Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
Омский государственный университет им. Ф.М. Достоевского
Математический факультет
Кафедра математической логики и логического программирования

Программная реализация БЧХ-кодов переменной длины

дипломная работа
студента гр. ММ-102
Антонова Вячеслава Ильича

Научный руководитель
к. ф.-м. наук
Рыбалов Александр Николаевич

Омск 2006

Содержание

Введение

С ростом использования электроники и компьютеров, растет потребность в быстрой и надежной передаче информации по каналам связи. В любом канале связи присутствуют шумы — сигналы, которые могут искажать передаваемую информацию. С этими искажениями можно бороться, преобразуя передаваемую информацию при помощи кода, который будет обнаруживать, и исправлять ошибки. В основе работы всех кодов лежит модифицирование исходных данных путем добавления некоторой избыточной информации, которая позволяет обнаруживать и исправлять ошибки, которые могли возникнуть при передаче кодированной информации по зашумленному каналу связи.

В 1969 году, при помощи искусственных спутников Mariner 6 и Mariner 7, было получено около двухсот фотографий Марса. Каждая фотография состояла из 658240 восьмибитных пикселей. Таким образом, для каждой фотографии требовалось около 5-ти миллионов бит информации. Эти биты были кодированы кодом, исправляющим ошибки, и переданы со скоростью 16200 бит в секунду на Землю, где они были успешно декодированы.

История кодирования, контролирующего ошибки, началась в 1948 г. публикацией знаменитой статьи Клода Шенона. Шенон показал, что с каждым каналом связано измеряемое в битах в секунду и называемое *пропускной способностью* канала число C, имеющее следующее значение. Если требуемая от системы связи скорость передачи информации R (измеряемая в битах с секунду) меньше C, то, используя коды, контролирующие ошибки, для данного канала можно построить такую систему связи, что вероятность ошибки на выходе будет сколь угодно мала.

В пятидесятые годы много усилий было потрачено на попытки построения в явном виде классов кодов, позволяющих получить обещанную сколь угодно малую вероятность ошибки, но результаты были скудными. В следующем десятилетии этой увлекательной задачи уделялось меньше внимания; вместо этого исследователи кодов предприняли атаку по нескольким основным направлениям.

Одно из направлений носило чисто алгебраический характер и преимущественно рассматривало *блоковые коды*. Первые блоковые коды были введены в 1950 г., когда Хемминг описал класс блоковых кодов, исправляющих одиночные ошибки. Несмотря на усиленные исследования, до конца пятидесятых годов не было построено лучшего класса кодов. Основной сдвиг произошел, когда Боуз и Рой-Чоудхури в 1960 г. и Хоквингем в 1959 г. нашли большой класс кодов, исправляющих кратные ошибки (коды БЧХ).

1 Постановка задачи

Была поставлена следующая задача:

1. создать программную реализацию алгоритмов кодирования и декодирования двоичного кода БЧХ длины $n = 2^m - 1$ и исправляюего t ошибок (m и t задаются пользователем);

2. создать программную оболочку для наглядной демонстрации пользы применения кодов, контролирующих ошибки.

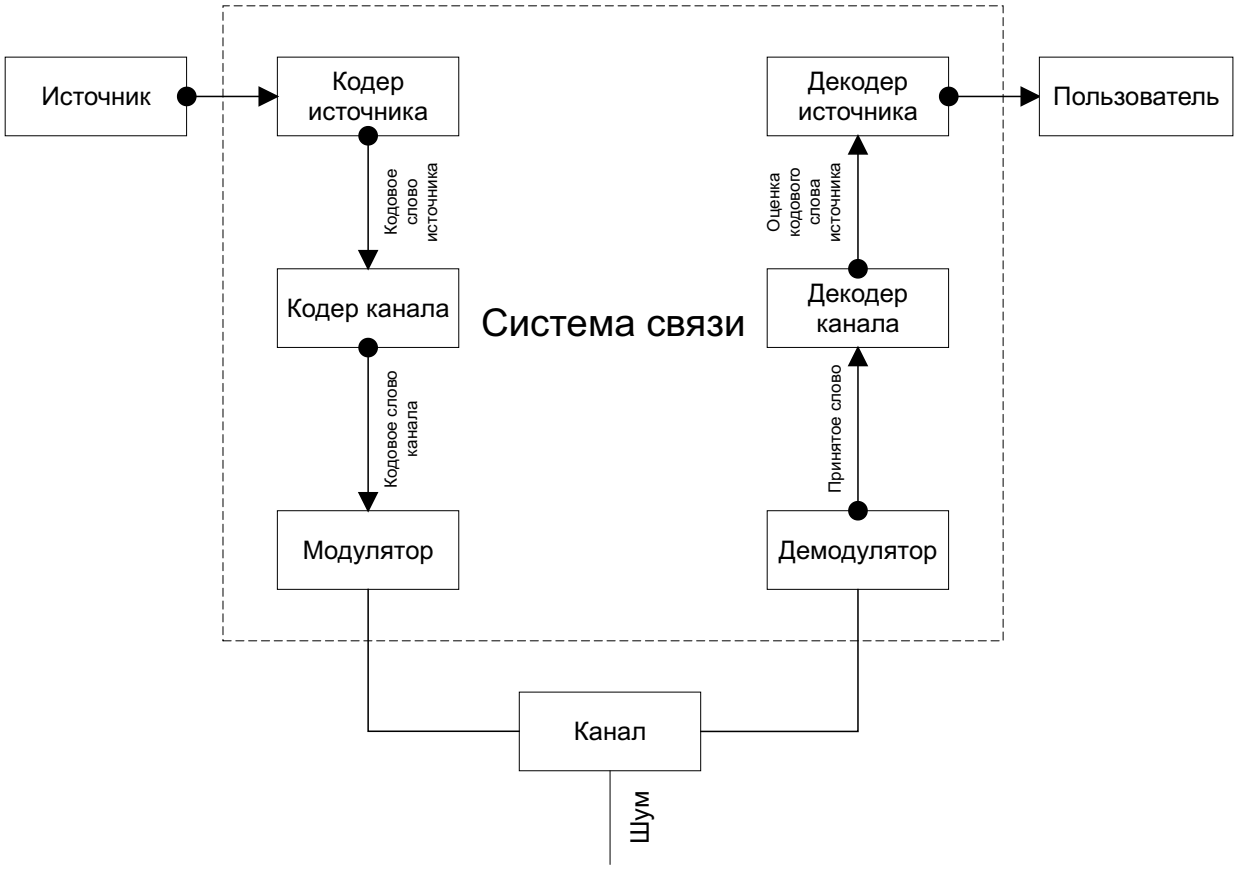
Для решения задачи была выбрана среда разработки Borland Delphi 2006.

2 Коды, контролирующие ошибки

2.1 Основные понятия

В настоящем разделе будут даны начальные сведения о кодах, контролирующих ошибки, необходимые для понимания изложенного далее материала.

Система связи соединяет источник данных с получателем данных посредством канала; примерами каналов являются микроволновые линии, коаксиальные кабели, телефонные линии и даже магнитные ленты. При проектировании системы связи разрабатываются устройства, подготавливающие вход и обрабатывающие выход каналов. Уже стало традицией подразделять основные функции системы связи так, как показано на блок-схеме:



Данные поступающие в систему связи от источника данных, прежде всего обрабатываются кодером источника, предназначенным для более компактного представления данных источника. Это промежуточное представление является последовательностью символов, которая называется *кодовым словом источника*. Затем данные обрабатываются кодером канала, преобразующим последовательность символов кодового слова источника в другую последовательность символов, называемую *кодовым словом канала*. Кодовое слово канала представляет собой новую, более длинную последовательность с большей, чем у кодового источника, избыточностью. Каждый символ кодового слова канала может быть представлен битом или, возможно, группой битов.

