Корректирующие коды и их свойства

Корректирующими свойствами обладают только избыточные коды, для которых справедливо неравенство

$m^n > K$,

где m — число позиций, n — число разрядов кода (длина кодовой комбинации), K — объём алфавита источника.

Идея обнаружения ошибок в принятой кодовой комбинации $\mathbf{b}'_k$ состоит в том, что для передачи информации от источника используются не все $N = m^n$ возможных кодовых комбинаций, а лишь некоторая часть из них:

$K < m^n = N$.

Эти K используемых комбинаций называются *разрешёнными*, а оставшиеся $N - K$ комбинаций — *запрещёнными*. Если в результате воздействия помех передаваемая разрешённая комбинация превращается в одну из запрещённых, то тем самым и обнаруживается факт ошибки (или ошибок). Если же совокупность ошибок окажется такой, что переданная разрешённая комбинация превратится в одну из других разрешённых, то такие ошибки не будут обнаружены. Как только ошибка обнаружена, по обратному каналу на передающую сторону оправляется *запрос на повторную передачу* комбинации, в которой обнаружена ошибка. Если же обратный канал в системе не предусмотрен, то информация просто теряется. Как следствие, в системах без обратного канала используют коды, которые позволяют не только обнаружить, но и исправить ошибку.

В системах с исправлением ошибок принятые кодовые комбинации декодируются в соответствии с выбранным алгоритмом. Рассмотрим простую геометрическую интерпретацию декодирования с исправлением ошибок.

Всё множество возможных кодовых комбинаций N разбивается на K непересекающихся подмножеств, приписываемых отдельным решениям (то есть отдельным символам источника). Разбиение выполняется так, чтобы при декодировании запрещённых комбинаций восстановить ту из разрешённых, которая могла быть передана с наибольшей вероятностью. Если кодовую комбинацию длиной n рассматривать как вектор в n-мерном пространстве, то это будет та из разрешённых комбинаций, которая находится ближе всего к принятой запрещённой:

$$\hat{\mathbf{b}}_j = \arg\min_i \left\| \mathbf{b}' - \mathbf{b}_i \right\|,$$

где $\hat{\mathbf{b}}_j$ — решение декодера о переданной комбинации, $\mathbf{b}'$ — принятая комбинация, $\mathbf{b}_i$ — все возможные разрешённые кодовые комбинации, $\|\cdot\|$ — норма (длина) разностного вектора (расстояние).

В двоичном случае $(m=2)$ пространство двоичных векторов длиной n называют *пространством Хемминга*. Длина вектора в пространстве Хемминга равна числу единиц в этом векторе, а расстояние между двумя векторами равно числу разрядов, в которых эти вектора различаются:

$$\|\mathbf{b}\| = \sqrt{\sum_l b_l^2} = \sqrt{\sum_l b_l};$$
$$\|\mathbf{b}' - \mathbf{b}\| = \sqrt{\sum_l \left(b_l' \oplus b_l\right)^2} = \sqrt{\sum_l \left(b_l' \oplus b_l\right)}.$$

Обычно система работает или в режиме обнаружения, или в режиме исправления ошибок, однако, возможны и системы, в которых некоторые запрещённые комбинации (например, близкие по расстоянию к разрешённой) сначала декодируются, а другие (например те, для которых отличие по расстоянию велико) перезапрашиваются.

Исправляющая и обнаруживающая способности кодов

Для корректирующих кодов с исправлением и обнаружением ошибок вводится понятие *обнаруживающей* и *исправляющей* способности кода.

Количество ошибок в кодовой комбинации, которое позволяет обнаружить данный код, называется *обнаруживающей способностью* кода. *Исправляющей способностью* называется количество ошибок, которое может исправить данный код. Обнаруживающая и исправляющая способности кодов определяются величиной *кодового расстояния*.

Кодовым расстоянием $d_{\min}$ называется минимальное расстояние между двумя разрешёнными комбинациями данного кода.

Блочный код с кодовым расстоянием $d_{\min}$ может гарантированно обнаружить

$$q_{\text{o}} = d_{\min} - 1$$

ошибок. Действительно, поскольку между двумя разрешёнными кодовыми комбинациями расстояние не меньше $d_{\min}$, то любая комбинация с числом ошибок меньше $d_{\min}$ попадает в число запрещённых, что позволяет обнаружить ошибку. Поскольку в принципе из-за большого числа ошибок передаваемая кодовая комбинация может превратиться в запрещённую, расстояние до которой будет больше $d_{\min}$ и, кроме того, расстояние между некоторыми кодовыми комбинациями также могут быть больше $d_{\min}$, код может обнаружить и большее число ошибок. Однако, вероятности появления таких ошибок пренебрежимо малы и поэтому они обычно не учитываются.

Аналогичным образом, можно сказать, что исправляющая способность кода с заданным $d_{\min}$

$$q_{\text{и}} < \frac{d_{\min}}{2},$$

поскольку все комбинации с числом ошибок меньше $\frac{d_{\min}}{2}$ окажутся внутри области разбиения соответствующей переданной комбинации и, следовательно, будут исправлены. Для нечётных $d_{\min}$ исправляющая способность

$$q_{\text{и}} = \frac{d_{\min} - 1}{2},$$

а для чётных

$$q_{\text{и}} = \frac{d_{\min}}{2} - 1.$$

В симметричном двоичном канале вероятность ошибки кратности q в комбинации длины n

$$p_q = \mathrm{C}_n^q p_0^q \left(1 - p_0\right)^{n-q},$$

где p_0 — вероятность ошибки в элементе, быстро убывает с ростом q. Как следствие, обычно достаточно малую вероятность ошибки декодирования позволяют получить коды с исправляющей способностью $q_{\text{и}}$ порядка $1...3$ ошибок.

Виды корректирующих кодов

Основная теорема кодирования Шеннона говорит о том, что сколь угодно малой вероятности ошибки можно достичь, используя кодовые комбинации большой длины. Однако, с ростом n существенно возрастает сложность декодирования по описанному выше алгоритму минимума расстояния. Поэтому большой интерес представляют избыточные коды и методы кодирования, не требующие перебора всех возможных кодовых комбинаций, но вместе с тем позволяющие получить малую вероятность ошибки декодирования.

Такие избыточные коды найдены, а наибольшее распространение среди них получили *линейные* коды. Линейным двоичным кодом длины n называется такой код, для которого сумма (по модулю два) любых двух разрешённых комбинаций данного кода также является разрешённой комбинацией данного кода. Среди разрешённых комбинаций линейного кода обязательно должна присутствовать комбинация, состоящая из всех нулей, чтобы сумма по модулю два любой разрешённой комбинации с самой собой также давала разрешённую («нулевую») комбинацию.

Если первые k символов кодовой комбинации длины n являются *информационными*, а остальные $r = n - k$ — неинформационными (*избыточными* или *проверочными*), то такой код будет называться *систематическим*.

Линейные коды можно сформировать следующим образом. К k информационным символам кодовых комбинаций, соответствующих $K = m^k$ сообщениям, прибавляются r проверочных символов, которые являются линейными комбинациями информационных.

Для линейного двоичного кода расстояние $d_{\min}$ определяется минимальным весом (числом единиц) по всем кодовым комбинациям (кроме нулевой). Действительно, расстояние по Хеммингу между двумя кодовыми комбинациями равно числу единиц в сумме этих комбинаций по модулю 2. Но для линейного кода сама эта сумма также является разрешённой комбинацией.

Линейные блочные систематические коды обычно обозначают как (n,k).

