)
   R = PolynomialRemainder[b, g, x, Modulus → q]; (* ostatok ot deleniya b na g *)
   For[i = 1, i ≤ Length[CoefficientList[R, x]], i++, (* podschitivaem ves ostatka *)
     If[Not[CoefficientList[R, x][[i]] = 0], w++]
    ]
    Print[Expand[b], ", остаток: "R, ", вес: ", w](* vizualizaciya processa *)
  ]

  Print["Правильное кодовое слово: ",
   Reverse[CoefficientList[PolynomialMod[Expand[x^(n - l) (b + R)], x^n - 1, Modulus → q],
]
```

Сначала необходимо определить длину кодовых слов (n). Для этого система строит полином $x^n - 1$, где $n = 1$ и раскладывает без остатка таким образом, чтобы среди множителей был образующий многочлен. Если не раскладывается или раскладывается с остатком, число n увеличивается на единицу. Через конечное число шагов получится полином, который разложится без остатка и найдем число n. Имеется число r – степень образующего многочлена, т.о. размер порождающей матрицы будет $n - r \times n$. Можем построить матрицы и код. Но метод позволяет декодировать без использования кода.

Полученный полином делится на образующий многочлен, определяется вес остатка w и если $w > t$, циклически сдвигается полученный полином и опять делится. Система снова проверяет условие $w > t$, если верно повторяет описанную процедуру. При этом число сдвигов l харнится в переменной. Через конечное число шагов будет выполнено условие $w \leq t$. Конечный полнином циклически сдвигает в обратную сторону на l позиций и получается верное кодовое слово.

Результат работы этого модуля:

Многочлен x^n-1, где n = 15 раскладывается на $(1+x)\,(1+x+x^2)\,(1+x+x^4)\,(1+x^3+x^4)\,(1+x+x^2+x^3+x^4)$

r = 4, n = 15 следовательно размер порождающей матрицы будет равен 11x15

При передаче возникла ошибка!

$x^2+x^3+x^4+x^7+x^9+x^{12}+x^{15}$, остаток: $(1+x^2)$, вес: 2

$x^3+x^4+x^5+x^8+x^{10}+x^{13}+x^{16}$, остаток: $(x+x^3)$, вес: 2

$x^4+x^5+x^6+x^9+x^{11}+x^{14}+x^{17}$, остаток: $(1+x^2+x^3)$, вес: 3

$x^5+x^6+x^7+x^{10}+x^{12}+x^{15}+x^{18}$, остаток: $(1+x)$, вес: 2

$x^6+x^7+x^8+x^{11}+x^{13}+x^{16}+x^{19}$, остаток: $(x+x^2)$, вес: 2

$x^7+x^8+x^9+x^{12}+x^{14}+x^{17}+x^{20}$, остаток: (x^2+x^3), вес: 2

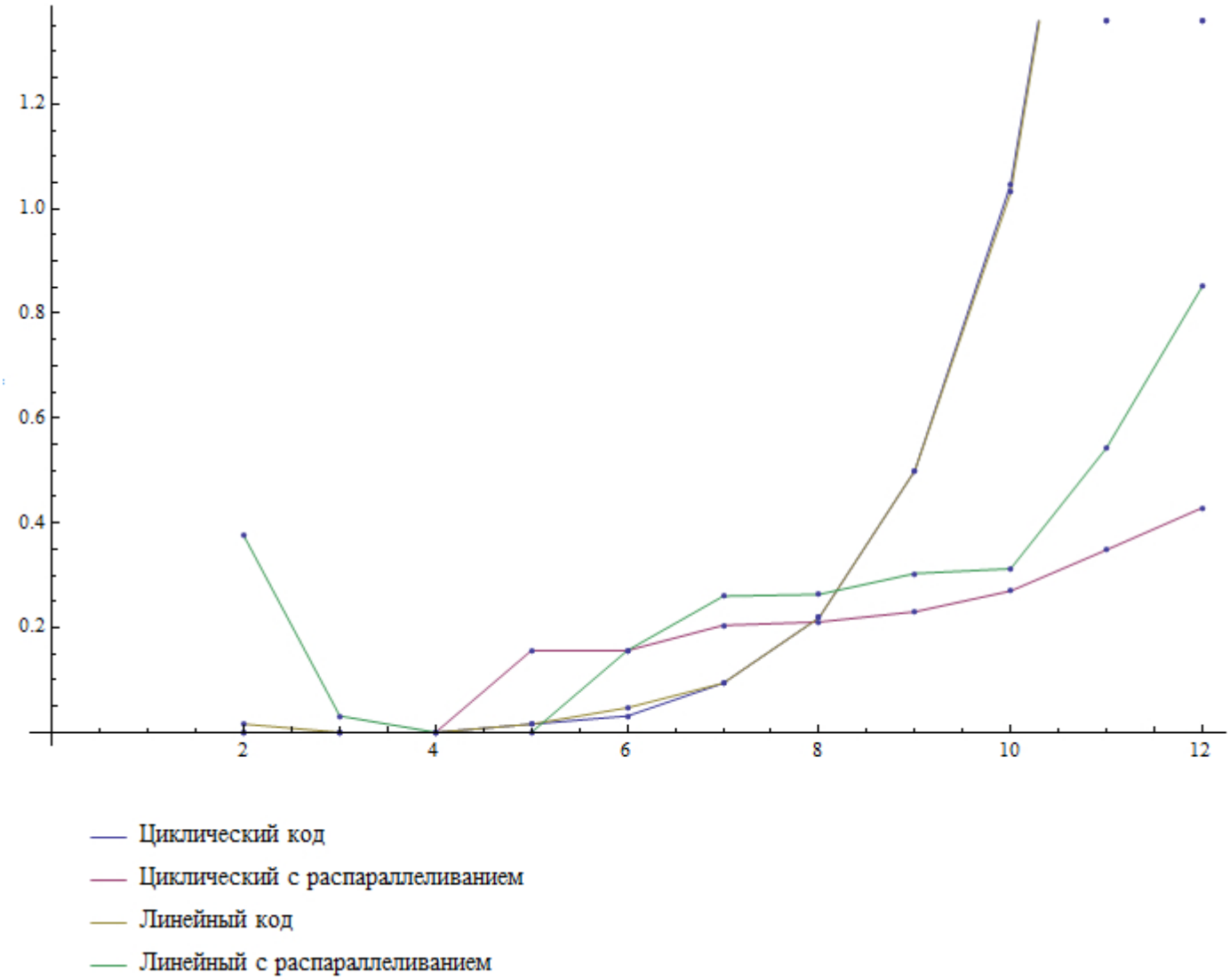
$x^8+x^9+x^{10}+x^{13}+x^{15}+x^{18}+x^{21}$, остаток: , вес: 1

Правильное кодовое слово: {1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0}