Далее модулятор преобразует каждый символ кодового слова канала в соответствующий аналоговый символ из конечного множества допустимых аналоговых символов. Последовательность аналоговых символов передается по каналу. Так как в канале возникают различного рода

шумы, искажения и интерференция, то выход канала отличается от его входа. Демодулятор преобразует каждый полученный на выходе канала сигнал в последовательность символов одного из кодовых слов канала. Каждый принятый символ является лучшей оценкой переданного символа, но из-за шума в канале демодулятор делает ошибки. Демодулированная последовательность символов называется *принятым словом*. Из-за ошибок символы принятого слова не всегда соответствуют символам кодового слова канала.

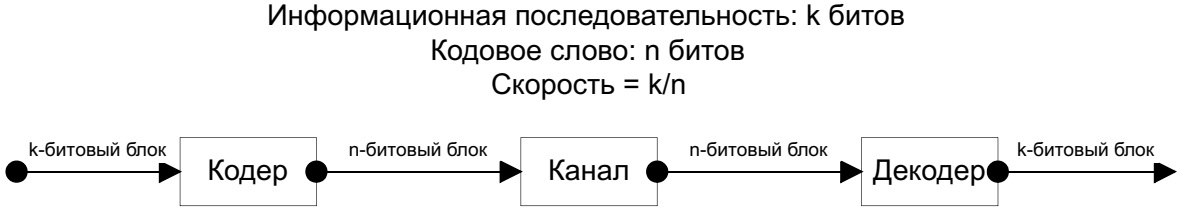
Декодер канала использует избыточность кодового слова канала для того, чтобы обнаружить и исправить ошибки в принятом слове, и затем выдает оценку кодового слова источника. Если все ошибки исправлены, то оценка кодового слова источника совпадает с исходным кодовым словом источника. Декодер источника выполняет операцию, обратную операцию кодера источника, результат которой поступает получателю.

Далее рассматривается только конструкция кодера и декодера канала — дисциплина, известная как *кодирование, контролирующее ошибки*. В дальнейшем кодер и декодер канала будут называться просто кодером и декодером соответственно.

Определение 2.1 ***Блоковый код*** *мощности M над алфавитом из q символов определяется как множество из M q-ичных последовательностей длины n, называемых кодовыми словами.*

Если $q = 2$, то символы называются битами, а код ***двоичным***. Обычно $M = q^k$ для некоторого целого k. Такой код называется (n, k)-*кодом*. Каждой последовательности из k q-ичных символов можно сопоставить последовательность из n q-ичных символов, являющуюся кодовым словом.

Блоковые коды являются одним из основных классов кодов.



Блоковый код задает блок из k информационных символов n-символьным кодовым словом.

Определение 2.2 *Величина $R = k/n$ называется* ***скоростью*** *блокового кода.*

О блоковом коде судят по трем параметрам: длине блока n, информационной длине k и минимальному расстоянию d^*. Минимальное расстояние является мерой различия двух наиболее похожих кодовых слов.

Определение 2.3 ***Расстояние по Хеммингу*** *между двумя q-ичными последовательностями x и y длины n называется число позиций, в которых они различны. Это расстояние обозначается через $d(x, y)$.*

Например, $d(01011, 00111) = 2$.

Определение 2.4 *Пусть* $\mathfrak{C} = \{c_i, i = 0, \ldots, M-1\}$ *— код. Тогда* ***минимальное расстояние*** *кода* $\mathfrak{C}$ *равно наименьшему из всех расстояний по Хеммингу между различными парами кодовых слов, т.е.*

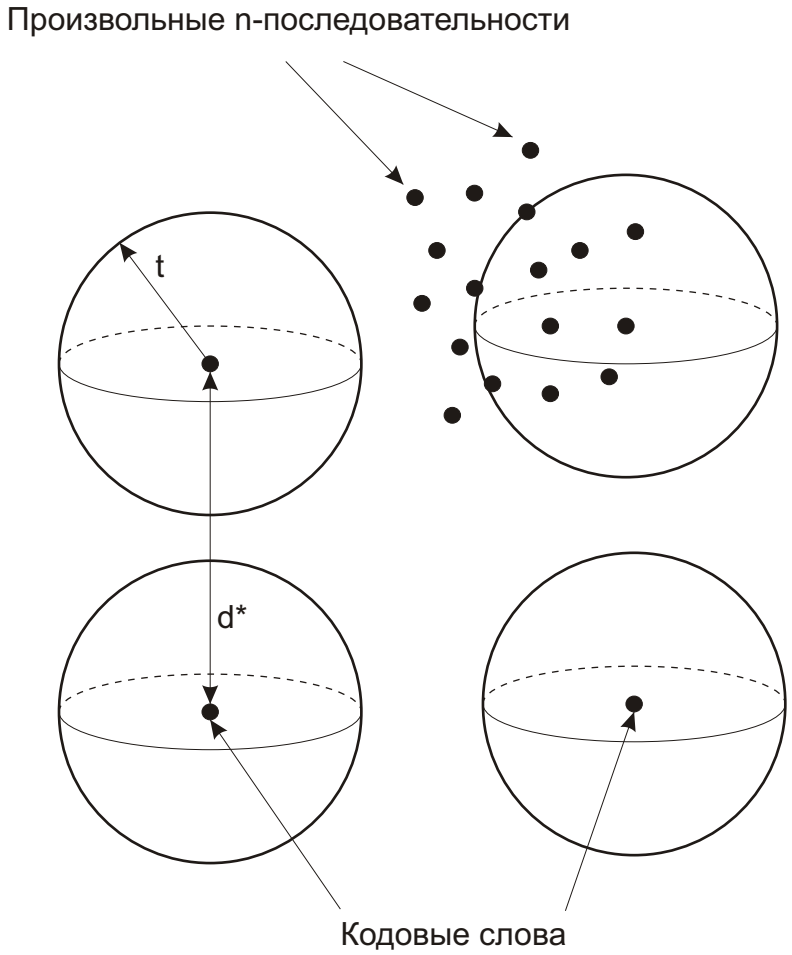
$$d^* = \min_{\substack{c_i, c_j \in \mathfrak{C} \\ i \neq j}} (c_i,\ c_j)$$

(n, k)-код с минимальным расстоянием d^* называется также (n, k, d^*)-кодом.

Предположим, что передано кодовое слово и в канале произошло t ошибок. Если расстояние от принятого слова до каждого другого кодового слова больше t, то декодер исправит эти ошибки, приняв ближайшее к принятому кодовое слово в качестве действительно переданного. Это всегда будет так если

$$d^* \geq 2t + 1.$$

Иногда удается исправить t ошибок даже когда это неравенство не выполняется. Однако если $d^* < 2t + 1$, то исправление любых t ошибок не может быть гарантировано, так как тогда оно зависит от того, какое слово было передано и какова была конфигурация из t ошибок внутри блока.