Избыточность линейного двоичного кода можно определить формулой

$$\rho_к = 1 - \frac{\log_2 2^k}{n} = 1 - \frac{k}{n} = \frac{r}{n}.$$

Величина

$$R = 1 - \rho_к = 1 - \frac{r}{n} = \frac{r}{n}$$

называется *скоростью* кода.

Свобода выбора проверочных символов позволяет при заданных n и k построить большое число различных кодов. Естественно, код стараются построить так, чтобы он был в некотором смысле *оптимальным*. В теории передачи информации оптимальным считается код, который при той же длине n и избыточности обеспечивает минимум вероятности ошибки декодирования. Коды, вся избыточность которых расходуется на исправление ошибок заданной кратности, называются *совершенными*.

Примеры линейных блочных двоичных систематических кодов

1. *Код с одной проверкой на чётность* $(n, n-1)$

К $k = n - 1$ информационным разрядам добавляется один проверочный. Общее число комбинаций равно $N = 2^n$, из них $K = 2^{n-1} = \frac{N}{2}$ — разрешённые, а остальные $N - K = \frac{N}{2}$ — запрещённые. Проверочный символ получается из k информационных сложением разрядов по модулю 2:

$$b_{n-1} = \sum_{l=0}^{k-1} b_l .$$

Из формулы видно, что проверочный разряд равен нулю, если среди информационных символов чётное число единиц, и единице — если число единиц нечётное. Таким образом, в кодовой комбинации число единиц всегда чётное. Следовательно, если в канале произойдёт одна ошибка, то число единиц станет нечётным. Комбинации, содержащие нечётное число единиц, являются запрещёнными, их ровно половина. Появление на приёме запрещённой комбинации — признак ошибки. Если в канале произойдут одновременно две ошибки в одной кодовой комбинации, число единиц останется чётным, и такая ситуация окажется незамеченной декодером, так же, как и появление 4, 6, 8 и т. д. (то есть чётного числа) ошибок. Ошибки же нечётной кратности (1, 3, 5 и т. д.) будут обнаружены.

Вероятность того, что ошибка в кодовой комбинации не будет обнаружена, равна сумме вероятностей того, что произойдёт чётное число ошибок:

$$p_{\text{но}} = p_2 + p_4 + \ldots .$$

Кодовое расстояние кода с одной проверкой на чётность $d_{\min} = 2$, то есть этот код гарантированно обнаруживает 1 ошибку (а также все ошибки нечётной кратности), но не может исправлять ошибки.

2. *Коды Хемминга*

Коды Хемминга получаются по следующему правилу: $n = 2^r - 1$, где n — общее число разрядов, r — число проверочных разрядов и $n - r$ — число информационных разрядов. Кодовое расстояние кодов Хемминга $d_{\min} = 3$. Коды Хемминга являются совершенными: они исправляют одну и только одну ошибку. Рассмотрим подробно случай $r = 3$ — код Хемминга $(7, 4)$.

Три проверочных разряда получаются из информационных таким образом, чтобы соблюдалось условие их линейной независимости. Например:

$$b_4 = b_0 \oplus b_1 \oplus b_2,$$
$$b_5 = b_1 \oplus b_2 \oplus b_3,$$
$$b_6 = b_0 \oplus b_2 \oplus b_3.$$

Таким образом, каждый из 7 символов участвует хотя бы в одной проверке (символ b_2 участвует в трёх проверках). Эти равенства используются как проверочные при приёме кодовой комбинации. Если в одном из разрядов содержится ошибка, то те равенства, в которых он участвует, не соблюдаются. Так как равенств три, то возможны 8 вариантов ситуаций. В ситуации, когда все равенства выполняются, кодовая комбинация или принята безошибочно, или в канале произошли ошибки, приведшие к новой разрешённой кодовой комбинации, то есть ошибка в кодовой комбинации не обнаружена. В другом случае, в зависимости от того, какая из оставшихся семи ситуаций имеет место в конкретном опыте, декодер определяет, какой из семи элементов принят неверно. Например, если не выполняются все три проверки, то это указывает на b_2, а если не выполняется только одна проверка, например, b_4, то это указывает на то, что произошла ошибка в самом проверочном символе, в данном случае это b_4.

Таким образом, одиночные ошибки не только обнаруживаются, но и могут быть исправлены путём замены ошибочного элемента на противоположный. Двойные ошибки обнаруживаются, но не исправляются.

Код Хемминга может использоваться в двух режимах: режиме исправления или режиме обнаружения ошибок.

В режиме исправления ошибок исправляются все одиночные ошибки в кодовой комбинации, следовательно, вероятность остаточной ошибки при декодировании равна вероятности того, что в кодовой комбинации произойдёт более одной ошибки:

$$p_{\text{к}} = \sum_{i=2}^{7} \mathrm{C}_7^i p^i (1-p)^{7-i}.$$

Для малых p

$$p_{\text{к}} \approx \mathrm{C}_7^2 p^2 (1-p)^5 \approx \mathrm{C}_7^2 p^2 = 21p^2.$$

В режиме обнаружения ошибок гарантированно обнаруживаются две ошибки, то есть вероятность необнаруженной ошибки определяется как вероятность того, что в принятой кодовой комбинации будет более двух ошибочных элементов:

$$p_{\text{но}} = \sum_{i=3}^{7} \mathrm{C}_7^i p^i (1-p)^{7-i}.$$

Для малых p

$$p_{\text{но}} \approx \mathrm{C}_7^3 p^3 (1-p)^4 \approx \mathrm{C}_7^3 p^3 = 35 p^3 .$$

Векторно-матричное представление кодов

Символы исходного сообщения и кодового слова можно рассматривать как элементы соответствующих векторов. Обозначим информационный вектор как

$$\mathbf{a} = (a_0\ a_1\ ...\ a_{k-1}),$$

а кодовый вектор как

$$\mathbf{b} = (b_0\ b_1\ ...\ b_{n-1})$$

Для линейных блоковых кодов операцию кодирования можно представить совокупностью линейных уравнений вида $b_i = a_{0,i}g_{0,i} + a_{1,i}g_{1,i} + ... + a_{k-1,i}g_{k-1,i},\ i = 0, 1, ..., n-1$.

В двоичном случае коэффициенты $g_{i,j}$ принимают значения 0 или 1. Эти линейные уравнения можно переписать в векторно-матричной форме:

$\mathbf{b} = \mathbf{aG}$

Матрица $\mathbf{G}$ называется *порождающей матрицей* кода:

$$\mathrm{G} = \begin{Vmatrix} g_0 \\ g_1 \\ ... \\ g_{k-1} \end{Vmatrix} = \begin{Vmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ ... & ... & ... & ... \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{Vmatrix}.$$

Таким образом, произвольное слово представляет собой линейную комбинацию векторов g_i — строк матрицы G:

$\mathbf{b} = a_0\mathrm{g}_0 + a_1\mathrm{g}_1 + ... + a_{k-1}\mathrm{g}_{k-1}$.

Векторы g_i должны выбираться таким образом, чтобы соблюдалось условие их линейной независимости. Такие векторы называются *базисом* (n,k) кода. Совокупность базисных векторов не единственная и, следовательно, матрица G не уникальна.

Любую порождающую матрицу (n,k) кода путём проведения операций над строками и перестановкой столбцов можно свести к систематической форме:

$$G = \|I : P\| = \left\| \begin{matrix} 1 & 0 & \dots & 0 & p_{0,0} & p_{0,1} & \dots & p_{0,r-1} \\ 0 & 1 & \dots & 0 & p_{1,0} & p_{1,1} & \dots & p_{1,r-1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & p_{k-1,0} & p_{k-1,1} & \dots & p_{k-1,r-1} \end{matrix} \right\|.$$

Здесь $\mathbf{I}$ — единичная матрица, P — матрица, которая определяет r проверочных символов. Такую форму записи проверочной матрицы называют *канонической*. После умножения информационного вектора на порождающую матрицу в канонической форме, получается кодовый вектор, в котором первые k элементов — информационные, а последние — проверочные.

