**Анализ методов передачи данных.**

**Исследование существующих методов исправления ошибок**

**Коды Рида — Соломона**  — 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).

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)

### 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.

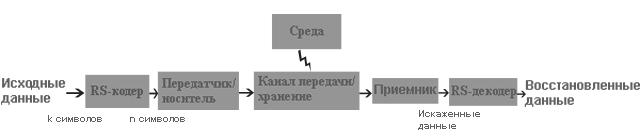
(q^m - 1,  q^m -2 - 2t)-код Рида — Соломона над полем \textstyle GF(q^m) с кодовым расстоянием d = 2t + 1 можно рассматривать как ((q^m - 1)m,(q^m -1 - 2t)m)-код над полем \textstyle GF(q), который может исправлять любую комбинацию ошибок, сосредоточенную в t или меньшем числе блоков из m символов. Наибольшее 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

**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. Отличается возможностью построения кода с заранее определёнными корректирующими свойствами, а именно, минимальным кодовым расстоянием.

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. 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.

**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.

**Анализ результатов.**