В пространстве всех q-ичных n-последовательностей выбрано множество n-последовательностей объявленных кодовыми словами. Если d^* — минимальное расстояние этого кода, а

t — наибольшее целое число удовлетворяющее условию $d^* \geq 2t+1$, то вокруг каждого кодового слова можно описать непересекающиеся сферы радиуса t, называемые ***сферами декодирования***. Принятые слова, лежащие внутри сфер, декодируются как кодовое слово, являющееся центром соответствующей сферы. Если произошло не более t ошибок, то принятое слово всегда лежит внутри соответствующей сферы и декодируется правильно.

Некоторые принятые слова, содержащие более t ошибок, попадут внутрь сферы, описанной вокруг другого кодового слова, и будут декодированы неправильно. Другие принятые слова, содержащие более t ошибок, попадут в промежуточные между сферами декодирования области. Этот факт можно интерпретировать двумя способами.

Неполный декодер декодирует только те принятые слова, которые лежат внутри сфер декодирования, описанных вокруг кодовых слов. Остальные принятые слова декодер объявляет нераспознаваемыми. Большинство используемых декодеров являются неполными декодерами.

Полный декодер декодирует каждое принятое слово в ближайшее кодовое слово. Если для принятого слова имеется несколько равноудаленных кодовых слов, то произвольно выбирается одно из них. Если происходит более t ошибок, то полный декодер часто декодирует неправильно, но бывают и случаи правильного декодирования. Полный декодер используется в тех случаях, когда лучше угадывать сообщение, чем вообще не иметь никакой его оценки.

2.2 Линейные блоковые коды

Большинство известных хороших кодов принадлежат классу ***линейных кодов***.

Определение 2.5 ***Линейный код*** *есть подпространство в векторном пространстве* $GF^n(q)$.

Таким образом, линейный код есть непустое множество n-последовательностей над $GF(q)$[1], называемых кодовыми словами, такое, что сумма двух кодовых слов является кодовым словом, и произведение любого кодового слова на элемент поля также является кодовым словом.

Определение 2.6 ***Систематическим кодом*** *называется код, у которого каждое кодовое слово начинается с информационных символов. Оставшиеся символы называются* ***проверочными символами***.

Принято говорить о систематическом коде, хотя это всегда подразумевает систематическое кодирование соответствующего кода.

Определение 2.7 ***Вес Хемминга*** $\omega(\mathrm{c})$ *кодового слова* c *равен числу его ненулевых компонент.* ***Минимальный вес кода*** ω^* *есть минимальный вес ненулевого кодового слова.*

Теорема 2.1 *Для линейного кода минимальное расстояние* d^* *равно минимальному весу* ω^*.

Теорема 2.2 ***(граница Синглтона).*** *Минимальное расстояние любого линейного* (n,k)*-кода удовлетворяет неравенству*

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

Определение 2.8 *Любой* (n,k)*-код с минимальным расстоянием, удовлетворяющим равенству*

$$d^* = 1 + n - k$$

называется ***кодом с максимальным расстоянием***.

[1] $GF(q)$ — поле Галуа, т.е. поле состоящее из q элементов

Граница Синглтона показывает, что для исправления t ошибок код должен иметь не менее $2t$ проверочных символов (2 проверочных символа на ошибку). Коды с максимальным расстоянием имеют точно $2t$ проверочных символов.

2.3 Циклические коды

Циклические коды являются подклассом в классе линейных кодов, удовлетворяющих дополнительному сильному структурному требованию. В силу этой структуры поиск хороших кодов, контролирующих ошибки, в классе циклических кодов оказался наиболее успешным. При этом в качестве математического аппарата, облегчающего поиск хороших кодов, была использована теория полей Галуа. Вне класса циклических кодов теория полей Галуа помогает мало.

Важность циклических кодов обусловлена тем, что заложенные в основу их определения идеи теории полей Галуа приводят к процедурам кодирования и декодирования, эффективным как с алгоритмической, так и с вычислительной точки зрения.

Определение 2.9 *Линейный (n, k)-код $\mathfrak{C}$ называется **циклическим**, если из того, что слово $c = (c_0, c_1, \ldots, c_{n-1})$ принадлежит коду $\mathfrak{C}$, следует, что слово $c' = (c_{n-1}, c_0, c_1, \ldots, c_{n-2})$ также принадлежит коду $\mathfrak{C}$.*

Кодовое слово c' получается циклическим сдвигом всех компонент слова c вправо на одну позицию. Каждый линейный код над $GF(q)$ длины n представляет собой подпространство пространства $GF^n(q)$, а циклический код является частным случаем подпространства, так как обладает дополнительным свойством цикличности.

Каждый вектор из $GF^n(q)$ можно представить многочленом от x степени не выше $n-1$. Компоненты вектора отождествляются с коэффициентами многочлена. Множество многочленов обладает структурой векторного пространства, идентичной структуре пространства $GF^n(q)$. Это же множество многочленов обладает структурой кольца $GF(q)[x]/(x^n-1)$. Как в кольце, в этом множестве определено умножение

$$p_1(x) \cdot p_2(x) = R_{x^n-1}\left[p_1(x)p_2(x)\right]$$

где $R_{h(x)}\left[f(x)g(x)\right]$ — есть остаток от деления на многочлен $h(x)$ произведения многочленов $f(x)$ и $g(x)$. Заметим, что в приведенное равенство входят произведения двух видов. Произведение в левой части является произведением в кольце $GF(q)[x]/(x^n-1)$, определенным через произведения к кольце $GF(q)[x]$ в правой части.

Циклический сдвиг может быть записан через умножение в этом кольце:

$$x \cdot p(x) = R_{x^n-1}[xp(x)].$$

Итак, если кодовые слова некоторого кода задаются в виде многочленов, то код является подмножеством кольца $GF(q)[x]/(x^n-1)$. Такой код является циклическим, если вместе с каждым кодовым словом $c(x)$ он содержит кодовый многочлен $x \cdot c(x)$.

Пусть $\mathfrak{C}$ — подмножество кольца $GF(q)[x]/(x^n-1)$ образует циклический код. Выберем в $\mathfrak{C}$ ненулевой кодовый многочлен наименьшей степени и обозначим его степень через $n-k$. Умножим этот многочлен на элемент поля, чтобы сделать его приведенным. Поскольку код $\mathfrak{C}$ линеен, результирующий многочлен тоже принадлежит $\mathfrak{C}$. Причем в коде не содержится приведенного многочлена такой же степени. Т.к. иначе в коде бы содержался многочлен меньшей степени, что противоречит выбору исходного многочлена.

Определение 2.10 *Единственный приведенный ненулевой многочлен наименьшей степени в коде* $\mathfrak{C}$ *называется* ***порождающим многочленом*** *кода* $\mathfrak{C}$ *и обозначается через* $g(x)$.

Теорема 2.3 *Циклический* (n,k)*-код состоит из всех произведений порождающего многочлена* $g(x)$ *степени* $n-k$ *на многочлены степени не выше* $k-1$.

Теорема 2.4 *Циклический код длины* n *с порождающим многочленом* $g(x)$ *существует тогда и только тогда, когда* $g(x)$ *делит* x^n-1.

Согласно теореме 2.4, для порождающего многочлена $g(x)$ любого циклического кода выполняется равенство

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

при некотором многочлене $h(x)$. Многочлен $h(x)$ называется ***проверочным многочленом***. Каждое кодовое слово $c(x)$ удовлетворяет равенству

$$R_{x^n-1}[h(x)c(x)]=0.$$

Пусть $c(x)$ обозначает переданное кодовое слово. Это значит что символами переданного слова были коэффициенты многочлена $c(x)$. Пусть многочлен $v(x)$ обозначает принятое слово, и пусть $e(x)=v(x)-c(x)$. Многочлен $e(x)$ называется ***многочленом ошибок***. Ненулевые коэффициенты этого многочлена стоят в тех позициях, где в канале произошли ошибки.

