**Разработка архитектуры и реализация протокола передачи данных с исправлением данных.**

**Модель p a**

Основным понятием, положенным в основу данной модели является плотность ошибка порядка t. Это неслучайная функция от n и t , где - среднее число ошибок на блоке длинной n с t или большим количеством ошибок. Значение плотности порядка t подчиняется следующему условию . Значения функции v(t,n) не убывают с ростом t, то есть

, если выполняются условия больше p хотя бы в несколько раз. Параметр носит название показателя группирования ошибок, подчиняется условию . При получаем канал с независимыми ошибками, а при получаем канал с “жестким” пакетированием ошибок.

На практике применяют данное соотношение для задания модели (p, a). Параметр p – характеризует вероятность ошибки символа и рассчитывается по формуле ФОРМУЛА. Параметр а – находят из уравнения .

Достоинством применение данной модели для исследования каналов связи является то, что учитывается пакетирования ошибок, возможность единообразно описать разные типы каналов передачи данных. Так для кабельных каналов значение максимально (>0.5), а для радиоканалов применяют значение в промежутке от 0.3 до 0.45. У данного способа моделирования каналов связи есть недостаток, заключающийся в неполноте и вопрос модели на уровне блоков.

**Модель ОПП.**

Наблюдаемое пакетирование ошибок в каналах связи при предположении о пуассоновском характере потока можно объяснить, если считать параметр л не константой, а случайной величиной или процессом. Получающийся путем рандомизации л новый случайный процесс называют обобщенным пуассоновским . Будем считать л случайной величиной, закон распределения которой известен F(л). Тогда канал задается как поток ошибок первым способом:

Т.е формула для P(k,n) сохраняется, но осуществляется усреднение по параметру.

Поскольку вид и параметры закона распределения для реальных каналов обычно неизвестны, указанной выше формулой воспользоваться не удается.

Используя производящую функцию вероятностей:

и

Таким образом, для ОПП, зная функцию распределения интервалов между ошибками или P0(t) вычисляются вероятности P(k,t), таким образом получаем первый способ задание потока.

Таким образом, для ОПП, зная функцию распределения интервалов между ошибками или P0(t) вычисляются вероятности P(k,t) , т.е. приходим к заданию потока первым способом но конструктивным.

Рассмотрим один частный случай, когда распределение интервалов задается обобщенной гиперболой

**Коды Рида — Соломона**

**Коды Рида — Соломона** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Reed–Solomon codes*) — недвоичные [циклические коды](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4), позволяющие исправлять ошибки в блоках данных. Элементами кодового вектора являются не биты, а группы битов (блоки). Очень распространены коды Рида — Соломона, работающие с байтами (октетами).

Коды Рида — Соломона являются важным частным случаем [БЧХ-кода](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0), корни порождающего полинома которого лежат в том же [поле](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5), над которым строится код (m=1). Пусть \alpha — элемент [поля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) \textstyle GF(q) порядка \textstyle n. Если \alpha —*примитивный* элемент, то его порядок равен q-1, то есть \alpha^{q-1}=1,\quad \alpha^i \neq 1, 0<i<q-1. Тогда нормированный [полином](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD_%D0%BD%D0%B0%D0%B4_%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%BC_%D0%BF%D0%BE%D0%BB%D0%B5%D0%BC) g(x) минимальной степени над полем \textstyle GF(q), [корнями](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%B5%D0%BD%D1%8C_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD%D0%B0) которого являются d-1 подряд идущих степеней \alpha^{l_0}, \alpha^{l_0+1},...,\alpha^{l_0+d-2} элемента \alpha, является порождающим полиномом кода Рида — Соломона над полем \textstyle GF(q):

g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1})\dots(x - \alpha^{l_0+d-2})

где l_0 — некоторое целое число (в том числе 0 и 1), с помощью которого иногда удается упростить кодер. Обычно полагается l_0 = 1. Степень многочлена \textstyle g(x) равна d-1.

Длина полученного кода n, минимальное расстояние d (минимальное расстояние d линейного кода является минимальным из всех [расстояний Хемминга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3%D0%B0) всех пар кодовых слов, см. [Линейный код](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)). Код содержит r=d-1=\deg (g(x)) проверочных символов, где \deg() обозначает степень полинома; число информационных символов k = n - r = n - d + 1. Таким образом \textstyle d = n - k + 1 и код Рида — Соломона является *разделимым кодом с максимальным расстоянием* (является оптимальным в смысле [границы Синглтона](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0_%D0%A1%D0%B8%D0%BD%D0%B3%D0%BB%D1%82%D0%BE%D0%BD%D0%B0)).