Сравнение результатов

В ходе работы я взял порождающую матрицу размера 11×15 для циклического и линейого кода и строил коды в пространствах $Z_2, Z_3, ..., Z_{12}$ в обычном режиме и в режиме с распараллеливанием. Считал время, которое затрачивается на построение.

Результаты я отразил в следующем графике:



По оси абсцисс пространство Z_n, по оси ординат – время. Исходя из этого графика можно сказать, что до пространства Z_8 распараллеливание процессов работает в ущерб и связано это с тем, что системе до начала работы кода необходимо "распределить нагрузку а на это нужно время. Т.е. при работе с небольшими кодами распараллеливание не эффективно, а уже в пространствах больше Z_8 эффективность на лицо. Я провел вычисления и в процентах определил это значение.

До пространства $Z_8 - 0\%$, а после нее

в Z_{10} и для циклического и для линейного кода в среднем 70%

в Z_{12} для линейного кода 79.5%

в Z_{12} для циклического кода 89.6%

Точка перехода для разных кодов разный. Она зависит от размеров порождающей матрицы и пространства в котором осуществляются вычисления. Я пытался определить эту закономерность, для этого взял данные маленьких размеров и строил код. Время затраченное на построение всегда равнялось нулю. А строить большие коды не удалось, т.к. не хватило ресурсов компьютера. Иногда вычисления проходили сутками, а результат не получал, потому что комьютер выключался от перегрева.

Заключение

Итак, в работе были продемонстрированы методы теории кодирования в компьютерной системе Wolfram Mathematica, работа функции распараллеливания и высчитаны примерное время выполнения. Удалось выяснить, что построение кода до некоторого пространства (обозначим Z_n) с включенной функцией распараллеливания выполняется дольше, нежели в обычном режиме и это связано с тем, что до начала выполнения функций методов система, грубо говоря, распределяет нагрузку между процессорами и на это программе тоже нужно время. А при построении кода в полях, которые состоят из элементов больше, чем в Z_n картина меняется в обратную сторону.