Представим информационную последовательность в виде многочлена $i(x)$ степени не выше $k-1$. Множество информационных многочленов можно отобразить в кодовые многочлены многими способами. Одним простым правилом является

$$c(x)=i(x)g(x).$$

Такой кодер является несистематическим, так как по многочлену $c(x)$ нельзя сразу установить $i(x)$. Систематическое правило кодирования имеет следующий вид:

$$c(x)=x^{n-k}i(x)+t(x),$$

где $t(x)=-R_{g(x)}\left[x^{n-k}i(x)\right]$.

И систематическое, и несистематическое правила кодирования дают одно и тоже множество кодовых слов, но соответствия между $i(x)$ и $c(x)$ различны.

Определение 2.11 ***Синдромный многочлен*** $s(x)$ *есть остаток от деления многочлена* $v(x)$ *на* $g(x)$*:*

$$s(x)=R_{g(x)}[v(x)]=R_{g(x)}[c(x)+e(x)]=R_{g(x)}[e(x)].$$

2.4 Коды Боуза-Чоудхури-Хоквингема

Коды БЧХ [1] представляют собой обширный класс кодов, способных исправлять несколько ошибок и занимают заметное место в теории и практике кодирования. Интерес к кодам БЧХ определяется тем, что они позволяют исправлять любое наперед заданное число ошибок и для них существуют эффективные алгоритмы кодирования и декодирования.

Порождающий многочлен циклического кода можно представить в виде

$$g(x)=\text{НОК}\left[f_1(x),f_2(x),\ldots,f_r(x)\right],$$

где $f_i(x), i = 1, \ldots, r$ — минимальные многочлены корней $g(x)$.

Пусть элементы $GF(q^m)$ $\gamma_1, \gamma_2, \ldots, \gamma_r$ — корни порождающего многочлена $g(x)$. Тогда

$$v(\gamma_i) = c(\gamma_i) + e(\gamma_i) = e(\gamma_i) = \sum_{j=0}^{n-1} e_j \gamma_i^j.$$

В результате мы получаем r уравнений, содержащих только величины, определяемые ошибками и не зависящие от кодового слова. Если эти уравнения можно разрешить относительно e_j, то мы сможем определить многочлен ошибок. Нужно выбрать γ_i таким образом, чтобы система r уравнений могла быть решена относительно e_i каждый раз, когда не более t неизвестных отличны от нуля.

Для произвольного циклического кода с порождающим многочленом $g(x)$, имеющим корни $\gamma_1, \ldots, \gamma_r$, определим компоненты синдрома

$$S_j = v(\gamma_j), j = 1, \ldots, r.$$

Эти элементы поля отличны от синдромного многочлена $s(x)$, но содержат эквивалентную информацию. Мы хотим подобрать $\gamma_1, \ldots, \gamma_r$ так, чтобы по $S_1, \ldots, S_r$ можно было найти t ошибок. В качестве таких γ_i можно взять степени $\{\alpha, \alpha^2, \ldots, \alpha^{2t}\}$ примитивного элемента α поля $GF(q^m)$.

Определение 2.12 *Пусть заданы q и m, и пусть β — любой элемент поля $GF(q^m)$ порядка n. Тогда для любого положительного целого числа t соответствующий* ***код БЧХ*** *является циклическим кодом длины n с порождающим многочленом*

$$g(x) = \text{НОК}\left(f_1(x), \ldots, f_{2t}(x)\right),$$

где $f_j(x)$ — минимальный многочлен элемента β^j.

Определение 2.13 *БЧХ коды длины $q^m - 1$ называются* ***примитивными***.

Таким образом, для того, чтобы построить порождающий многочлен примитивного БЧХ кода нужно:

1. Задать длину кода $n = q^m - 1$ и число t ошибок, которые необходимо исправлять.
2. Найти неприводимый многочлен степени m и построить поле $GF(q^m)$.
3. Найти примитивный элемент α в поле $GF(q^m)$.
4. Найти минимальные многочлены $f_i(x)$ для $\alpha^i, i = 1, \ldots, 2t$ над $GF(q)$.
5. Взять в качестве $g(x) = \text{НОК}\left(f_1(x), f_2(x), \ldots, f_{2t}(x)\right)$.

Более подробно построение порождающего многочлена примитивного кода БЧХ будет рассмотрено позже.

2.4.1 Декодер Питерсона-Горенстейна-Цирлера

Рассматриваемый в настоящем параграфе алгоритм был впервые предложен Питерсоном для двоичных кодов. Общий случай был разработан Горенстейном и Цирлером [1].

Предположим, что в основе конструкции кода БЧХ лежит элемент α поля $GF(q^m)$, возможно, не примитивный. Многочлен ошибок равен

$$e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \ldots + e_1x + e_0,$$

где не более t коэффициентов отличны от нуля. Пусть произошло ν ошибок, $0 \leq \nu \leq t$, и что этим ошибкам соответствуют неизвестные позиции $i_1, i_2, \ldots, i_\nu$. Многочлен ошибок можно записать в виде

$$e(x) = e_{i_1}x^{i_1} + e_{i_2}x^{i_2} + \ldots + e_{i_\nu}x^{i_\nu},$$

где e_{i_j} — величина j-й ошибки (в двоичном случае $e_{i_j} = 1$). Мы не знаем ни $i_1, \ldots, i_\nu$, ни $e_{i_1}, \ldots, e_{i_\nu}$. В действительности мы даже не знает числа ν. Для исправления ошибок нужно вычислить все эти числа. Чтобы получить компоненту синдрома S_1, нужно найди значение полученного многочлена в точке α:

$$S_1 = v(\alpha) = c(\alpha) + e(\alpha) = e(\alpha) = e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \ldots + e_{i_\nu}\alpha^{i_\nu}.$$

Определим для всех $j = 1, \ldots, \nu$ величины ошибок $Y_j = e_{i_j}$ и локаторы ошибок $X_j = \alpha^{i_j}$, где i_j — истинное положение j-й ошибки, а X_j — элемент поля, ассоциированный с этим положением. Т.к. порядок элемента α по условию равен n, то все локаторы рассматриваемой конфигурации ошибок различны.

В этих обозначениях S_1 запишется в виде

$$S_1 = Y_1X_1 + Y_2X_2 + \ldots + Y_\nu X_\nu.$$

Аналогично можно вычислить значения принятого многочлена при всех степенях элемента α, входящих в определение $g(x)$. Для $j = 1, \ldots, 2t$ определим синдромы равенствами

$$S_j = v(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j).$$

Тогда получим следующую систему из $2t$ уравнений относительно ν неизвестных локаторов $X_1, \ldots, X_\nu$ и ν неизвестных величин ошибок $Y_1, \ldots, Y_\nu$:

$$\begin{aligned}
S_1 &= Y_1X_1 + Y_2X_2 + \ldots Y_\nu X_\nu \\
S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots Y_\nu X_\nu^2 \\
&\vdots \\
S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots Y_\nu X_\nu^{2t}.
\end{aligned}$$

В силу определения синдрома эта система должна иметь хотя бы одно решение (на самом деле решение единственно).