Любой несистематический линейный блочный (n,k) код, полученный с помощью несистематической порождающей матрицы, эквивалентен (n,k) коду с соответствующей систематической порождающей матрицей.

С любым линейным (n,k) кодом связан *дуальный код* $(n,n-k)$. Дуальный код является линейным кодом с 2^{n-k} кодовыми векторами, которые образуют нуль-пространство по отношению к (n,k) коду. Порождающая матрица $\mathbf{H}$ дуального кода состоит из $(n-k)$ линейно независимых векторов, выбираемых в нуль-пространстве. То есть любое кодовое слово (n,k) кода ортогонально любому кодовому слову дуального $(n,n-k)$ кода. Следовательно, любое кодовое слово (n,k) кода ортогонально любой строке матрицы $\mathbf{H}$:

$\mathbf{bH}^{\mathrm{T}} = 0$.

Поскольку это равенство справедливо для любого кодового слова, то $\mathbf{GH}^{\mathrm{T}} = 0$.

Если линейный (n,k) код является систематическим, то из этого равенства следует, что $\mathbf{H} = [-\mathbf{P}^{\mathrm{T}} : \mathbf{I}]$.

Для двоичных кодов знак «–» может быть опущен, так как операции сложения и вычитания по модулю 2 идентичны.

Матрица $\mathbf{H}$ используется при декодировании (n,k) кода в качестве *проверочной*. Если выполняется условие $\mathbf{b}'\mathbf{H}^{\mathrm{T}} = 0$, значит, принятый кодовый вектор является разрешённым и ошибок нет.

Принятый вектор $\mathbf{b}' = \mathbf{b} + \mathbf{e}$, где $\mathbf{b}$ — переданный вектор, $\mathbf{e}$ — вектор ошибки.

Произведение

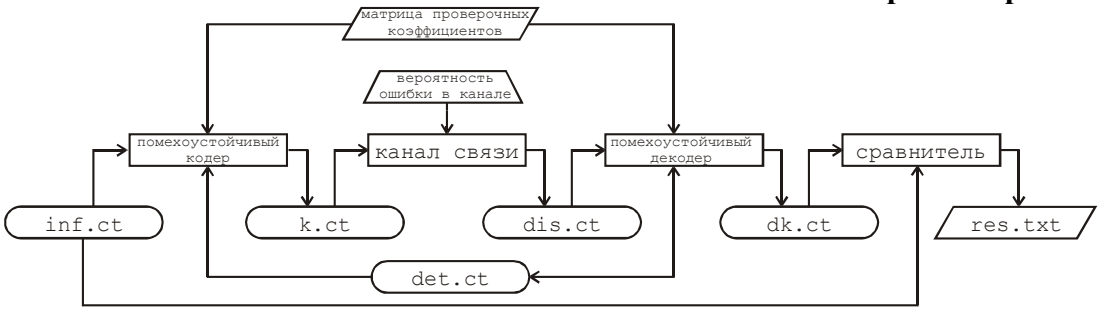
$\mathbf{b}'\mathbf{H}^{\mathrm{T}} = (\mathbf{b} + \mathbf{e})\mathbf{H}^{\mathrm{T}} = \mathbf{bH}^{\mathrm{T}} + \mathbf{eH}^{\mathrm{T}} = \mathbf{eH}^{\mathrm{T}} = \mathbf{c}$.

Вектор $\mathbf{c}$ называется *вектором синдрома*. Число элементов вектора $\mathbf{c}$ равняется числу проверочных символов. Отличие синдрома от нуля всегда указывает на наличие ошибки. Кроме того, в режиме исправления по виду синдрома можно определить вектор ошибки и, следовательно, исправить ошибки в принятой кодовой комбинации. При декодировании все возможные варианты синдрома сводятся в *таблицу синдромов*. Пример таблицы синдромов:

Синдром	Номер ошибочного разряда
001	4
010	5
011	3
…	…

Синдром $\mathbf{c}$ является характеристикой вектора ошибок, а не переданного кодового вектора. Хотя в принципе возможно 2^n вариантов ошибок, из них лишь $2^{n-k} = 2^r$ будут синдромными. Следовательно, разные ошибки могут приводить к одинаковым синдромам.

Моделирование линейных блочных систематических с использованием векторно-матричного представления



Пусть задан линейный код (8,4): n=8 k=4 r=4

Порождающая матрица: $\mathbf{G} = \begin{Vmatrix} 10000111 \\ 01001011 \\ 00101101 \\ 00011110 \end{Vmatrix}$

Информационные и кодовые комбинации:

```
0000: 00000000
0001: 00011110
0010: 00101101
0011: 00110011
0100: 01001011
0101: 01010101
0110: 01100110
0111: 01111000
1000: 10000111
1001: 10011001
1010: 10101010
1011: 10110100
1100: 11001100
1101: 11010010
1110: 11100001
1111: 11111111
dmin=4
```

Проверочная транспонированная матрица:

$$\mathbf{H}^{T} = \begin{Vmatrix} 0111 \\ 1011 \\ 1101 \\ 1110 \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{Vmatrix}$$

Вектор-синдром в десятичной и двоичной системе счисления, а также маска для бита, который является ошибочным

		76543210 номера разрядов
7	0111	10000000
11	1011	01000000
13	1101	00100000
14	1110	00010000
8	1000	00001000
4	0100	00000100
2	0010	00000010
1	0001	00000001

Все остальные варианты синдромов соответствуют обнаружению ошибки

Кодирование байта информации: cp1251 «ы»: 11011011 он разбивается на два полубайта: 1101 1011

Полубайты становятся двумя последовательными 4-разрядными информационными комбинациями. Умножаем 4-разрядные информационные комбинации на порождающую матрицу, получаем 8-разрядные разрешённые кодовые комбинации:

$$\mathbf{a}_{13}\mathbf{G} = \left\|1101\right\| \begin{Vmatrix} 10000111 \\ 01001011 \\ 00101101 \\ 00011110 \end{Vmatrix} = \begin{Vmatrix} 1\cdot1\oplus1\cdot0\oplus0\cdot0\oplus1\cdot0=1\oplus0\oplus0\oplus0=1 \\ 1\cdot0\oplus1\cdot1\oplus0\cdot0\oplus1\cdot0=0\oplus1\oplus0\oplus0=1 \\ 1\cdot0\oplus1\cdot0\oplus0\cdot1\oplus1\cdot0=0\oplus0\oplus0\oplus0=0 \\ 1\cdot0\oplus1\cdot0\oplus0\cdot0\oplus1\cdot1=0\oplus0\oplus0\oplus1=1 \\ 1\cdot0\oplus1\cdot1\oplus0\cdot1\oplus1\cdot1=0\oplus1\oplus0\oplus1=0 \\ 1\cdot1\oplus1\cdot0\oplus0\cdot1\oplus1\cdot1=1\oplus0\oplus0\oplus1=0 \\ 1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1=1\oplus1\oplus0\oplus1=1 \\ 1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0=1\oplus1\oplus0\oplus0=0 \end{Vmatrix} = \left\|11010010\right\| = \mathbf{b}_{13}$$

$$\mathbf{a}_{11}\mathbf{G} = \left\|1011\right\| \begin{Vmatrix} 10000111 \\ 01001011 \\ 00101101 \\ 00011110 \end{Vmatrix} = \begin{Vmatrix} 1\cdot1\oplus0\cdot0\oplus1\cdot0\oplus1\cdot0=1\oplus0\oplus0\oplus0=1 \\ 1\cdot0\oplus0\cdot1\oplus1\cdot0\oplus1\cdot0=0\oplus0\oplus0\oplus0=0 \\ 1\cdot0\oplus0\cdot0\oplus1\cdot1\oplus1\cdot0=0\oplus0\oplus1\oplus0=1 \\ 1\cdot0\oplus0\cdot0\oplus1\cdot0\oplus1\cdot1=0\oplus0\oplus0\oplus1=1 \\ 1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus1\cdot1=0\oplus0\oplus1\oplus1=0 \\ 1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot1=1\oplus0\oplus1\oplus1=1 \\ 1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus1\cdot1=1\oplus0\oplus0\oplus1=0 \\ 1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus1\cdot0=1\oplus0\oplus1\oplus0=0 \end{Vmatrix} = \left\|10110100\right\| = \mathbf{b}_{11}$$