Помимо программы описанной в работе, имеется, так называемая, "пользовательская"версия. Её я написал в качестве бонуса и не могу не сказать пару слов о ней. Я планировал разместить эту программу в интернете для работы через браузер, чтобы каждый желающий без препятствий мог воспользоваться ею. Для этого Wolfram Research выпустила программу, которая называется CDFreader (устанавливается как расширение браузера). После ее установки возможно чтение "динамических документов". К сожалению, в настоящее время, не все методы которые были использованы в моем приложении поддерживаются программой. По этому поводу я связывался с сертифицированным инструктором и официальным представителем компании Wolfram Research. Он сказал, что работы в этом направлении ведутся. Может быть, скоро и получится реализовать такой функционал... К слову, версия Wolfram Mathematica для ipad, если я не ошибаюсь, разрабатывается третий год. Эту программу я всё равно выложу в открытый доступ.

Листинг

Описание использованных функций Mathematica

Графические функции:

- Button["Текст кнопки действие] - создает кнопку, при активации которой происходит действие.
- Checkbox[x] - представляет флажок. Если он установлен x = True, иначе False.
- Dynamic[x] - переменная x обновляется динамически.
- InputField[x, Number] - создает поле ввода, содержимым которого будет число.
- Grid[$\{x_1, x_2, ..., x_n\}, \{y_1, y_2, ..., y_n\}, ...$] - строит таблицу, где в первой строке x, во второй y...
- Print[] - вывод на экран.

Прочие функции:

- Append[X,y] - добавляет в множество X, элемент y.
- Cases[a,b,a,c, Except[a]] - возвращает элемент без a.
- CoefficientList[f[x,y], x] - возвращает коэффициенты функции f при x.
- Count[a, b, a, a, b, c, b, b] - возвращает число вхождений b.
- Exponent[f[x,y], x] - возвращает максимальную степень функции f при x.
- F[x,...] = Module[y,...,] - функция, которую можно вызвать, передав параметры в x. В модуле переменные, доступные только внутри нее, а после запятой описание действий.
- Head[x] - возвращает тип значения переменной x.
- IdentityMatrix[] - формирует единичную матрицу.
- Length[H] - возвращает длину H.
- Min[] - находит минимум.

- Mod[x, n] - возвращает число от x по модулю n.
- NullSpace[M] - находит фундаментальную систему решений матрицы M.
- PolynomialRemainder[p,q,x] - возвращает остаток от деления p на q по x.
- Position[a, b, a, a, b, c, b, b] - возвращает позиции b.
- RandomChoice[list,n] - возвращает список случайных элементов.
- RandomInteger[] - создает случайное целое число.
- Reverse[x,y] - возвращает элементы в обратном порядке.

Линейный код