Рассмотрим многочлен от x:

$$\Lambda(x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1}x^{\nu-1} + \ldots + \Lambda_1 x + 1,$$

известный под названием ***многочлена локаторов ошибок*** и определяемый как многочлен, корнями которого являются обратные к локаторам ошибок величины X_l^{-1} для $l = 1, \ldots, \nu$. Итак,

$$\Lambda(x) = (1 - xX_1)(1 - xX_2)\ldots(1 - xX_\nu).$$

Умножим обе части равенства, определяющих этот многочлен, на $Y_l X_l^{j+\nu}$ и положим $x = X_l^{-1}$. Тогда, получим

$$0 = Y_l X_l^{j+\nu} \left(1 + \Lambda_1 X_l^{-1} + \Lambda_2 X_l^{-2} + \ldots + \Lambda_{\nu-1} X_l^{-1-\nu} + \Lambda_\nu X_l^{-\nu}\right),$$

или

$$Y_l \left(X_l^{j+\nu} + \Lambda_1 X_l^{j+\nu-1} + \ldots + \Lambda_\nu X_l^j\right) = 0.$$

Это равенство верно при любых l и j. Просуммируем эти равенства по l от 1 до ν

$$\sum_{l=1}^{\nu} Y_l \left(X_l^{j+\nu} + \Lambda_1 X_l^{j+\nu-1} + \ldots + \Lambda_\nu X_l^j\right) = 0,$$

или

$$\sum_{l=1}^{\nu} Y_l X_l^{j+\nu} + \sum_{l=1}^{\nu} \Lambda_1 Y_l X_l^{j+\nu-1} + \ldots + \sum_{l=1}^{\nu} \Lambda_\nu Y_l X_l^j = 0.$$

Заметим, что каждая сумма является компонентом синдрома. Поэтому уравнение приводится к виду

$$S_{j+\nu} + \Lambda_1 S_{j+\nu-1} + \ldots + \Lambda_\nu S_j = 0.$$

Получаем систему уравнений

$$\Lambda_1 S_{j+\nu-1} + \ldots + \Lambda_\nu S_j = -S_{j+\nu}, j = 1, \ldots, \nu,$$

которую можно записать в матричном виде

$$\begin{bmatrix} S_1 & S_2 & S_3 & \ldots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & S_4 & \ldots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & S_4 & \ldots & S_{\nu+1} & S_{\nu+2} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ S_\nu & S_{\nu+1} & S_{\nu+2} & \ldots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix}.$$

Теорема 2.5 *Матрица компонент синдрома*

$$M = \begin{bmatrix} S_1 & S_2 & \cdots & S_u \\ S_2 & S_3 & \cdots & S_{u+1} \\ \vdots & \vdots & & \vdots \\ S_u & S_{u+1} & \cdots & S_{2u-1} \end{bmatrix}$$

невырождена, если u равно числу v произошедших в действительности ошибок. Матрица является вырожденной, если u больше v.

Приступим к описанию алгоритма декодирования Питерсона-Горенстейна-Цирлера. Пусть α — элемент поля $GF(q^m)$, по которому строился код БЧХ, а t — количество ошибок, исправляемых кодом.

1. На вход алгоритму поступает принятое слово $v(x)$.
2. Вычисляем компоненты синдрома $S_i = v(\alpha^i),\ i = 1, \ldots, 2t$.

3. Полагаем $v = t$.

4. Строим матрицу
$$M = \begin{bmatrix} S_1 & S_2 & \cdots & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ \vdots & \vdots & & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{bmatrix}$$

5. Вычисляем определитель матрицы M. Если он равен нулю, уменьшаем v на единицу и возвращаемся к шагу 4.

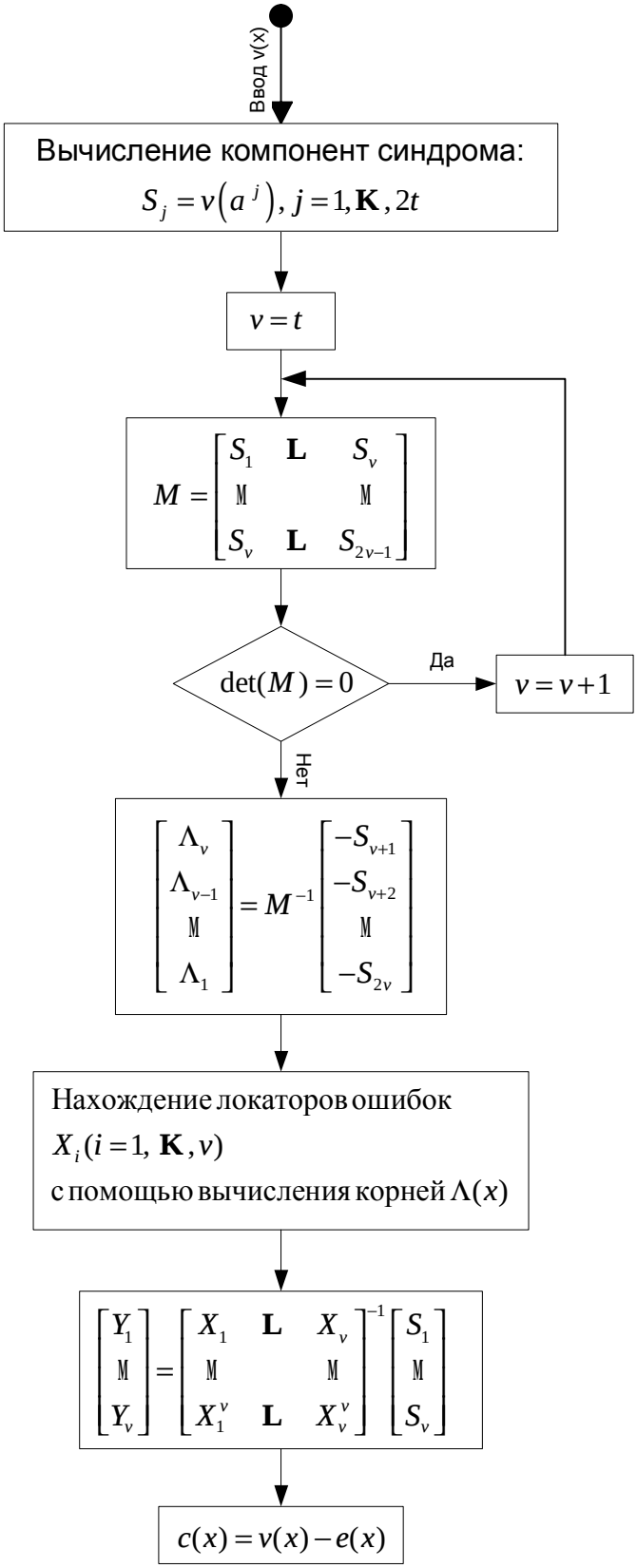
6. Обращаем матрицу M и вычисляем коэффициенты многочлена $\Lambda(x)$:
$$\begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = M^{-1} \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v} \end{bmatrix}.$$

7. Вычисляем корни многочлена $\Lambda(x)$. Поскольку число элементов поля конечно, обычно корни ищут *процедурой Ченя*. Эта процедура заключается в последовательном вычислении $\Lambda(\alpha^i)$ для каждого i и проверки полученных значений на нуль.

8. Найдя корни, найдем локаторы ошибок X_j (корни многочлена $\Lambda(x)$ являются обратными к локаторам ошибок).