Если в канале нет ошибок, то кодовые комбинации остаются неизменными, то есть разрешёнными. Результат умножения разрешённых кодовых комбинаций на проверочную матрицу даёт нулевой 4-х разрядный синдром

$$\mathbf{b}_{13}\mathbf{H}^{\mathrm{T}} = \left\|11010010\right\| \begin{Vmatrix} 0111 \\ 1011 \\ 1101 \\ 1110 \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{Vmatrix} = \begin{Vmatrix} 1\cdot0\oplus1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus1\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0 \\ 1\cdot1\oplus1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0 \\ 1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus1\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0=0 \\ 1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0=0 \end{Vmatrix} = \left\|0000\right\| = \mathbf{c}_0$$

$$\mathbf{b}_{11}\mathbf{H}^{T}=\left\|10110100\right\|\left\|\begin{matrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{matrix}\right\|=\left\|\begin{matrix}1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0=0\oplus0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0=1\oplus0\oplus1\oplus1\oplus0\oplus1\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus1\cdot1\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus1\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot0\oplus0\cdot1=1\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0=0\end{matrix}\right\|=\left\|0000\right\|=\mathbf{c}_{0}$$

Пусть в канале есть ошибки. Сделаем одну ошибку в 6-м разряде в кодовой комбинации $\mathbf{b}_{13}$ и две ошибки в 4-м и 1-м разряде в кодовой комбинации $\mathbf{b}_{11}$.

$$\mathbf{b}_{13}=\left\|11010010\right\|\rightarrow\mathbf{b}'_{13}=\left\|10010010\right\|\qquad\mathbf{b}_{11}=\left\|10110100\right\|\rightarrow\mathbf{b}''_{11}=\left\|10100110\right\|$$

Результат умножения полученных кодовых комбинаций на проверочную матрицу даёт ненулевые 4-х разрядные синдромы

$$\mathbf{b}_{13}\mathbf{H}^{T}=\left\|10010010\right\|\left\|\begin{matrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{matrix}\right\|=\left\|\begin{matrix}1\cdot0\oplus0\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=1\\1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0=1\\1\cdot1\oplus0\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0=1\end{matrix}\right\|=\left\|1011\right\|=\mathbf{c}_{11}$$

Синдром, содержащий комбинацию 1011, есть в таблице синдромов и как раз соответствует предположению об ошибке в 6-м разряде. Поэтому следующим действием будет инвертирование 6-го разряда в кодовой комбинации (исправление ошибки)

$$\mathbf{b}'_{13}=\left\|1\bar{0}010010\right\|\rightarrow\mathbf{b}_{13}=\left\|\underline{1}1\underline{1}0\underline{1}0010\right\|\rightarrow1101$$

после этого старшие 4 разряда являются искомыми, которые и передаются на выход декодера: 1101

$$\mathbf{b}_{11}\mathbf{H}^{\mathrm{T}}=\left\|10100110\right\|\left\|\begin{matrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{matrix}\right\|=\left\|\begin{matrix}1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot1\oplus1\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0=1\\1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0\oplus0=1\\1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus1\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0=0\end{matrix}\right\|=\left\|1100\right\|=\mathbf{c}_{12}$$

Здесь синдром содержит комбинацию 1100, которой нет в таблице синдромов. Если отсутствует канал обратной связи, то информационные символы считаются стёртыми. Можно их закодировать символами «2»:

$$\mathbf{b}''_{11}=\left\|10100110\right\|\rightarrow 2222$$

Таким образом, на выходе декодера получается следующая последовательность символов: 11012222
Можно запоминать номера стёртых информационных комбинаций (сформировать из них файл). Тогда при повторной работе кодера на его выходе будут формироваться кодовые комбинации, соответствующие информационным комбинациям, только для тех, порядковые номера которых, записаны в файле.
Последовательность на выходе кодера также может быть подвержена искажениям в виде изменения некоторых бит (с 0 на 1 или с 1 на 0). Декодер также сделает произведение на проверочную матрицу и в случае положительного исхода согласно порядковым номерам в соответствующем файле заменит стёртые символы на искомые. Если останутся стёртые символы, то файл с порядковыми номерами для стёртых информационных комбинаций будет обновлён. При этом размер этого файла либо не изменится (что крайне маловероятно) или станет меньше.
Процесс будет повторяться до тех пор, пока размер файла с порядковыми номерами для стёртых информационных комбинаций не станет равным нулю.
Пример программ на C++, реализующих работу кодера и декодера приведён ниже. Входной файл с матрицей проверочных символов представляет собой текстовый файл. Примеры для кода Хемминга (7,4) и для линейного кода (8,4) приведены также.

$$\mathbf{P}_{(7,4)}=\left\|\begin{matrix}111\\011\\101\\110\end{matrix}\right\|,\quad \mathbf{P}_{(8,4)}=\left\|\begin{matrix}0111\\1011\\1101\\1110\end{matrix}\right\|.$$