Кодовый полином c(x) может быть получен из информационного полинома m(x), \deg m(x) \leqslant k-1, путем перемножения m(x) и g(x):

c(x) = m(x)g(x)

Код Рида — Соломона над \textstyle GF(q^m), исправляющий t ошибок, требует 2t проверочных символов и с его помощью исправляются произвольные пакеты ошибок длиной t и меньше. Согласно теореме о границе Рейгера, коды Рида — Соломона являются оптимальными с точки зрения соотношения длины пакета и возможности исправления ошибок — используя 2t дополнительных проверочных символов исправляется t ошибок (и менее).

**Теорема (граница Рейгера)**. Каждый линейный блоковый код, исправляющий все пакеты длиной t и менее, должен содержать, по меньшей мере, 2t проверочных символов.

Код, двойственный коду Рида — Соломона, есть также код Рида-Соломона. Двойственным кодом для [циклического кода](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) называется код, порожденный его проверочным многочленом.

Матрица G=[
\begin{array}{cc}
I_{k*k} & P_{k*(n-k)} \\ 
\end{array}] порождает код Рида — Соломона тогда и только тогда когда любой [минор](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%BE%D1%80_(%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0)" \o "Минор (линейная алгебра))матрицы P_{k*(n-k)} отличен от нуля.

При **выкалывании** или **укорочении** кода Рида-Соломона снова получается код Рида — Соломона. **Выкалывание**— операция, состоящая в удалении одного проверочного символа. Длина n кода уменьшается на единицу, размерность k сохраняется. Расстояние кода d должно уменьшиться на единицу, ибо в противном случае удаленный символ был бы бесполезен.**Укорочение** - фиксируем произвольный столбец (n,k,d) кода и выбираем только те векторы, которые в данном столбце содержат 0. Это множество векторов образует [подпространство](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE#.D0.9C.D0.B0.D1.82.D0.B5.D0.BC.D0.B0.D1.82.D0.B8.D0.BA.D0.B0).

### Исправление многократных ошибок

Код Рида — Соломона является одним из наиболее мощных кодов, исправляющих многократные пакеты ошибок. Применяется в каналах, где пакеты ошибок могут образовываться столь часто, что их уже нельзя исправлять с помощью кодов, исправляющих одиночные ошибки.

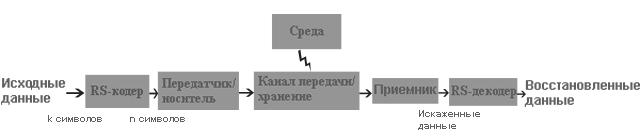
(q^m - 1,  q^m -2 - 2t)-код Рида — Соломона над полем \textstyle GF(q^m) с кодовым расстоянием d = 2t + 1 можно рассматривать как ((q^m - 1)m,(q^m -1 - 2t)m)-код над полем \textstyle GF(q), который может исправлять любую комбинацию ошибок, сосредоточенную в t или меньшем числе блоков из m символов. Наибольшее число блоков длины m, которые может затронуть пакет длины l_i, где l_i \leqslant mt_i - (m-1), не превосходит t_i, поэтому код, который может исправить t блоков ошибок, всегда может исправить и любую комбинацию из p пакетов общей длины l, если l+(m-1) \leqslant mt.

Кодирование с помощью кода Рида — Соломона может быть реализовано двумя способами: систематическим и несистематическим (см. [[1]](http://www.insidepro.com/kk/027/027r.shtml), описание кодировщика).

При несистематическом кодировании информационное слово умножается на некий неприводимый полином в поле Галуа. Полученное закодированное слово полностью отличается от исходного и для извлечения информационного слова нужно выполнить операцию декодирования и уже потом можно проверить данные на содержание ошибок. Такое кодирование требует большие затраты ресурсов только на извлечение информационных данных, при этом они могут быть без ошибок.

Структура систематического кодового слова Рида — Соломона

При систематическом кодировании к информационному блоку из k символов приписываются 2t проверочных символов, при вычислении каждого проверочного символа используются все k символов исходного блока. В этом случае нет затрат ресурсов при извлечении исходного блока, если информационное слово не содержит ошибок, но кодировщик/декодировщик должен выполнить k(n - k) операций сложения и умножения для генерации проверочных символов. Кроме того, так как все операции проводятся в поле Галуа, то сами операции кодирования/декодирования требуют много ресурсов и времени. Быстрый алгоритм декодирования, основанный на быстром преобразовании Фурье, выполняется за время порядка  O({ln( {n}) }^2).