```
In[9]:= "Проверочная матрица" Checkbox[Dynamic[ProverMatrica]]
"Включить распараллеливание" Checkbox[Dynamic[ParallelOn]]
InputField[Dynamic[q], Number, FieldHint -> "Введите q"]
InputField[Dynamic[k], Number, FieldHint -> "Введите k"]
InputField[Dynamic[n], Number, FieldHint -> "Введите n"]
{
 Button["Вычислить",
  DoMatrix[];
  CalculateMatrices[];
  Print[Column[{"Проверочная матрица " MatrixForm[H],
     "Порождающая матрица " MatrixForm[G]}]];
  ],
 Button["Показать код",
  DoLCod[];
  Print[LCodTable];
  ],
 Button["Минимальный вес",
  DoMDist[];
  Print["Минимальный вес кода равен ", LCodMin];
  Print["Вектор: ",
   LCodTable[[ Position[MnojestvoMinim, LCodMin][[1, 1]] + 1]]];
  ],
 Button["Таблица синдромов",
  DoSkTable[];
  Print["Таблица синдромов и соответствующих лидеров смежных классов \
", Grid[{{Grid[Sindrom], Grid[LiderSK]}}, Frame -> All], ";"];
  ]
```

```
 }
{Button["Отправить",
  Clear[Otpravlen, GetVector];
  Otpravlen = RandomChoice[LCodTable];
  Print["Отправлен: ", Otpravlen];
  SendVector[Position[LCodTable, Otpravlen][[1, 1]]];
  Print["Получен: ", GetVector];
  ],
 Button["Исправить",
  RepairVector[];
  Print["Был отправлен: ", GetVector];
  ]
 }
{
 Button["Подсчитать время",
  Vremya = 0;
  If[ParallelOn,
   Vremya = Timing[ParallelDo[DoMatrix[], {1}]][[1]];
   Vremya += Timing[ParallelDo[CalculateMatrices[], {1}]][[1]];
   Vremya += Timing[ParallelDo[DoLCod[], {1}]][[1]];
   Vremya += Timing[ParallelDo[DoMDist[], {1}]][[1]];
   Vremya += Timing[ParallelDo[DoSkTable[], {1}]][[1]];
   Print["Время работы ", Vremya];
   ,
   Vremya = Timing[DoMatrix[]][[1]];
   Vremya += Timing[CalculateMatrices[]][[1]];
   Vremya += Timing[DoLCod[]][[1]];
   Vremya += Timing[DoMDist[]][[1]];
   Vremya += Timing[DoSkTable[]][[1]];
   Print["Время работы ", Vremya];
   ]
  ],
 Button["Очистить",
  Clear[H, G, LCodTable, LCodMin, MnojestvoMinim, Sindrom, LiderSK]
  ]
 }

Out[9]= "Проверочная матрица" \!\(\*
CheckboxBox[Dynamic[$CellContext`ProverMatrica]]\)
```

```
Out[10]= "Включить распараллеливание" \!\(\*
CheckboxBox[Dynamic[$CellContext`ParallelOn]]\)

Out[11]= \!\(
InputFieldBox[Dynamic[$CellContext`q], Number,
FieldHint->"Введите q"]\)

Out[12]= \!\(
InputFieldBox[Dynamic[$CellContext`k], Number,
FieldHint->"Введите k"]\)

Out[13]= \!\(
InputFieldBox[Dynamic[$CellContext`n], Number,
FieldHint->"Введите n"]\)

Out[14]= {\!\(\*
ButtonBox["\<\"Вычислить\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`DoMatrix[]; $CellContext`\
CalculateMatrices[]; Print[
Column[{"Проверочная матрица " MatrixForm[$CellContext`H],
        "Порождающая матрица " MatrixForm[$CellContext`G]}]]; Null),
Evaluator->Automatic,
Method->"Preemptive"]\), \!\(\*
ButtonBox["\<\"Показать код\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`DoLCod[]; Print[$CellContext`LCodTable]\
; Null),
Evaluator->Automatic,
Method->"Preemptive"]\), \!\(\*
ButtonBox["\<\"Минимальный вес\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`DoMDist[]; Print[
      "Минимальный вес кода равен ", $CellContext`LCodMin]; Print[
      "Вектор: ",
Part[$CellContext`LCodTable, Part[
Position[$CellContext`MnojestvoMinim, $CellContext`LCodMin], 1,
          1] + 1]]; Null),
Evaluator->Automatic,
Method->"Preemptive"]\), \!\(\*
```

```
ButtonBox["\<\"Таблица синдромов\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`DoSkTable[]; Print[
      "Таблица синдромов и соответствующих лидеров смежных классов ",
Grid[{{
Grid[$CellContext`Sindrom],
Grid[$CellContext`LiderSK]}}, Frame -> All], ";"]; Null),
Evaluator->Automatic,
Method->"Preemptive"]\)}