9. Если код код двоичный, то ошибки Y_j известны. В противном случае вычислим их:
$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_v \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & \cdots & X_v \\ X_1^2 & X_2^2 & \cdots & X_v^2 \\ \vdots & \vdots & & \vdots \\ X_1^v & X_2^v & \cdots & X_v^v \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_v \end{bmatrix}.$$

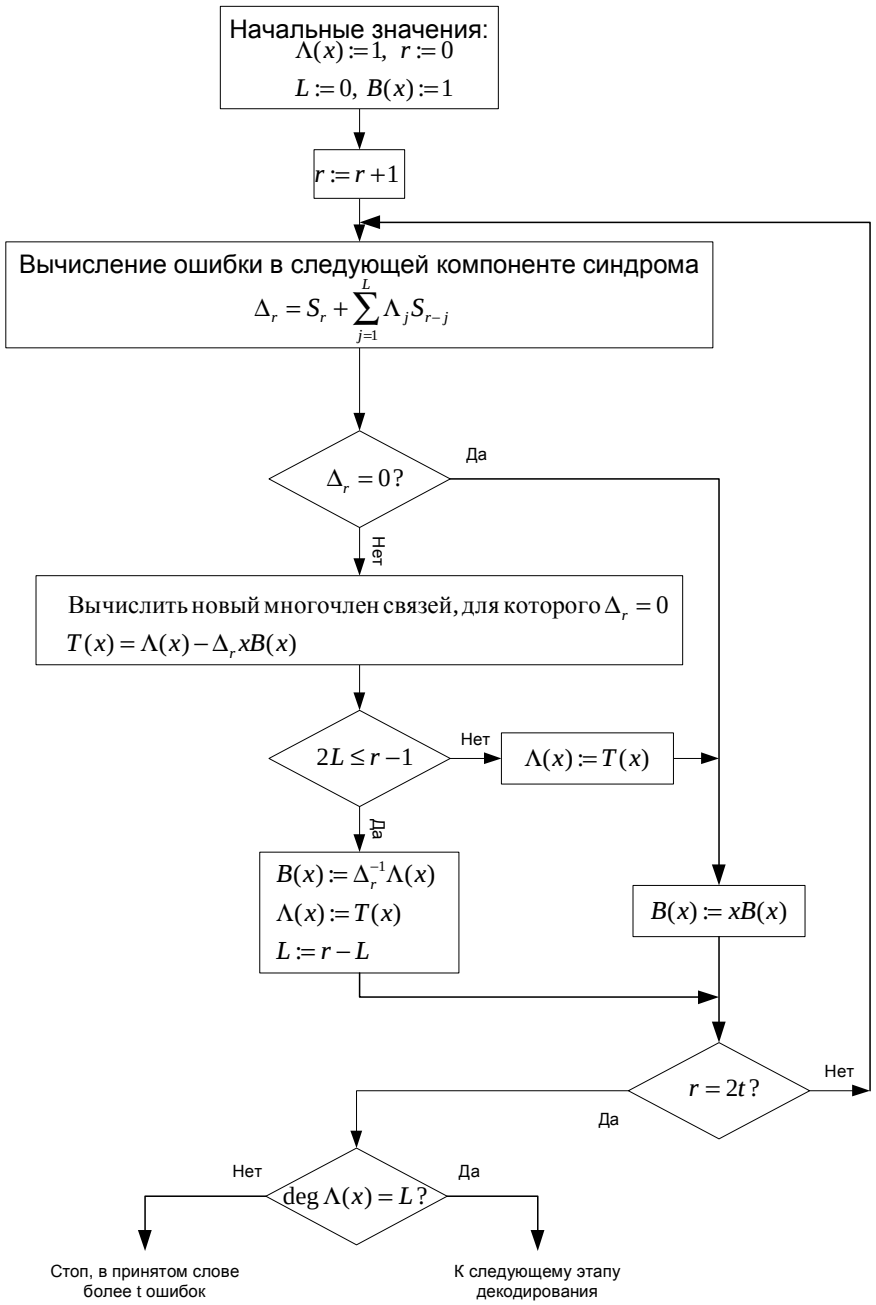
10. Исправляем в полученном слове ошибки, и получаем на выходе алгоритма кодовое слово.


Ввод v(x)
Вычисление компонент синдрома:
$S_j = v\left(a^j\right), j = 1, \mathbf{K}, 2t$
$v = t$
$M = \begin{bmatrix} S_1 & \mathbf{L} & S_v \\ \mathbf{M} & & \mathbf{M} \\ S_v & \mathbf{L} & S_{2v-1} \end{bmatrix}$
$\det(M) = 0$
Да
$v = v + 1$
Нет
$\begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \mathbf{M} \\ \Lambda_1 \end{bmatrix} = M^{-1} \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \mathbf{M} \\ -S_{2v} \end{bmatrix}$
Нахождение локаторов ошибок
$X_i (i = 1, \mathbf{K}, v)$
с помощью вычисления корней $\Lambda(x)$
$\begin{bmatrix} Y_1 \\ \mathbf{M} \\ Y_v \end{bmatrix} = \begin{bmatrix} X_1 & \mathbf{L} & X_v \\ \mathbf{M} & & \mathbf{M} \\ X_1^v & \mathbf{L} & X_v^v \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ \mathbf{M} \\ S_v \end{bmatrix}$
$c(x) = v(x) - e(x)$

2.4.2 Алгоритм Беркелэмпа-Месси

Декодер Питерсона-Горенстейна-Цирлера предполагает обращение двух матриц размера $t \times t$. Вычислительная работа при больших t может оказаться чрезмерно большой. Однако обращения первой матрицы можно избежать применяя алгоритм Беркелэмпа-Месси [1].

На вход алгоритму подаются $2t$ компонент синдрома $S_1, \ldots, S_{2t}$. На выходе алгоритм выдает многочлен локаторов ошибок $\Lambda(x)$.



2.4.3 Алгоритм Беркелэмпа проверки неприводимости многочлена

Данный алгоритм [2] понадобится при построении порождающего многочлена кода БЧХ, так как на одном из шагов необходимо найти неприводимый в поле $GF(q)$ многочлен степени m.

На вход алгоритму подается многочлен $f(x)$, $deg(f) = n$. На выходе алгоритм выдает заключение о неприводимости многочлена $f(x)$.

1. Если $n < 2$, то сразу делаем заключение о неприводимости многочлена $f(x)$. Иначе, переходим к следующему шагу.

2. Проверим, что многочлен $f(x)$ не содержит кратных сомножителей. Для этого находим наибольший общий делитель многочлена и его производной.

$$d(x) = \text{НОД}(f(x), f'(x))$$

   Если $d(x) = 1$, то $f(x)$ не содержит кратных сомножителей, и мы переходим к следующему шагу алгоритма. Иначе, заключаем, что многочлен $f(x)$ приводим.

3. Построим $n \times n$-матрицу B, строки которой — остатки от деления x^{iq} на $f(x)$, где $i = 0, \ldots, n-1$.

4. Вычислим ранг матрицы $(B - I)$, где I — единичная $n \times n$-матрица. Пусть он равен r. Многочлен $f(x)$ неприводим, если $n - r = 1$. В противном случае многочлен является приводимым.

3 Программная реализация кодов БЧХ

3.1 Арифметика конечных полей

Конечные поля — это основа кодов БЧХ. Построение порождающего многочлена, а также реализация эффективных алгоритмов кодирования и декодирования кодов БЧХ, было бы невозможно без реализации арифметики конечных полей.

В данной дипломной работе рассматриваются только двоичные коды. Поэтому необходимо реализовать арифметику поля $GF(2^m)$.