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D0%B4%D0%B0_%D0%A0%D0%B8%D0%B4%D0%B0-%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0.gif)

### Кодирование

При операции кодирования информационный полином умножается на порождающий многочлен. Умножение исходного слова S длины k на неприводимый полином при систематическом кодировании можно выполнить следующим образом:

* К исходному слову приписываются 2t нулей, получается полином \textstyle T = S x^{2t}.
* Этот полином делится на порождающий полином G, находится остаток R, \textstyle S x^{2t} = Q G + R, где Q — частное.
* Этот остаток и будет корректирующим кодом Рида — Соломона, он приписывается к исходному блоку символов. Полученное кодовое слово \textstyle C = S x^{2t} + R.

Кодировщик строится из сдвиговых регистров, сумматоров и умножителей. Сдвиговый регистр состоит из ячеек памяти, в каждой из которых находится один элемент поля Галуа.

Существует и другая **процедура** кодирования (более практичная и простая). Положим  a_{i} \in GF(q) , (i = 1,2,\ldots,k-1) , \alpha \in GF(q) - примитивный элемент поля GF(q), и пусть 
a = (a_0,a_1,\ldots,a_{k-1})
 - вектор информационных символов , а значит a(x) = a_0 + a_{1}x + \ldots + a_{k-1}x^{k-1} - информационный многочлен. Тогда вектор u = (a(1),a(\alpha),\ldots,a(\alpha^{q-2})) есть вектор кода Рида - Соломона , соответствующий информационному вектору a. Этот способ кодирования показывает , что для кода РС вообще не нужно знать порождающего многочлена и порождающей матрицы коды, достаточно знать разложение [поля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) GF(q)по примитивному элементу \alpha и размерность кода k (длина кода в этом случае определяется как n = q - 1). Все дело в том , что за разностью n-k полностью скрывается порождающий многочлен g(x) и кодовое расстояние.

### Декодирование

Декодировщик, работающий по авторегрессивному спектральному методу декодирования, последовательно выполняет следующие действия:

* Вычисляет синдром ошибки
* Строит полином ошибки
* Находит корни данного полинома
* Определяет характер ошибки
* Исправляет ошибки

#### Вычисление синдрома ошибки

Вычисление синдрома ошибки выполняется синдромным декодером, который делит кодовое слово на порождающий многочлен. Если при делении возникает остаток, то в слове есть ошибка. Остаток от деления является синдромом ошибки.

#### Построение полинома ошибки

Вычисленный синдром ошибки не указывает на положение ошибок. Степень полинома синдрома равна 2t, что много меньше степени кодового слова n. Для получения соответствия между ошибкой и ее положением в сообщении строится полином ошибок. Полином ошибок реализуется с помощью [алгоритма Берлекэмпа — Месси](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%B5%D1%80%D0%BB%D0%B5%D0%BA%D1%8D%D0%BC%D0%BF%D0%B0_%E2%80%94_%D0%9C%D1%8D%D1%81%D1%81%D0%B8), либо с помощью алгоритма Евклида. Алгоритм Евклида имеет простую реализацию, но требует больших затрат ресурсов. Поэтому чаще применяется более сложный, но менее затратоемкий алгоритм Берлекэмпа — Месси. Коэффициенты найденного полинома непосредственно соответствуют коэффициентам ошибочных символов в кодовом слове.

#### Нахождение корней

На этом этапе ищутся корни полинома ошибки, определяющие положение искаженных символов в кодовом слове. Реализуется с помощью процедуры Ченя, равносильной полному перебору. В полином ошибок последовательно подставляются все возможные значения, когда полином обращается в ноль — корни найдены.

#### Определение характера ошибки и ее исправление

По синдрому ошибки и найденным корням полинома с помощью алгоритма Форни определяется характер ошибки и строится маска искаженных символов. Однако для кодов РС существует более простой способ отыскания характера ошибок. Как показано в [[2]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A0%D0%B8%D0%B4%D0%B0_%E2%80%94_%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0#cite_note-pro-2) для кодов РС с произвольным множеством 2t_d последовательных нулей \alpha^b,\alpha^{b+1},\ldots,\alpha^{b+\delta},\delta = 2t_d - 1

 e_{j_{l}} = \frac{(\alpha^{j_{l}})^{2-b} \Lambda(\alpha^{-j_{l}}) }{\sigma'(\alpha^{-j_{l}})}  \quad \quad \quad(*) 

где \sigma'(x) формальная производная по x многочлена локаторов ошибок \sigma(x) ,а \Lambda(x) = \sigma(x) S(x)\mod x^{2t_d + 1}

Далее после того как маска найдена , она накладывается на кодовое слово с помощью операции [XOR](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) и искаженные символы восстанавливаются. После этого отбрасываются проверочные символы и получается восстановленное информационное слово.