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int size,i,j,k,n,r,l,dmin,N; char h,c=0; long long s; long u;
if(mn==5||mn==4); else { cerr<<nm[0]<<" in.ct out.ct p.txt r.lb\n"; return 1; }
ifstream det;
ifstream pk(nm[3],ios::binary); if(!pk) { cerr<<"input file \""<<nm[3]<<"\" not open!\n"; return 1; }
ofstream ou(nm[2],ios::binary); if(!ou) { cerr<<"output file \""<<nm[2]<<"\" not open!\n"; return 1; }
ifstream in(nm[1],ios::binary); if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n"; return 1; }
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
pk.seekg(0,ios::end);size=pk.tellg();pk.seekg(0,ios::beg);
char* p=new char[size+2]; p[size]=0x0D;p[size+1]=0; pk.read(p,size); pk.close();
for(j=0,i=-1;p[++i];) if((p[i]|1)==0x31||p[i]==0x0D) p[j++]=p[i]; p[j]=0;
for(j=0,i=-1;p[++i];c=p[i]) if(p[i]==0x0D&&c==0x0D); else p[j++]=p[i]; p[j]=0;
for(r=0;p[r++]!=0x0D;); k=j/r--; n=k+r;
cout<<"(n,k)=("<<n<<","<<k<<") r="<<r<<endl;
long long* G=new long long[k]; memset(G,0,k*sizeof(long long));
for(l=i=0;i<k;l++,G[i++]>>=1)
 { for(j=0;j<k;G[i]<<=1) G[i]|=(i==j++);
 for(j=-1;p[++j]!=0x0D;G[i]<<=1) G[i]|=p[l++]&1;
 }
for(i=0;i<(1<<k);i++) for(s=0,l=1<<(k-1),j=0;j<k;s^=G[j++]*!!(i&l),l>>=1);
for(dmin=n,i=1;i<1<<k;i++)
 { for(l=s=0,j=i;j;j>>=1) s^=G[l++]*!!(j&1);
 for(l=0;s;s>>=1) l+=!!(s&1); if(dmin>l) dmin=l;
 }
if(mn==5) { det.open(nm[4],ios::binary); if(!det) mn=4; }
cout<<"dmin="<<dmin<<endl;
if(mn!=5)
 for(N/=k,i=0;i<N;i++)
  { for(s=0,j=0;j<k;c=in.get()&1,s^=G[j++]*c);
  for(l=1<<(n-1);l;l>>=1) c=!!(l&s)+h,ou.put(c);
  }
else
 { det.seekg(0,ios::end);N=det.tellg()/sizeof(long);det.seekg(0,ios::beg);
 for(i=0;i<N;i++)
  { det.read((char*)&u,sizeof(long)); in.seekg(u*sizeof(long));
  for(s=0,j=0;j<k;c=in.get()&1,s^=G[j++]*c);
  for(l=1<<(n-1);l;l>>=1) c=!!(l&s)+h,ou.put(c);
  }
 det.close();
 }
ou.close(); in.close(); delete [] G; delete [] p; return 0;
}
```

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int size,i,j,k,n,r,l,dmin,icr=0,N; char h,c=0; long long s,w;
char *p,*d=NULL; long long *G,*er,*H; int* si;
if(mn==5||mn==4); else { cerr<<nm[0]<<" in.ct out.ct p.txt <r.lb>\n"; return 1; }
ifstream det; fstream ou; ifstream cr(nm[2]); icr=!!cr; if(icr) cr.close(); else { ou.open(nm[2],ios::out); ou.close(); }
ifstream pk(nm[3],ios::binary); if(!pk) { cerr<<"input file \""<<nm[3]<<"\" not open!\n"; return 1; }
ou.open(nm[2],ios::binary|ios::in|ios::out); if(!ou) { cerr<<"output file \""<<nm[2]<<"\" not open!\n"; return 1; }
ifstream in(nm[1],ios::binary); if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n"; return 1; }
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
pk.seekg(0,ios::end);size=pk.tellg();pk.seekg(0,ios::beg);
p=new char[size+2]; p[size]=0x0D;p[size+1]=0; pk.read(p,size); pk.close();
for(j=0,i=-1;p[++i];) if((p[i]|1)==0x31||p[i]==0x0D) p[j++]=p[i]; p[j]=0;
for(j=0,i=-1;p[++i];c=p[i]) if(p[i]==0x0D&&c==0x0D); else p[j++]=p[i]; p[j]=0;
for(r=0;p[r++]!=0x0D;); k=j/r--; n=k+r;
cout<<"(n,k)=("<<n<<","<<k<<") r="<<r<<endl;
G=new long long[k]; memset(G,0,k*sizeof(long long));
si=new int[n];
er=new long long[n];
for(l=i=0;i<k;l++,G[i++]>>=1) { for(j=0;j<k;G[i]<<=1) G[i]|=(i==j++); for(j=-1;p[++j]!=0x0D;G[i]<<=1) G[i]|=p[l++]&1; }
for(dmin=n,i=1;i<1<<k;i++) { for(l=s=0,j=i;j;j>>=1) s^=G[l++]*!!(j&1); for(l=0;s;s>>=1) l+=!!(s&1); if(dmin>l) dmin=l; }
if(mn==5) det.open(nm[4],ios::binary);
H=new long long[n]; cout<<"dmin="<<dmin<<endl;
for(i=0;i<k;i++) H[i]=G[i]&((1<<r)-1);
for(j=1<<r;j>>=1;) H[i++]=j;
for(w=1<<(n-1),j=0;j<n;si[j]=H[j],er[j++]=w,w>>=1);
for(N/=n,i=0;i<N;i++)
 { for(w=0,l=j=0;j<n;c=in.get()&1,l^=H[j++]*c,w<<=1,w|=c);
 if(l) for(j=0;j<N;j++) if(l==si[j]) { w^=er[j]; l=0; break; }
 l=!!l;  w*=!l;  l<<=1;  w>>=r;
 if(cr&&det) det.read((char*)&s,sizeof(long)),ou.seekg(s*k);
 for(j=1<<(k-1);j;j>>=1) c=l+!!(j&w)+h,ou.put(c);
 }
ou.close(); in.close(); if(cr&&det) det.close();
if(mn==5)
 { ifstream out(nm[2],ios::binary); ofstream det(nm[4],ios::binary);
out.seekg(0,ios::end);N=out.tellg();out.seekg(0,ios::beg);
 for(l=s=0;s<N/k;s++) { out.seekg(s*k); c=out.get()-h; if(c==2) l=1,det.write((char*)&s,sizeof(long)); }
 det.close(); out.close(); if(!l) remove(nm[4]);
 }
if(d) delete [] d; delete [] G; delete [] H; delete [] p; delete [] si; delete [] er;  return 0;
}
```

Канал связи, моделирующий ошибки с заданной вероятностью p:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int i,N; char c,h; srand(time(NULL));
if(mn!=4) { cerr<<nm[0]<<" in.ct out.ct Perr\n"; return 1; }
ifstream in(nm[1],ios::binary); if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n"; return 1; }
ofstream ou(nm[2],ios::binary); if(!ou) { cerr<<"output file \""<<nm[2]<<"\" not open!\n"; return 1; }
float perr=atof(nm[3]); if(perr<0||perr>1) { cerr<<"error Perr\n"; return 1; }
long p=RAND_MAX*perr;
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
for(i=0;i<N;i++,c=in.get()&1,c^=(rand()<p),ou.put(c+h));
return 0;
}
```

Для осуществления обратной связи, если имеется в наличии режим обнаружения ошибок, ниже приведён командный файл в среде Linux:

```
touch 2.lb
./gen 1.ct 100 0.5
while
   ./linkod 1.ct 2.ct g3.txt 1.lb
   ./chan 2.ct 3.ct 0.1
   ./lindek 3.ct 4.ct g3.txt 1.lb
   [ -e 1.lb ]
do
   cat 2.lb 1.lb > 2.lb.tmp
   mv 2.lb.tmp 2.lb
done
cmp 1.ct 4.ct
```

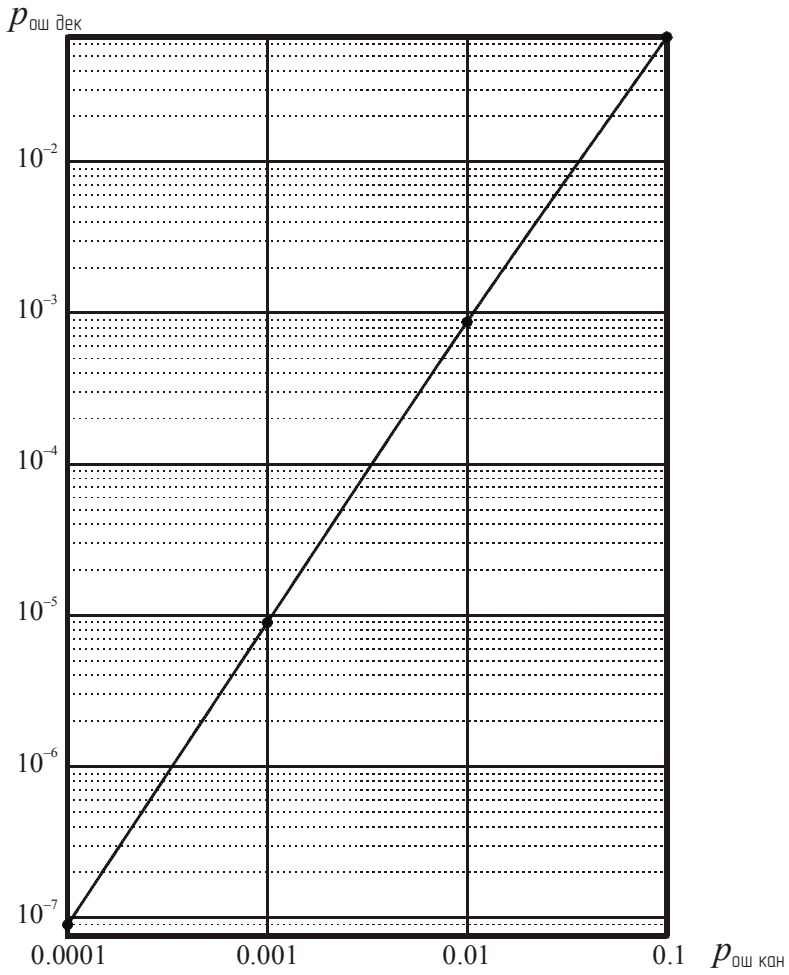
Аналогичный файл в среде Windows:

```
copy nul 2.lb
gen 1.ct 100 0.5
:b
linkod 1.ct 2.ct g3.txt 1.lb
chan 2.ct 3.ct 0.1
lindek 3.ct 4.ct g3.txt 1.lb
copy 2.lb+1.lb /b
if exist 1.lb  goto b
cmp 1.ct 4.ct
```

Вероятность ошибки для кода Хемминга (7,4):

$$p_{\text{ош дек}} = 9p^2 - 26p^3 + 38p^4 - 12p^5.$$

График помехоустойчивости для кода Хемминга (7,4) приведён ниже как зависимость вероятности ошибки декодирования от вероятности ошибки в канале связи:



Полиномиальное представление кодов, циклические коды

Кодовое слово также можно представить в виде полинома степени $\le n-1$:

$B(p)=b_{n-1}p^{n-1}+b_{n-2}p^{n-2}+...+b_1p+b_0$

Для двоичного кода каждый из коэффициентов полинома b_i равен или 0, или 1.

Определим также полином информационного слова

$A(p)=a_{k-1}p^{k-1}+a_{k-2}p^{k-2}+...+a_1p+a_0$

и *порождающий полином*

$G(p)=p^{n-k}+g_{n-k-1}p^{n-k-1}+...+g_1p+1$.

Тогда операция кодирования будет иметь вид

$B(p)=A(p)G(p)$.

Поскольку степень информационного полинома не выше $k-1$, а порождающего — не выше $n-k$, степень их произведения будет не выше $n-1$ — максимальной степени кодового слова.

Циклическими называют коды, для которых циклический сдвиг разрешённой кодовой комбинации также является разрешённой кодовой комбинацией:

$b_i=\left[b_{n-1}b_{n-2}\,...\,b_1\,b_0\right]\rightarrow b_j=\left[b_{n-2}\,b_{n-3}\,...\,b_0\,b_{n-1}\right]$.

В полиномиальной форме циклический сдвиг записывается как:

$pB(p)=b_{n-1}p^n+b_{n-2}p^{n-1}+...+b_1p^2+b_0p$;

$\frac{pB(p)}{p^n-1}=b_{n-1}+\frac{B_1(p)}{p^n+1}$;

$B_1(p)=b_{n-2}p^{n-1}+b_{n-3}p^{n-2}+...+b_0p+b_{n-1}$.

То есть циклически сдвинутая на одну позицию комбинация получается как остаток от деления $pB(p)$ на полином p^n+1:

$B_1(p)=pB(p)\bmod\left(p^n+1\right)$.

Аналогично, сдвиг на i позиций можно записать как

$B_i(p)=p^iB(p)\bmod\left(p^n+1\right)$.

В общем виде можно записать

$p^iB(p)=Q(p)\left(p^n+1\right)+B_i(p)$.

Здесь $B_i(p)$ — кодовое слово циклического сдвига, $Q(p)$ — частное.

Порождающий полином циклического кода $G(p)$ получается как один из множителей при факторизации полинома p^n+1:

$p^n+1=G(p)H(p)$.

Полином $H(p)$ будет с одной стороны являться *проверочным* для циклического (n,k) кода, а с другой стороны — *порождающим* для дуального $(n,n-k)$ кода.

Из порождающего полинома (n,k) кода можно получить порождающую матрицу. Порождающая матрица должна состоять из любого набора k линейно независимых векторов. По заданному порождающему полиному $G(p)$ такой набор генерируется циклическим сдвигом этого полинома:

$p^{k-1}G(p), p^{k-2}G(p), ..., pG(p), G(p)$.

Проверочная матрица получается аналогичным образом из соответствующего проверочного полинома.

Декодирование циклических кодов

Принимаемый кодовый вектор в полиномиальной форме

$B'(p)=B(p)+E(p)=A(p)G(p)+E(p)$.

Здесь $E(p)$ — полиномиальная запись вектора ошибки. Разделим $B'(p)$ на порождающий полином:

$$\frac{B'(p)}{G(p)}=A(p)+\frac{E(p)}{G(p)}=Q(p)+\frac{R(p)}{G(p)}.$$

Здесь $R(p)$ — остаток от деления $B'(p)$ на $G(p)$, представляющий собой полином степени не выше $n-k-1$, $Q(p)$ — частное. Поскольку первое (полезное) слагаемое $B'(p)$ делится на $G(p)$ без остатка, остаток от деления $R(p)$ будет определяться только вектором ошибки $E(p)$. Если $C(p)=R(p)=0$, то есть принятый полином делится на порождающий без остатка, то ошибок нет.

Вместо деления на порождающий полином можно умножать принятый вектор на проверочный полином $H(p)$.

Примером циклического кода является код Хемминга $(7,4)$.

Циклическое кодирование в систематической форме.

Систематическая форма позволяет уменьшать сложность процедуры кодирования и декодирования.

Вектор сообщения в полиномиальной форме:

$$\mathbf{m}(X) = m_0 + m_1 X + m_2 X^2 + \ldots + m_{k-1} X^{k-1}.$$

Символы сообщения используются как часть кодового слова.

$$X^{n-k}\mathbf{m}(X) = m_0 X^{n-k} + m_1 X^{n-k+1} + \ldots + m_{k-1} X^{n-1}. \quad (1)$$

Разделим (1) на $\mathbf{g}(X)$

$$X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{p}(X). \quad (2)$$

Остаток в результате деления:

$$\mathbf{p}(X) = p_0 + p_1 X + p_1 X^2 + \ldots + p_{n-k-1} X^{n-k-1}$$

Можно (2) записать в виде:

$$\mathbf{p}(X) = X^{n-k}\mathbf{m}(X) \bmod \mathbf{g}(X). \quad (3)$$

Прибавим $\mathbf{p}(X)$ к обеим частям (2) и используя сложение по модулю 2, получим:

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) = \mathbf{U}(X). \quad (4)$$

Левая часть уравнения (4) является действительным полиномом кодового слова, так как это полином степени $n-1$ или менее, который при делении на $\mathbf{g}(X)$ даёт нулевой остаток. Это кодовое слово через все члены полинома запишется в следующем виде:

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = p_0 + p_1 X + p_2 X^2 + \ldots + p_{n-k-1} X^{n-k-1} + m_0 X^{n-k} + m_1 X^{n-k+1} + \ldots + m_{k-1} X^{n-1}.$$

Полином кодового слова соответствует вектору кода

$$\mathbf{U} = \left(\underbrace{p_0, p_1, \ldots, p_{n-k-1}}_{\substack{n-k \\ \text{бит чётности}}}, \underbrace{m_0, m_1, \ldots, m_{k-1}}_{\substack{k \\ \text{бит сообщения}}} \right)$$