Out[15]= {\!\(\*
ButtonBox["\<\"Отправить\"\>",
Appearance->Automatic,
ButtonFunction:>(
     Clear[$CellContext`Otpravlen, \
$CellContext`GetVector]; $CellContext`Otpravlen = RandomChoice[$\
CellContext`LCodTable]; Print[
      "Отправлен: ", $CellContext`Otpravlen]; $CellContext`SendVector[
Part[
Position[$CellContext`LCodTable, $CellContext`Otpravlen], 1,
        1]]; Print["Получен: ", $CellContext`GetVector]; Null),
Evaluator->Automatic,
Method->"Preemptive"]\), \!\(\*
ButtonBox["\<\"Исправить\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`RepairVector[]; Print[
      "Был отправлен: ", $CellContext`GetVector]; Null),
Evaluator->Automatic,
Method->"Preemptive"]\)}

Out[16]= {\!\(\*
ButtonBox["\<\"Подсчитать время\"\>",
Appearance->Automatic,
ButtonFunction:>($CellContext`Vremya = 0; If[$CellContext`ParallelOn, \
$CellContext`Vremya = Part[
Timing[
ParallelDo[
$CellContext`DoMatrix[], {1}]], 1]; AddTo[$CellContext`Vremya,
Part[
Timing[
```

```
ParallelDo[
$CellContext`CalculateMatrices[], {1}]],
          1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
ParallelDo[
$CellContext`DoLCod[], {1}]], 1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
ParallelDo[
$CellContext`DoMDist[], {1}]], 1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
ParallelDo[
$CellContext`DoSkTable[], {1}]], 1]]; Print[
        "Время работы ", $CellContext`Vremya]; Null, \
$CellContext`Vremya = Part[
Timing[
$CellContext`DoMatrix[]], 1]; AddTo[$CellContext`Vremya,
Part[
Timing[
$CellContext`CalculateMatrices[]], 1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
$CellContext`DoLCod[]], 1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
$CellContext`DoMDist[]], 1]]; AddTo[$CellContext`Vremya,
Part[
Timing[
$CellContext`DoSkTable[]], 1]]; Print[
        "Время работы ", $CellContext`Vremya]; Null]),
Evaluator->Automatic,
Method->"Preemptive"]\), \!\(\*
ButtonBox["\<\"Очистить\"\>",
Appearance->Automatic,
ButtonFunction:>Clear[$CellContext`H, $CellContext`G, \
$CellContext`LCodTable, $CellContext`LCodMin, \
$CellContext`MnojestvoMinim, $CellContext`Sindrom, \
$CellContext`LiderSK],
```

```
Evaluator->Automatic,
Method->"Preemptive"]\)}

G = {{1, 0, 1, 1}, {0, 1, 0, 1}};

H = Reverse[Mod[NullSpace[G], q]];

In[1]:= DoMatrix[] := Module[{i, j},
  If[Head[k] == Symbol || Head[n] == Symbol,
   Print["Введите размеры матрицы!"]];
  If[k >= n, Print["k не может быть больше n!"]]
   If[Head[k] == Integer && Head[n] == Integer && k < n,
    M = Array[Subscript[a, ##] &, {n - k, n}];
     If[ProverMatrica,
      For[i = 1, i <= n - k, i++,
       For[j = 1, j <= n, j++,
        M[[i, j]] = RandomInteger[{0, q - 1}];
         ]
       ]
      ,
      M = IdentityMatrix[{k, n}];
      For[i = 1, i <= k, i++,
       For[j = k + 1, j <= n, j++,
        M[[i, j]] = RandomInteger[{0, q - 1}];
         ]
       ]
      ]
     ]
  ]

In[2]:= CalculateMatrices[] := Module[{},
  If[ProverMatrica,
   H = M;
   G = Reverse[Mod[NullSpace[H], q]];
   ,
   G = M;
   H = Reverse[Mod[NullSpace[G], q]];
   ]
  ]
```

```
In[3]:= DoLCod[] := Module[{u, Cod},
  LCodTable = {};
  (* формирование линейного кода *)
  For[i = 0, i <= q^k - 1, i++,
   u = IntegerDigits[i, q, k];
   Cod = u.G;
   LCodTable = Append[ LCodTable, Mod[Cod, q] ];
   ];
  ]