Этот алгоритм кодирования используется при передаче данных по сетям [WiMAX](https://ru.wikipedia.org/wiki/WiMAX" \o "WiMAX), в [оптических линиях связи](https://ru.wikipedia.org/wiki/%D0%92%D0%9E%D0%9B%D0%A1), в[спутниковой](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D1%83%D1%82%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C) и [радиорелейной связи](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D1%80%D0%B5%D0%BB%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C). Метод прямой коррекции ошибок в проходящем трафике (Forward Error Correction, FEC) основывается на кодах Рида — Соломона.

**Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды)** — в [теории кодирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) это широкий класс [циклических кодов](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4), применяемых для защиты информации от ошибок (см. [Обнаружение и исправление ошибок](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BD%D0%B0%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%B8%D1%81%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA)). Отличается возможностью построения кода с заранее определёнными корректирующими свойствами, а именно, минимальным кодовым расстоянием. Частным случаем БЧХ-кодов является [код Рида — Соломона](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A0%D0%B8%D0%B4%D0%B0_%E2%80%94_%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0).

БЧХ-код является [циклическим кодом](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4), который можно задать [порождающим полиномом](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4#.D0.9F.D0.BE.D1.80.D0.BE.D0.B6.D0.B4.D0.B0.D1.8E.D1.89.D0.B8.D0.B9_.D0.BF.D0.BE.D0.BB.D0.B8.D0.BD.D0.BE.D0.BC). Для его нахождения в случае БЧХ-кода необходимо заранее определить длину кода n (она не может быть произвольной) и требуемое минимальное расстояние d \leqslant n. Найти порождающий полином можно следующим образом.

Пусть ~\alpha — [примитивный элемент](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8F) [поля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) ~GF(q^m) (то есть \alpha^{q^m-1}=1, \alpha^i \neq 1, i< q^m-1), пусть ~\beta=\alpha^s , — элемент поля ~GF(q^m) порядка ~n, \quad s = (q^m-1) / n . Тогда нормированный [полином](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD_%D0%BD%D0%B0%D0%B4_%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%BC_%D0%BF%D0%BE%D0%BB%D0%B5%D0%BC) g(x) минимальной степени над полем GF(q), [корнями](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%B5%D0%BD%D1%8C_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD%D0%B0) которого являются ~d-1 подряд идущих степеней ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2}элемента ~\beta для некоторого целого ~l_0 (в том числе 0 и 1), является порождающим полиномом БЧХ-кода над полем ~GF(q) с длиной n и минимальный расстоянием ~d_0 \geqslant d. Поясним почему у получившегося кода будут именно такие характеристики (длина кода ~n , минимальное расстояние ~d_0). Действительно, как показано в [[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#cite_note-saga-1) , длина БЧХ кода равна [порядку](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C_%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%BE%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D1%83%D0%BF%D0%BF#.D0.9F) элемента ~\beta, если ~d>2 и равна порядку элемента ~\beta^{l_0}, если ~d=2, тогда, так как случай ~d=2нам не интересен (такой код не может исправлять ошибки, только обнаруживать), то длина кода будет равна порядку элемента ~\beta ,то есть равна ~n. Минимальное расстояние ~d_0 может быть больше ~d, когда корнями минимальных функций(стр.83[[2]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#cite_note-2)) от элементов ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} будут элементы расширяющие последовательность, то есть элементы ~\beta^{l_0+d-1},\beta^{l_0+d},\ldots,\beta^{l_0+d_0 - 2}.

Число проверочных символов r равно степени g(x), число информационных символов k=n-r, величина dназывается *конструктивным расстоянием* БЧХ-кода. Если n=q^m-1, то код называется *примитивным*, иначе*непримитивным*.

Так же, как и для циклического кода, кодовый полином c(x) может быть получен из информационного полинома m(x), степени не больше k-1, путём перемножения m(x) и g(x):

c(x)=m(x)g(x).