Конечное поле $GF(2^m)$ можно представить как остатки от деления на неприводимый двоичный многочлен степени m, т.е. многочлен, коэффициентами которого могут быть только 0 и 1.

Двоичные многочлены реализует класс *TBinaryPolynom* в модуле *Polynom.pas*.
Класс *TBinaryPolynom* является потомком абстрактного класса *TFieldItem*, для которого определены такие операции как: сложение, умножение, вычитание, деление, присваивание и другие.

Коэффициенты многочлена хранятся в массиве размером 4096 байта, массив выровнен по границе в 16 байт и разбит на блоки по 16 байт. Каждый бит массива представляет собой коэффициент при соответствующей степени. Максимально поддерживаемая степень многочлена равна 32767. Кроме коэффициентов в поле класса хранится степень многочлена.

Блоки разделяются на активные и не активные. Активные блоки содержат коэффициенты многочлена, степень которого хранится в поле класса. Содержание неактивных блоков может быть произвольным. Класс следит за тем, чтобы неиспользуемые биты в последнем активном блоке всегда были равны нулю.

Операции сложения и вычитания многочленов A(x) и B(x) реализуются следующим образом:

1. Определяется минимум m количества активных блоков в обоих операндах.
2. Первые m активных блоков обоих операндов складываются при помощи SSE-инструкции xorps.
3. Оставшиеся активные блоки переносятся в результат копированием.
4. Если оба операнда имели одинаковую степень, запускается процедура уточнения степени многочлена, так как в результате операции она уменьшилась.

Таким образом, эффективнее прибавлять к многочлену большей степени, многочлен меньшей степени. Это позволяет исключить операции копирования оставшихся активных блоков.

Операция умножения реализуется при помощи обычного "раскрытия скобок", но производится оно не побитово, а побайтово, используя специально подготовленную таблицу умножения.

Деление же реализуется классическим делением уголком.

Наследником класса *TBinaryPolynom* является класс *TGF2Item*. В нем арифметические операции выполняются по модулю многочлена, заданного в поле класса.

3.2 Генерация неприводимых многочленов

Для реализации кода БЧХ необходимо генерировать неприводимый многочлен заданной степени. Генерация неприводимого многочлена производится в методе *BuildIrreducible* класса *TBinaryPolynom*

$$function\ BuildIrreducible(N:\ LongWord):\ TBinaryPolynom,$$

где N — степень неприводимого многочлена, который нужно построить.

Снегерированные ранее неприводимые многочлены хранятся в файле *Irreducible.cache*. Поэтому, в начале проверяется нет ли в нем многочлена степени N. Если такой многочлен найден, то он принимается за неприводимый и работа метода заканчивается.

Если многочлен степени N в этом файле не найден, то начинается поиск неприводимого многочлена.

Создается многочлен $f(x) = x^N + a_{N-1}x^{N-1} + \ldots + a_1 x + 1$. После этого запускается бесконечный цикл, состоящий из следующих шагов:

1. коэффициенты $a_i,\ i = 1,\ \ldots,\ N-1$ многочлена $f(x)$ случайным образом заполняются 0 или 1;
2. многочлен $f(x)$ проверяется на неприводимость;
3. если $f(x)$ неприводим, то он записывается в файл *Irreducible.cache* и происходит выход из цикла.

Проверка многочлена $f(x)$ на неприводимость осуществляется методом *Irreducible* класса *TBinaryPolynom*, который производит ее алгоритмом Беркелэмпа.

3.3 Генерация порождающего многочлена БЧХ-кода

Порождающий многочлен БЧХ-кода строит метод *BuildCode* класса *TBCHExCode*

$$procedure\ TBCHExCode.BuildCode(m,\ t:\ LongWord),$$

где m — задает длину кода $n = 2^m - 1$, а t — количество ошибок, исправляемых кодом. В силу ограничений в реализации двоичных многочленов классом *TBinaryPolynom*, параметр m не может быть больше 15.

Информация о построенных ранее кодах БЧХ хранится в файле *BCH.cache*. Если информация (в том числе, порождающий многочлен) о коде БЧХ с параметрами m и t есть в этом файле, то она читается из него и работа метода прекращается.

Как уже говорилось ранее, для построения порождающего многочлена $g(x)$ нужно:

1. найти неприводимый многочлен степени m;
2. найти примитивный элемент α в поле $GF(2^m)$;
3. найти минимальные многочлены $f_i(x)$ для $\alpha^i,\ i = 1,\ \ldots\ 2t$ над полем $GF(2)$;
4. положить $g(x) = \mathrm{НОК}(f_1(x),\ \ldots,\ f_{2t}(x))$.

Неприводимый многочлен степени m строится методом *BuildIrreducible* класса *TBinaryPolynom*.

Рассмотрим поиск примитивного элемента. Поле $GF(2^m)$ можно представить с помощью остатков от деления на неприводимый многочлен степени m. В начале α полагается равным многочлену x. Затем в цикле вычисляются степени элемента α и заносятся в таблицу. Если при очередном вычислении степени элемента α получилась 1, и были перебраны все элементы мультипликативной группы поля $GF(2^m)$, то элемент α является примитивным. Если же были перебраны не все элементы, то α рассматривается как число, к нему прибавляется 1, и цикл начинается заново. В результате, не только определяется примитивный элемент α поля $GF(2^m)$, но и строится таблица его степеней, которая используется при декодировании.

Теперь, перейдем к поиску минимальных многочленов для $\alpha^i,\ i = 1,\ \ldots,\ 2t$ над $GF(2)$, и вычислению порождающего многочлена $g(x)$ [2]. Сначала создается таблица, в которой будет хранится информация об обработанных степенях элемента α, а $g(x)$ полагается равным 1. Затем начинается цикл из $2t$ итераций. Пусть i — номер итерации.

1. Если α^i уже обработан, происходит переход к следующей итерации.
2. Для элемента $\beta = \alpha^i$ выбираются все различные сопряженные с ним элементы относительно поля $GF(2)$. Т.е. элементы $\beta,\ \beta^2,\ \beta^{2^2},\ \ldots,\ \beta^{2^{d-1}}$, где d такое что $\beta = \beta^{2^d}$. Все эти элементы помечаются как *обработанные*.
3. Минимальный многочлен для сопряженных элементов есть
$$f(x) = (x - \beta)(x - \beta^2)\ldots(x - \beta^{2^{d-1}}).$$
4. Многочлен $g(x)$ домножается на многочлен $f(x)$.

Домножая, по ходу цикла, $g(x)$ на минимальные многочлены $f(x)$ мы вычислили $\mathrm{НОК}(f_1(x), \ldots, f_{2t}(x))$. Дело в том, что минимальные многочлены $f(x)$ неприводимы над полем $GF(2)$, и сопряженным относительно поля элементам соответствует один и тот же минимальный многочлен.

После того, как порождающий многочлен построен, в файл *BCH.cache* записывается следующая информация о коде БЧХ с параметрами m и t:

- неприводимый многочлен степени m, по которому строилось поле $GF(2^m)$;
- примитивный элемент α;
- порождающий многочлен $g(x)$.

3.4 Реализация кодера и декодера

Класс *TBCHExCode*, реализующий код БЧХ, наследует методы *Encode* и *Decode* родительского абстрактного класса *TCode*. Метод *Encode* преобразует информационное слово длины k в кодовое слово длины n. Метод *Decode* преобразует принятое слово длины n в информационное слово длины k, используя алгоритм Беркелэмпа-Месси.