Пример:

С помощью полиномиального генератора $\mathbf{g}(X) = 1 + X + X^3$ получим систематическое кодовое слово из набора кодовых слов (7,4) для вектора сообщения $\mathbf{m} = 10011$.

$\mathbf{m}(X) = 1 + X^2 + X^3,\ n = 7,\ k = 4,\ n - k = 3;$

$X^{n-k}\mathbf{m}(X) = X^3\left(1 + X^2 + X^3\right) = X^3 + X^5 + X^6$

Разделив $X^{n-k}\mathbf{m}(X)$ на $\mathbf{g}(X)$, можно записать следующее:

$$X^3 + X^5 + X^6 = \left(\underbrace{1 + X + X^2 + X^3}_{\text{частное } \mathbf{q}(X)}\right)\left(\underbrace{1 + X + X^3}_{\text{генератор } \mathbf{g}(X)}\right) + \left(\underbrace{1}_{\text{остаток } \mathbf{p}(X)}\right)$$

$\mathbf{U}(X) = \mathbf{p}(X) + X^3\mathbf{m}(X) = 1 + X^3 + X^4 + X^6$

$$\mathbf{U} = \underbrace{1\ \ 0\ \ 0}_{\text{биты чётности}}\ \ \underbrace{1\ \ 0\ \ 1\ \ 1}_{\text{биты сообщения}}$$

Логическая схема для реализации полиномиального деления.

При циклическом сдвиге полинома кодового слова и кодирования полинома сообщения применяется операция деления полиномов друг на друга. Такие операции реализуются с помощью регистра сдвига с обратной связью.

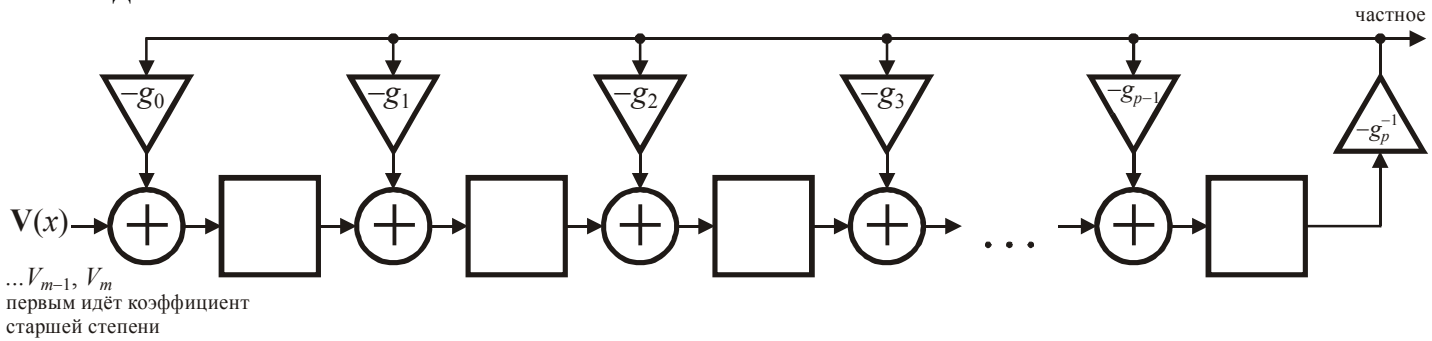
Пусть даны 2 полинома

$$\mathbf{V}(X) = v_0 + v_1 X + v_2 X^2 + \ldots + v_m X^m$$

$$\mathbf{g}(X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_p X^p$$

при этом $m \geq p$

Схема деления



Выполняет деление $\mathbf{V}(X)$ на $\mathbf{g}(X)$, определяя частное и остаток

$$\frac{\mathbf{V}(X)}{\mathbf{g}(X)} = \mathbf{q}(X) + \frac{\mathbf{p}(X)}{\mathbf{g}(X)}.$$

В исходном состоянии разряды регистра содержат нули.

Коэффициенты $\mathbf{V}(X)$ поступают и продвигаются по регистру сдвига по одному за такт, начиная с коэффициентов более высокого порядка.

После *p*-го сдвига частное на выходе равно $\mathbf{g}_p^{-1} v_m$; это слагаемое наивысшего порядка в частном.

Далее для каждого коэффициента частного q_i из делимого вычитается полином $q_i \mathbf{g}(X)$. Это вычитание обеспечивает обратная связь.

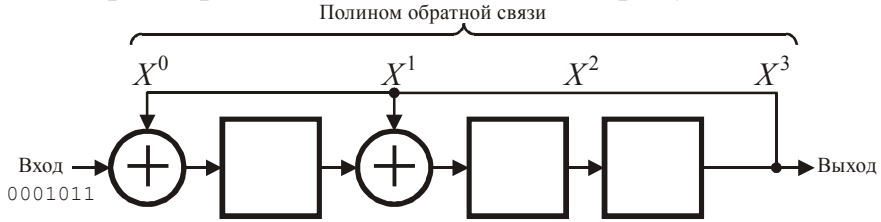
Разность крайних слева p слагаемых остаётся в делимом, а слагаемое обратной связи $q_i \mathbf{g}(X)$ формируется при каждом сдвиге схемы и отображается в виде содержимого регистра.

При каждом сдвиге регистра разность смещается на один разряд; слагаемое наивысшего порядка удаляется, в то время как следующий значащий коэффициент в $\mathbf{V}(X)$ перемещается на его место.
После всех $m+1$ сдвигов регистра, на выход последовательно выдаётся частное, а остаток остаётся в регистре.

Пример схемы полиномиального деления
Разделим $\mathbf{V}(X)=X^3+X^5+X^6$ $(\mathbf{V}=0001011)$ на $\mathbf{g}(X)=(1+X+X^3)$. Найдём частное и остаток.
Схема деления должна выполнить следующее действие:

$$\frac{X^3+X^5+X^6}{1+X+X^3}=\mathbf{q}(X)+\frac{\mathbf{p}(X)}{1+X+X^3}$$

Регистр с обратной связью показан на рисунке.



Первоначально регистр содержит нули. Схема выполнит следующие шаги:

Входная очередь	**Номер сдвига**	**Содержимое регистра**	**Выход и обратная связь**
0001011	0	000	-
000101	1	100	0
00010	2	110	0
0001	3	011	0
000	4	011	1
00	5	111	1
0	6	101	1
-	7	100	1

После 4-го сдвига коэффициенты частного $\{q_i\}$, последовательно поступающие с выхода, выглядят как 1111 или же полином имеет вид $\mathbf{q}(X)=1+X+X^2+X^3$. Коэффициенты остатка $\{p_i\}$ имеют вид 100, то есть полином остатка имеет вид $p(X)=1$. Таким образом схема выполнила следующие вычисления:

$$\frac{X^3+X^5+X^6}{1+X+X^3}=1+X+X^2+X^3+\frac{1}{1+X+X^3}.$$

Прямое деление полиномов:

$$\begin{array}{lll}
 & X^6+X^5\quad\quad+X^3 & \big|\,X^3+X+1 \\
\text{обратная связь после 4-го сдвига} \rightarrow & \underline{X^6+\quad\quad X^4+X^3} & X^3+X^2+X+1 \\
\text{регистр после 4-го сдвига} \rightarrow & \quad X^5+X^4 & \uparrow\ \uparrow\ \uparrow\ \uparrow \\
\text{обратная связь после 5-го сдвига} \rightarrow & \quad \underline{X^5+\quad\quad X^3+X^2} & 4\quad 5\quad 6\quad 7 \\
\text{регистр после 5-го сдвига} \rightarrow & \quad\quad X^4+X^3+X^2 & \\
\text{обратная связь после 6-го сдвига} \rightarrow & \quad\quad \underline{X^4+\quad\quad X^2+X} & \\
\text{регистр после 6-го сдвига} \rightarrow & \quad\quad\quad X^3+\quad\quad X & \\
\text{обратная связь после 7-го сдвига} \rightarrow & \quad\quad\quad \underline{X^3+\quad\quad X+1} & \\
\text{регистр после 7-го сдвига} \rightarrow & \quad\quad\quad\quad\quad\quad 1 & \\
\text{(остаток)} & &
\end{array}$$

Свёрточные коды

Свёрточные коды не являются блочными и не являются систематическими. Свёрточный кодер представляет собой сдвиговый регистр с отводвами

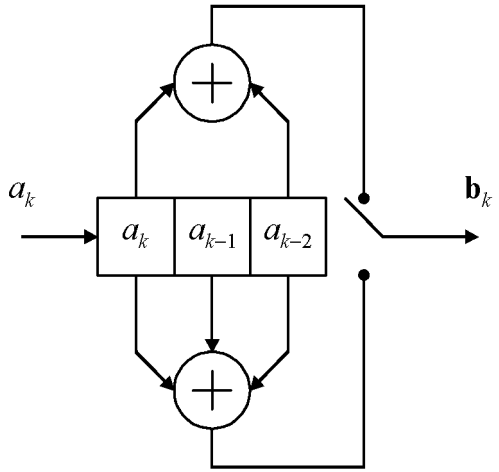


Рис. 1. Пример построения свёрточного кода со скоростью $R = \frac{1}{2}$

Число ячеек сдвигового регистра $K = v + 1$. Величину v называют *кодовым ограничением* свёрточного кодера. Кодовое ограничение определяет *память* свёрточного кодера.