## **Построение**

Для нахождения порождающего полинома необходимо выполнить несколько этапов:

* выбрать q, то есть поле GF(q), над которым будет построен код;
* выбрать длину n кода из условия n=(q^m-1)/s, где m,s — целые положительные числа;
* задать величину d конструктивного расстояния;

1) построить циклотомические классы элемента \beta=\alpha^s поля GF(q^m) над полем GF(q), где \alpha — примитивный элемент GF(q^m);

2) поскольку каждому такому циклотомическому классу соответствует неприводимый полином над GF(q), корнями которого являются элементы этого и только этого класса, со степенью равной количеству элементов в классе, то выбрать \beta^{l_0}, \beta^{l_0+1},\ldots, \beta^{l_0+d-2} таким образом, чтобы суммарная длина циклотомических классов была минимальна; это делается для того, чтобы при заданных характеристиках кода ~n и ~d минимизировать количество проверочных символов ~k;

3) вычислить порождающий полином g(x)=f_1(x)f_2(x)\ldots f_h(x), где f_i(x) — полином, соответствующий i-ому циклотомическому классу; или вычислить g(x), как [НОК](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B5%D0%B5_%D0%BE%D0%B1%D1%89%D0%B5%D0%B5_%D0%BA%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5) минимальных функций от элементов \beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} (стр.168[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#cite_note-saga-1)).

## **Кодирование**

Для кодирования кодами БЧХ применяются те же методы, что и для [кодирования циклическими кодами.](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4#.D0.9A.D0.BE.D0.B4.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5)

## **Методы декодирования**

Коды БЧХ являются циклическими кодами, поэтому к ним применимы все методы, используемые для декодирования циклических кодов. Однако существуют гораздо лучшие [алгоритмы](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), разработанные именно для БЧХ-кодов(стр.91[[3]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#cite_note-.D0.9C.D0.BE.D1.80.D0.B5.D0.BB.D0.BE.D1.81-3)).

Главной идеей в декодировании БЧХ кодов является использование элементов [конечного поля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) для нумерации позиций кодового слова(или, эквивалентно, в порядке коэффициентов ассоциированного многочлена). Ниже приведена такая нумерация для вектора ~r = (r_0,r_1,\ldots,r_{n-1}), соответствующего многочлену ~r(x).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| значения | r_0 | r_1 | \ldots | r_{n-1} |
| локаторы позиций | 1 | \alpha | \ldots | \alpha^{n-1} |

Пусть принятое слово ассоциировано с полиномом r(x)=\nu(x) + e(x), где многочлен ошибок определён как: e(x) = e_{J_1}x^{J_1}+e_{J_2}x^{J_2}+\ldots+e_{J_\nu}x^{J_\nu}, где \nu \le t_d число ошибок в принятом слове. Множества 
{e_{J_1},e_{J_2},\ldots,e_{J_\nu}} и {\alpha^{J_1},\alpha^{J_2},\ldots,\alpha^{J_\nu}} называют *значениями ошибок* и *локаторами ошибок* соответственно, где e_J \in GF(q) и \alpha \in GF(q^m).

*Синдромы* определены как значения принятого полинома r(x) в нулях порождающего многочлена кода:

 S_1 = r(\alpha^b) = e_{J_1}\alpha^{bJ_1}+e_{J_2}\alpha^{bJ_2}+\ldots+e_{J_\nu}\alpha^{bJ_\nu}

 S_2 = r(\alpha^{b+1}) = e_{J_1}\alpha^{(b+1)J_1}+e_{J_2}\alpha^{(b+1)J_2}+\ldots+e_{J_\nu}\alpha^{(b+1)J_\nu}

 \vdots 

 S_{2t_d} = r(\alpha^{b+2t_d-1}) = e_{J_1}\alpha^{(b+2t_d-1)J_1}+e_{J_2}\alpha^{(b+2t_d-1)J_2}+\ldots+e_{J_\nu}\alpha^{(b+2t_d-1)J_\nu}   

*Здесь* ~2*t_d = d - 1 Для нахождения множества локаторов ошибок, введем в рассмотрение *многочлен локаторов ошибок*

\sigma(x) = \prod_{l=1}^\nu (1 + \alpha^{J_l}x) = 1 + \sigma_{1}x + \sigma_{2}x^2 + \ldots + \sigma_{\nu}x^{\nu} 

корни которого равны обратным величинам локаторов ошибок. Тогда справедливо следующее соотношение между коэффициентами многочлена локаторов ошибок и синдромами(см. например[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#cite_note-saga-1), стр.200):