Класс TBCHExCode, в зависимости от наличия директивы условной компиляции *$DEFINE SYSTEM_CODE*, производит систематическое или не систематическое кодирование и декодирование.

Кодер и декодер реализованы в модулях *Code.pas*, и *CodeThreads.pas*.

Модуль Code.pas содержит следующие продедуры выполняющие кодирование и декодирование данных:

- *procedure RawDecodeStream(InStream, OutStream: TStream; Code: TCode);*
  Выполняет декодирование потока *InStream* кодом, указанным в параметре *Code*, в поток *OutStream*.
- *procedure RawEncodeStream(InStream, OutStream: TStream; Code: TCode);*
  Выполняет кодирование потока InStream кодом, указанным в параметре *Code*, в поток OutStream без записи заголовка, содержащего информацию о коде *Code*.

- *procedure DecodeStream(InStream, OutStream: TStream);*
  Выполняет декодирование потока InStream в поток OutStream. Код, для декодирования, загружается из заголовка в потоке InStream.

- *procedure EncodeStream(InStream, OutStream: TStream; Code: TCode);*
  Выполняет кодирование потока InStream кодом, указанным в параметре *Code*, в поток OutStream с записью заголовка, содержащего информацию о коде *Code*.

- *procedure DecodeFile(InputFileName, OutputFileName: String);*
  procedure EncodeFile(InputFileName, OutputFileName: String; Code: TCode);
  Выполняют кодирование и декодирование файлов. Действуют аналогично процедурам *EncodeStream* и *DecodeStream*.

Модуль *CodeThreads.pas* содержит классы *TCodecThread* и *TFileCodecThread*, которые реализуют кодирование и декодирование потоков и файлов соответственно, с возможностью прерывания операции. Кроме того, во время выполнения операции, они периодически вызывают обработчик события *OnProgress*, в котором сообщают количество обработанных бит. Это позволяет отображать на экране пользователя статистику и оценивать время необходимое для завершения операции.

4 Magic Coder

Magic Coder — это оболочка для демонстрации работы кодов, контролирующих ошибки. Данная версия поддерживает следующие реализации кодов:

1. *Двоичный (15, 5)-код БЧХ, исправляющий три ошибки.* Порождающий многочлен кода $g(x) = x^{10}+x^8+x^5+x^4+x^2+x+1$. Используется табличное кодирование и декодирование. Таблицы строятся по время запуска программы.

2. *Двоичный код Рида-Маллера с параметрами m и r.* Параметры задаются пользователем. Используется матричное кодирование. Декодирование осуществляется мажорандным голосованием.

3. *Двоичный код БЧХ с параметрами m и t.* Параметры задаются пользователем. Кодирование производится умножением на порождающий многочлен. Декодирование производится по алгоритму Беркелэмпа-Месси.

4.1 Структура программы

Программа основана на идеях объектно-ориентированного программирования. Что делает ее проще и гибче. Все классы, реализующие коды, контролирующие ошибки, являются наследниками базового класса TCode. Процедуры и классы кодера и декодера оперируют базовым классом TCode. Это позволяет легко вносить поддержку новых кодов, контролирующих ошибки.

Кратко пройдемся по основным модулям программы:

- *BitsUtils.pas*
  Содержит класс TBits, который служит для работы с упакованным массивом бит. Данный класс используется для хранения информационных, кодовых и принятых слов.

- *Code.pas*
  Этот модуль — сердце программы. В нем находится описание абстрактного класса TCode, являющегося базовым классом, для всех реализаций кодов, контролирующих ошибки; классы для чтения и записи потока бит, функции регистрации кодов в потоковой системе, процедуры реализующие кодер и декодер. Класс TCode содержит методы кодирования и декодирования, а также свойства описывающие основные параметры кода, такие как: название кода, краткое описание кода, длину кода и длину информационного слова.

- *CodeThreads.pas*
  Содержит классы, реализующие кодер и декодер работающие в отдельном потоке с возможностью прерывания операции и ведения статистики.

- *BCH.pas*
  Данный модуль содержит реализацию БЧХ-кода с фиксированными параметрами $m = 4,\ t = 3$.

- *RM.pas*
  Содержит реализацию кода Рида-Маллера с произвольными параметрами.

- *FieldItem.pas и Polynom.pas*
  Эти модули содержат описание и реализацию элемента поля, арифметику многочленов и конечных полей.

- *BCHEx.pas*
  В данном модуле находится реализация двоичного кода БЧХ с произвольными параметрами m и t.

- *MainFrm.pas, CodecFrm.pas* и т.д. Эти модули описывают пользовательский интерфейс программы.

4.2 Руководство пользователя

В главном окне программы Вы можете выбрать интересующий вас код. Все функции, предоставляемые программой, будут использовать его в своей работе.

Здесь же отображается информация о параметрах выбранного кода. Если выбранный код поддерживает настройку своих параметров, то становится доступной кнопка "Настройка кода", которая вызывает диалог настройки параметров выбранного кода.

4.2.1 Кодирование и декодирование файлов

Для кодирования и декодирования файлов необходимо выбрать соответствующих пункт меню "*Файл*". После этого будут запрошены имя файла источника, и имя целевого файла, в который будет записан результат операции. Во время операции кодирования или декодирования на экране появляется окно, отображающее ход операции. В любой момент операцию можно прервать, закрыв это окно.

4.2.2 Демонстрация исправления ошибок

Для просмотра демонстрации исправления ошибок выберите пункт "*Демонстрация*"меню "*Код*". Программа попросит Вас указать файл с изображением в формате BMP или JPG. Затем на экране появится окно, содержащее бегунок, задающий уровень шумов (вероятность ошибки при передаче одного бита); а также три области с изображениями. В первой области отображается результат наложения шума на выбранное Вами изображение. Вторая область содержит выбранное изображение без каких-либо изменений. В третьей области отображается изображение, полученное в результате кодирования исходного изображения, наложения шума и последующего декодирования.

В результате, можно визуально оценить пользу от применения кодов, контролирующих ошибки, при передаче информации по каналам связи.

5 Заключение

В результате проделанной работы были реализованы:

1. гибкая, легко расширяемая система, основанная на принципах объектно-ориентированного программирования;
2. класс двоичного кода БЧХ длины $2^m - 1$, исправляющего t ошибок;
3. программа для демонстрации возможностей кодов, контролирующих ошибки.

Из-за стремления к универсальности и удобству использования, скорость работы невысока. Однако, имеются большие резервы для повышения производительности без ущерба универсальности и удобству. Дипломная работа имеет хорошие перспективы для дальнейшего развития. Первое что можно сделать, реализовать более эффективные алгоритмы декодирования (сверточные алгоритмы) и арифметику многочленов. Также можно использовать реализацию кодера и декодера для построения криптосистем с открытым ключом, основанных на кодах, контролирующих ошибки (система Мак-Элиса). И, наконец, можно использовать реализацию БЧХ-кода для исследования кодов БЧХ с большими длинами (рассмотреть зависимость информационной длины k от длины кода n и количества исправляемых ошибок t).

Дипломную работу можно использовать в учебных целях, для изучения основ теории алгебраических кодов, контролирующих ошибки.

6 Литература

1. *Блейхут Р. Теория и практика кодов, контролирующих ошибки.* М.: Мир, 1986.

2. *Лидл Р., Нидеррайтер Г. Конечные поля, т.1.* М.: Мир, 1988.