In[4]:= DoMDist[] := Module[{Minim},
  MnojestvoMinim = {};
  LCodMin = {};
  For[i = 1, i <= q^k - 1, i++,
   Minim = Count[LCodTable[[i]], Except[0]];
   If[Minim != 0,
    MnojestvoMinim = Append[MnojestvoMinim, Minim];
    ];
   ];
  LCodMin = Min[MnojestvoMinim];
  ]

In[5]:=
getWeight[vector_] := Module[{i, res},
   res = 0;
   For[i = 1, i <= Length[vector], i++,
    If[ vector[[i]] != 0, res++ ];
    ];
   res
    ];


DoSkTable[] := Module[{AllLiders, SortLiders, i, j, l, Q},
  LiderSK = {};
  Sindrom = {};

  (* все возможные лидеры смежных классов.
  Здесь строятся векторы длины n. Их количество равно
  q^n-1. *)
  AllLiders = {};
```

```
  For[i = 0, i <= q^n - 1, i++,
   AllLiders = Append[AllLiders, IntegerDigits[i, q, n]];
   ];


  SortLiders = {};
  For[i = 0, i <= n, i++,
   Q = Select[AllLiders, getWeight[#] == i &];
   For[j = 1, j <= Length[Q], j++,
    SortLiders = Append[SortLiders, Q[[j]]];
    ];
   ];

  l = 1;
  While[Length[LiderSK] != q^n/q^k,
   If[Position[Sindrom, Mod[H.SortLiders[[l]], q]] == {},
    Sindrom = Mod[Append[Sindrom, H.SortLiders[[l]]], q];
    LiderSK = Append[LiderSK, SortLiders[[l]]]
    ];
   l++;
   ];



  ]

In[7]:=  SendVector[Poziciya_] := Module[{i, ErrorVector},
  ErrorVector = RandomChoice[{0}, n];
  l = LCodMin;
  If[EvenQ[LCodMin], l = LCodMin/2, If[l != 1, l = (LCodMin - 1)/2]];
  For[i = 1, i <= l, i++,
   ErrorVector[[RandomInteger[{1, Length[ErrorVector]}]]] =
    RandomInteger[{0, q - 1}]
   ];
  GetVector = Mod[LCodTable[[Poziciya]] + ErrorVector, q];
  ]

In[8]:= RepairVector[] := Module[{},
  GetVector =
    Mod[GetVector -
      LiderSK[[Position[Sindrom, Mod[H.GetVector, q]][[1, 1]]]], q];
```

```
  ]
```

Циклический код

```
  "Включить распараллеливание" Checkbox[Dynamic[ParallelOn]]
InputField[Dynamic[q], Number, FieldHint -> "Введите q"]
InputField[Dynamic[g]]
{
 Button["Вычислить",
  DoMatrix[];
  Print["n = ", n];
  Print["Порождающая матрица имеет вид ",  MatrixForm[G]];
  H = Reverse[Mod[NullSpace[G], q]];
  k = Length[G];
  Print["Проверочная матрица имеет вид ", MatrixForm[H]];
  ],
 Button["Показать код",
  DoLCod[];
  Print[LCodTable];
  ],
 Button["Минимальный вес",
  DoMDist[];
  Print["Минимальный вес кода равен ", LCodMin];
  Print["Вектор: ",
   LCodTable[[ Position[MnojestvoMinim, LCodMin][[1, 1]] + 1]]];
  ],
 Button["Таблица синдромов",
  DoSkTable[];
  Print["Таблица синдромов и соответствующих лидеров смежных классов \
", Grid[{{Grid[Sindrom], Grid[LiderSK]}}, Frame -> All], ";"];
  ]
 }
{Button["Отправить",
  Clear[Otpravlen, GetVector];
  Otpravlen = RandomChoice[LCodTable];
  Print["Отправлен: ", Otpravlen];
  SendVector[Position[LCodTable, Otpravlen][[1, 1]]];
  Print["Получен: ", GetVector];
  ],
 Button["Исправить",
  RepairVector[];
```