 \sigma_{1}S_{t}+\sigma_{2}S_{t-1}+\ldots+\sigma_{t}S_{1} = -S_{t+1} 

 \sigma_{1}S_{t+1}+\sigma_{2}S_{t}+\ldots+\sigma_{t}S_{2} = -S_{t+2} 

 \ldots  \quad \quad \quad \quad\quad \quad \quad \quad\quad \quad \quad \quad\quad \quad \quad \quad \quad \quad \quad(*)

 \sigma_{1}S_{2t-1}+\sigma_{2}S_{2t-2}+\ldots+\sigma_{t}S_{t} = -S_{2t} 

Известны следующие методы для решения этой системы уравнений относительно коэффициентов *многочлена локаторов ошибок* \sigma_{i},i=1,2,\ldots,\nu(**ключевая система уравнений**).

* **Алгоритм Берлекемпа-Мэсси** (BMA)[[⇨]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC_.D0.91.D0.B5.D1.80.D0.BB.D0.B5.D0.BA.D0.B5.D0.BC.D0.BF.D0.B0-.D0.9C.D1.8D.D1.81.D1.81.D0.B8). По числу операций в [конечном поле](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) этот алгоритм обладает высокой эффективностью. BMA обычно используется для программной реализации или моделирования кодов БЧХ и[кодов Рида-Соломона](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A0%D0%B8%D0%B4%D0%B0_%E2%80%94_%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0).
* **Евклидов алгоритм** (ЕА)[[⇨]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%91%D0%BE%D1%83%D0%B7%D0%B0_%E2%80%94_%D0%A7%D0%BE%D1%83%D0%B4%D1%85%D1%83%D1%80%D0%B8_%E2%80%94_%D0%A5%D0%BE%D0%BA%D0%B2%D0%B8%D0%BD%D0%B3%D0%B5%D0%BC%D0%B0#.D0.95.D0.B2.D0.BA.D0.BB.D0.B8.D0.B4.D0.BE.D0.B2_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC). Из-за высокой регулярности структуры этого алгоритма его широко используют для аппаратной реализации декодеров БЧХ и [кодов Рида-Соломона](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A0%D0%B8%D0%B4%D0%B0_%E2%80%94_%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0).
* **Прямое решение** (алгоритм Питерсона — Горенстейна — Цирлера, ПГЦ). Исторически это первый метод декодирования, найденный Питерсоном для двоичного случая (q=2), затем Горенстейном и Цирлером для общего случая. Этот алгоритм находит коэффициенты *многочлена локаторов ошибок* прямым решением соответствующей системы линейных уравнений. В действительности, так как сложность этого алгоритма растет как куб минимального расcтояния ~d, прямой алгоритм может быть использован только для малых значений ~d, однако именно этот алгоритм лучше всего проясняет алгебраическую идею процесса декодирования.

### Алгоритм Берлекемпа-Мэсси

Этот алгоритм лучше всего рассматривать как итеративный процесс построения минимального регистра(сдвига) с обратной связью, генерирующего известную последовательность синдромов S_1,S_2,\ldots,S_{2t_d}. Его фактическая цель — построить полином \sigma^{i+1}(x) наименьшей степени, удовлетворяющему следующему уравнению \sum_{j=0}^{l_{i}+1} S_{k-j} \sigma_{j}^{i+1} = 0, l_{i}<k<i+1.Решение этого уравнения эквивалентно следующему условию \sigma^{i+1}(x) = 1 + \sigma_{1}^{i+1} x + \ldots + \sigma_{l_{i+1}}^{i+1} x^{l_{i+1}} . Итеративный процесс построения такого многочлена и есть*Алгоритм Берлекемпа-Мэсси*.

### Евклидов алгоритм

В основе этого метода лежит широко известный [алгоритм Евклида](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%B0) по нахождению наибольшего общего делителя двух чисел (НОД), только в данном случае ищем НОК не двух чисел, а двух полиномов. Обозначим *полином значений ошибок* как \Lambda = \sigma(x) S(x), где синдромный полином равен S(x) = 1 + S_1 x + \ldots + S_{2*t_d} x^{2t_d}\quad (1). Из системы уравнений (\*) следует что  \Lambda(x) = \sigma(x) S(x) \mod x^{2t_d + 1} \quad (2). Задача по сути сводится к тому чтобы определить \Lambda(x)удовлетворяющего (2) и при этом степени не выше t_{d}. По сути такое решение и будет давать расширенный алгоритм Евклида, примененный к многочленам r_0(x) = x^d  и r_1(x)=S(x) , где d = 2t_d + 1. Если на j-ом шаге расширенный алгоритм Евклида выдает решение r_{j} = a_{j}(x) x^{2t_d + 1} + b_{j}(x) S(x), такое что  \deg[r_{j}(x)] \le t_d , то \Lambda(x) = r_j(x)  и \sigma_i(x) = b_j(x). При этом найденный полином a_j(x) дальше не принимает участия в декодировании(он ищется только как вспомогательный). Таким образом будет найден полином локаторов ошибок \sigma(x).