```
    Print["Был отправлен: ", GetVector];
    ]
  }
{
  Button["Подсчитать время",
    Vremya = 0;
    If[ParallelOn,
     Vremya = Timing[ParallelDo[DoMatrix[], {1}]][[1]];
     Vremya += Timing[ParallelDo[DoLCod[], {1}]][[1]];
     Vremya += Timing[ParallelDo[DoMDist[], {1}]][[1]];
     Vremya += Timing[ParallelDo[DoSkTable[], {1}]][[1]];
     Print["Время работы ", Vremya];
     ,
     Vremya = Timing[DoMatrix[]][[1]];
     Vremya += Timing[DoLCod[]][[1]];
     Vremya += Timing[DoMDist[]][[1]];
     Vremya += Timing[DoSkTable[]][[1]];
     Print["Время работы ", Vremya];
     ]
    ],
  Button["Очистить",
    Clear[H, G, LCodTable, LCodMin, MnojestvoMinim, Sindrom, LiderSK]
    ]
  }

DoMatrix[] := Module[{i, res},
    For[n = 1,
     Not[Length[
         CoefficientList[
          PolynomialRemainder[x^n - 1, g, x, Modulus -> q], x]] == 0], n++];
   G = {};
    res = Length[CoefficientList[g, x]];
    For[i = 1, i <= n - Exponent[g, x], i++,
     v = RandomChoice[{0}, n];
     v[[i ;; res + i - 1]] = CoefficientList[g, x];
     G = Append[G, v];
     ]
    ]

DoLCod[] := Module[{u, Cod},
```

```
  LCodTable = {};
  (* формирование линейного кода *)
  For[i = 0, i <= q^k - 1, i++,
   u = IntegerDigits[i, q, k];
   Cod = u.G;
   LCodTable = Append[ LCodTable, Mod[Cod, q] ];
   ];
  ]

DoMDist[] := Module[{Minim},
  MnojestvoMinim = {};
  LCodMin = {};
  For[i = 1, i <= q^k - 1, i++,
   Minim = Count[LCodTable[[i]], Except[0]];
   If[Minim != 0,
    MnojestvoMinim = Append[MnojestvoMinim, Minim];
    ];
   ];
  LCodMin = Min[MnojestvoMinim];
  ]

LiderSK = {};
Sindrom = {};
(* формирование лидеров смежных классов и соответствующих синдромов *)
\

(* функция сортировки по минимальному весу *)
getWeight[vector_] := Module[{i, res},
   res = 0;
   For[i = 1, i <= Length[vector], i++,
    If[ vector[[i]] != 0, res++ ];
    ];
   res
    ];
DoSkTable[] := Module[{AllLiders, SortLiders, i, j, l, Q},
  (* все возможные лидеры смежных классов *)
  AllLiders = {};
  For[i = 0, i <= q^n - 1, i++,
   AllLiders = Append[AllLiders, IntegerDigits[i, q, n]];
   ];
```

```
  (* отсортированные лидеры смежных классов по возрастанию веса *)
  SortLiders = {};
  For[i = 0, i <= n, i++,
   Q = Select[AllLiders, getWeight[#] == i &];
   For[j = 1, j <= Length[Q], j++,
    SortLiders = Append[SortLiders, Q[[j]]];
    ];
   ];
  (* построение синдрома и лидеров смежных классов;
  количество смежных классов не больше q^n/q^k *)
  l = 1;
  While[Length[LiderSK] != q^n/q^k,
   If[Position[Sindrom, Mod[H.SortLiders[[l]], q]] == {},
    Sindrom = Mod[Append[Sindrom, H.SortLiders[[l]]], q];
    LiderSK = Append[LiderSK, SortLiders[[l]]]
    ];
   l++;
   ];
  ]

 SendVector[Poziciya_] := Module[{i, ErrorVector},
  ErrorVector = RandomChoice[{0}, n];
  l = LCodMin;
  If[EvenQ[LCodMin], l = LCodMin/2, If[l != 1, l = (LCodMin - 1)/2]];
  For[i = 1, i <= l, i++,
   ErrorVector[[RandomInteger[{1, Length[ErrorVector]}]]] =
    RandomInteger[{0, q - 1}]
   ];
  GetVector = Mod[LCodTable[[Poziciya]] + ErrorVector, q];
  ]