### Прямое решение (алгоритм Питерсона — Горенстейна — Цирлера, ПГЦ)

Пусть БЧХ код над полем GF(q) длины n и с конструктивным расстоянием d задается порождающим полиномом g(x), который имеет среди своих корней элементы \beta^{l_0},\beta^{l_0+1},\ldots,\beta^{l_0+d-2} \quad \beta \in GF(q^m), \quad \beta^n=1, \quad l_0 — целое число (например 0 или 1). Тогда каждое кодовое слово обладает тем свойством, что c(\beta^{l_0-1+j}) = 0, \quad j=1,2,\ldots,d-1. Принятое слово r(x) можно записать как r(x)=c(x)+e(x), где e(x) — полином ошибок. Пусть произошло u \leqslant t = (d-1)/2 ошибок на позициях i_1,i_2,\ldots,i_u (tмаксимальное число исправляемых ошибок), значит e(x) = e_{i_1}x^{i_1}+e_{i_2}x^{i_2}+\ldots+e_{i_u}x^{i_u}, а e_{i_1}, e_{i_2},\ldots, e_{i_u} — величины ошибок.

Можно составить j-ый *синдром* S_j принятого слова r(x):

S_j = r(\beta^{l_0-1+j}) = e(\beta^{l_0-1+j}), \quad j=1,\ldots,d-1\quad\quad (1).

Задача состоит в нахождений числа ошибок u, их позиций i_1,i_2,\ldots,i_u и их значений e_{i_1}, e_{i_2},\ldots, e_{i_u} при известных синдромах S_j.

Предположим, для начала, что u в точности равно t. Запишем (1) в виде [системы нелинейных(!) уравнений](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BD%D0%B5%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9&action=edit&redlink=1) в явном виде:


{ \begin{cases}
S_1 = e_{i_1} \beta^{l_0 i_1} + e_{i_2} \beta^{l_0 i_2} + \dots + e_{i_t} \beta^{l_0 i_t} \\
S_2 = e_{i_1} \beta^{(l_0+1) i_1} + e_{i_2} \beta^{(l_0+1) i_2} + \dots + e_{i_t} \beta^{(l_0+1) i_t} \\
\cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
S_{d-1} = e_{i_1} \beta^{(l_0+d-2) i_1} + e_{i_2} \beta^{(l_0+d-2) i_2} + \dots + e_{i_t} \beta^{(l_0+d-2) i_t} \\
\end{cases} }


Обозначим через X_k = \beta^{i_k} *локатор* k-ой ошибки, а через Y_k = e_{i_k} *величину* ошибки, k=1,\ldots,t. При этом всеX_k различны, так как порядок элемента \beta равен n, и поэтому при известном X_k можно определить i_k как i_k = \log_{\beta} X_k .


{ \begin{cases}
S_1 = Y_1 X_1^{l_0} + Y_2 X_2^{l_0} + \dots + + Y_t X_t^{l_0} \\
S_2 = Y_1 X_1^{l_0+1} + Y_2 X_2^{l_0+1} + \dots + + Y_t X_t^{l_0+1} \quad \quad \quad \quad \quad\quad(2) \\
\cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
S_{d-1} = Y_1 X_1^{l_0+d-2} + Y_2 X_2^{l_0+d-2} + \dots + + Y_t X_t^{l_0+d-2} 
\end{cases} }


Составим *полином локаторов ошибок*:

\Lambda (x) = (1-xX_1)(1-xX_2)\dots (1-xX_t) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + \dots + \Lambda_1 x + 1

Корнями этого полинома являются элементы, обратные локаторам ошибок. Помножим обе части этого полинома наY_l X_{l}^{\vartheta+t}. Полученное равенство будет справедливо для \vartheta = l_0,l_0+1,\dots,l_0+d-1,\quad l=1,\ldots,t:

\Lambda (x) Y_l X_{l}^{\vartheta+t} = \Lambda_t x^t Y_l X_{l}^{\vartheta+t} + \Lambda_{t-1} x^{t-1} Y_l X_{l}^{\vartheta+t} + \dots + \Lambda_1 x Y_l X_{l}^{\vartheta+t} + Y_l X_{l}^{\vartheta+t}  \quad (3)

Положим x=X_l^{-1} и подставим в (3). Получится равенство, справедливое для каждого l \in {1,2,...,t} и при всех \vartheta \in { l_0,l_0+1,\dots,l_0+d-1 }:

0 = \Lambda_t Y_l X_{l}^{\vartheta} + \Lambda_{t-1} Y_l X_{l}^{\vartheta+1} + \dots + \Lambda_{1} Y_l X_{l}^{\vartheta+t-1} + Y_l X_{l}^{\vartheta+t}

Таким образом для каждого l можно записать свое равенство. Если их просуммировать по l, то получится равенство, справедливое для каждого \vartheta \in { l_0,l_0+1,\dots,l_0+d-1 }:

0 = \Lambda_t \sum_{l=1}^t Y_l X_{l}^{\vartheta} + \Lambda_{t-1} \sum_{l=1}^t Y_l X_{l}^{\vartheta+1} + \dots + \Lambda_{1} \sum_{l=1}^t Y_l X_{l}^{\vartheta+t-1} + \sum_{l=1}^t Y_l X_{l}^{\vartheta+t}.

.

Учитывая (2) и то, что S_{j+p} = \sum_{l=1}^t Y_l X_{l}^{l_0+j+p-1} = \sum_{l=1}^t Y_l X_{l}^{\vartheta+p}, \quad j=1,\ldots,d-1, \quad \vartheta = l_0+j-1, (то есть \vartheta меняется в тех же пределах, что и ранее) получаем [систему линейных уравнений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9):


{ \begin{cases}
S_1 \Lambda_t + S_2 \Lambda_{t-1} + \dots + S_t \Lambda_1 = -S_{t+1} \\
S_2 \Lambda_t + S_3 \Lambda_{t-1} + \dots + S_{t+1} \Lambda_1 = -S_{t+2}   \quad \quad \quad \quad \quad\quad(4) \\
\cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
S_t \Lambda_t + S_{t+1} \Lambda_{t-1} + \dots + S_{2t-1} \Lambda_1 = -S_{2t}
\end{cases} }


.

Или в матричной форме


S^{(t)} \bar\Lambda^{(t)} = -\bar s^{(t)}


,

где


S^{(t)}={ \left[ \begin{matrix}
S_1 & S_2 & \dots & S_t \\
S_2 & S_3 & \dots & S_{t+1} \\
\cdots & \cdots & \cdots &  \\
S_t & S_{t+1} & \dots & S_{2t-1} 
\end{matrix} \right] },  \quad \quad \quad \quad \quad\quad(5)



\bar\Lambda^{(t)} = 
{ \left[ \begin{matrix}
\Lambda_t  \\
\Lambda_{t-1}  \\
\cdots  \\
\Lambda_1
\end{matrix} \right] },  

\quad \quad 
\bar s^{(t)}
{ \left[ \begin{matrix}
S_{t+1}  \\
S_{t+2} \\
\cdots  \\
S_{2t}
\end{matrix} \right] }



Если число ошибок и в самом деле равно t, то система (4) разрешима, и можно найти значения коэффициентов \Lambda_{1},\ldots,\Lambda_{t}. Если же число u < t, то [определитель](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C) матрицы S^{(t)} системы (4) будет равен 0. Это есть признак того, что количество ошибок меньше t. Поэтому необходимо составить систему (4), предполагая число ошибок равным t-1. Высчитать определитель новой матрицы S^{(t-1)} и т. д., до тех пор, пока не установим истинное число ошибок.

### Поиск Ченя

После того как **ключевая система уравнений** (*) решена, получаются коэффициенты полинома локаторов ошибок. Его корни (элементы, обратные локаторам ошибок) можно найти простым перебором по всем элементам поля GF(q^m). К ним найти элементы, обратные по умножению, — это локаторы ошибок X_k, \quad k=1,\ldots,u. Этот процесс легко реализовать аппаратно.

### Алгоритм Форни

Общая схема декодирования БХЧ кодов (алгоритм Форни)

По локаторам можно найти позиции ошибок (i_k=\log_{\beta}X_k), а значения Y_k ошибок из системы (2), приняв t=u. Декодирование завершено.