RepairVector[] := Module[{},
  GetVector =
    Mod[GetVector -
      LiderSK[[Position[Sindrom, Mod[H.GetVector, q]][[1, 1]]]], q];
  ]

Quit[]
```

Альтернативный код обнаружения и исправления ошибки

```
InputField[Dynamic[q], Number,
 FieldHint -> "Введите q (\!\(\*SubscriptBox[\(Z\), \(q\)]\))"]
InputField[Dynamic[t], Number,
 FieldHint -> "Введите t (количество ошибок)"]
InputField[Dynamic[g]]
InputField[Dynamic[b]]
{
 Button["Декодировать",
  FindCod[];
  ]
 }

 FindCod[] := Module[{},
  r = Exponent[g, x];
  For[n = 1,
   Not[Length[
       CoefficientList[
        PolynomialRemainder[x^n - 1, g, x, Modulus -> q], x]] == 0],
   n++];
  Print["Многочлен \!\(\*SuperscriptBox[\(x\), \(n\)]\)-1, где n = ",
   n, " раскладывается на ",
   Factor[x^n - 1, Modulus -> q]];
  Print["r = ", r, ", n = ", n,
   " следовательно размер порождающей матрицы будет
   равен ", (n - r), "x", n];
  Gp = Array[f, n - r];
  a =  PolynomialRemainder[ x^(n - 1), g, x, Modulus -> q ];
  f[1] = Reverse[CoefficientList[x^(n - 1) - a, x, Modulus -> q]];
  For[i = n - 2, i >= r, i--,
    a =  PolynomialRemainder[ x^i, g, x, Modulus -> 2 ];
    f[n - i] =
     Reverse[CoefficientList[x^(n - 1) + x^i - a, x, Modulus -> q]];
    ]
   For[j = 2, j <= n - r, j++,
    Gp[[j, 1]] = 0;
    ]
   w = 0; (* ves *)
  R = PolynomialRemainder[b, g, x, Modulus -> q]; (*
  ostatok ot deleniya b na g *)
  Rc = CoefficientList[R, x];
```

```
For[i = 1, i <= Length[Rc], i++,(* podschitivaem ves ostatka*)
  If[Not[Rc[[i]] == 0], w++];
  ]
 If[w > t, Print["При передаче возникла ошибка!"]]

 l = 0;(* chislo ciklicheskih sdvigov *)
While[w > t,
  w = 0;
  l = l + 1;
  Expand[b = x*b];(* ciklicheskii sdvig *)
  R = PolynomialRemainder[b, g, x, Modulus -> q];(*
  ostatok ot deleniya b na g *)
  For[i = 1, i <= Length[CoefficientList[R, x]], i++,(*
    podschitivaem ves ostatka *)
    If[Not[CoefficientList[R, x][[i]] == 0], w++]
    ]
   Print[Expand[b], ", остаток: " R, ", вес: ", w](*
  vizualizaciya processa *)
  ]

 Print["Правильное кодовое слово: ",
  Reverse[CoefficientList[
    PolynomialMod[Expand[x^(n - l) (b + R)], x^n - 1, Modulus -> q],
     x]]]
]
```

Литература

1. Мак-Вильямс Ф. Дж. А., Слоэн Н. Дж. А. Теория кодов, исправляющих ошибки. – М.: Связь, 1979 г. – 744 с.

2. Берлекэмп Э. – Алгебраическая теория кодирования. – М.: Мир, 1971 г. – 477 с.

3. Сидельников В.М. - Теория кодирования. – 2006 г. – 289 с.

4. Gashkov I. Error Correcting Codes with Mathematica//Computational Science – ICCS 2003, Lecture Notes in Computer Science, Volume 2657. – 2003. – P. 737-746

5. Ромащенко А.Е., Румянцев А.Ю., Шень А. – Заметки по теории кодирования. – М.: МЦНМО, 2011 г. – 80 с.