Конфигурация отводов кодера определяется коэффициентами порождающих полиномов $G_1, G_2, \ldots$. Для кодера, изображённого на рисунке, порождающие полиномы равны соответственно: $G_1(p) = p^2 + 1$, $G_2(p) = p^2 + p + 1$. В векторной форме записи: $\mathbf{g}_1 = [101]$, $\mathbf{g}_2 = [111]$.

Порождающие полиномы свёрточного кода принято записывать в восьмеричной форме: $G_1 = 5$, $G_2 = 7$.

Реакция свёрточного кодера на изолированную единицу (последовательность вида $100000\ldots$) называется *импульсной характеристикой* (ИХ) кодера. Число единиц («вес») ИХ определяет *свободное расстояние* свёрточного кода $d_{\text{св}}$.

Свободное расстояние является эквивалентом кодового расстояния $d_{\min}$ для блочных кодов, оно позволяет оценить исправляющую способность свёрточного кода:

$$q_{\text{и}} < \frac{d_{\text{св}}}{2}.$$

Считается, что свёрточный декодер исправляет не более $q_{\text{и}}$ ошибок на интервале удвоенного кодового ограничения $n = 2v$. Вероятность ошибки декодирования для свёрточного кода можно оценить через вероятность того, что в выбранном наугад отрезке декодируемой последовательности длиной $n = 2v$ произойдёт $q_{\text{и}} + 1$ ошибок:

$$p_{\text{к}} = \mathrm{C}_{2v}^{q_{\text{и}}+1} p^{q_{\text{и}}+1} (1-p)^{2v-q_{\text{и}}+1}.$$

Принципы сжатия JPEG

Что такое формат сжатия изображений **JPEG** никому объяснять, наверное, не нужно. Лучше я постараюсь описать как он, собственно, работает, на каких принципах основано сжатие и как конкретно работают его алгоритмы.

Для простоты не буду касаться вплотную принципов цветного сжатия - это не так важно для понимания основ. Цветное сжатие осуществляется совершенно аналогично черно-белому.

Итак, у нас есть некое черно-белое изображение. Самый распространенный способ хранения таких изображений - каждой точке на экране соответствует один байт (8 бит - 256 возможных значений), определяющий её яркость. 255 - яркость максимальная (белый цвет), 0 - минимальная, черный. Промежуточные значения составляют всю остальную гамму серых цветов.

Самый первый этап - картинка разбивается на квадраты размером 8x8 пикселей - это справедливо для всех вариантов JPEG. Какое-либо сжатие более крупными блоками не производится. Информация каждого блока подвергается **дискретному косинусному преобразованию** (Discrete Cosine Transform, **DCT**) - разновидность гармонического (спектрального) анализа. Все преобразования, которые обычно проделываются над сигналами при их цифровой обработке, так или иначе сводятся к разложению функции в другие, так называемые базисные функции.

[например - преобразование Фурье (DFT, или более быстрый метод - FFT, см. FFT анализ), раскладывает функцию на свободные синусоиды - то есть амплитуды отдельных частот и их фазы. **N** элементов исходной функции преобразуется в **N/2 пар** значений вида {для данной частоты - амплитуда синуса, амплитуда косинуса}, которые почти всегда удобнее преобразовать и воспринимать как пару {амплитуда данной частоты, её фаза}]

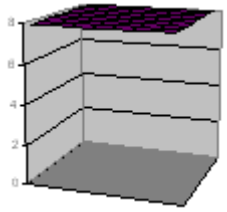
1	2	3	4	...			
9	...						
17	...						
...							
						...	56
					...	63	64

DCT тоже раскладывает функцию в базисные функции, которые представляют собой синусоиды, но без фазы - то есть привязанные к определенному пространственному положению. Понять, откуда следует его название 'косинусное', можно например так: если мы возьмем 128 значений - порядковые номера 1, 2, 3, ..., 128, и сформируем из них 255 значений по принципу отражения - 1, 2, 3, ..., 127, 128, 127, ..., 2, 1, и применим к получившемуся ряду значений преобразование Фурье, оно даст нам 128 пар значений {для данной частоты - амплитуда синуса, амплитуда косинуса}, в которых амплитуда синуса всё время будет **нулевая** - останутся одни косинусы. Это свойство симметричного относительно своей середины набора данных. Выкинем нулевые синусы - и мы, собственно, проделали косинусное преобразование (DCT) исходных данных - не самым эффективным способом, но это именно DCT.

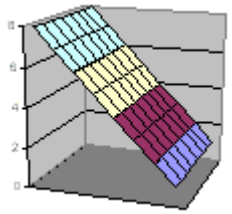
Таким образом, элементы квадрата изображения нумеруются (см. рисунок выше), выстраиваются в линейный массив и над ними производится DCT, давая нам другие 64 элемента - коэффициенты базисных частот - синусоид без фазы.

Поясняющие рисунки: поскольку мы на этот раз, в отличии от звука, работаем в двухмерном пространстве, можно разместить базисные функции в том же двухмерном пространстве 8x8. Базисная функция - какой-то характерный признак изображения этого квадрата. Трехмерные диаграммы (поверхности) следует понимать так - чем выше поверхность над данной точкой, тем светлее элемент изображения в этом месте. Еще раз приведен квадрат - просто так для удобства - и изображения базисных функций, амплитуды которых мы получаем в соответствующих местах этого же квадрата после DCT. Координаты вида [x, y] даны в квадратных скобках, а порядковый номер элемента - одномерный способ нумерации, изображенный на квадрате, подписан внизу.

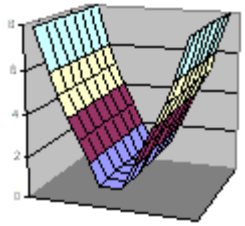
Базисные функции JPEG



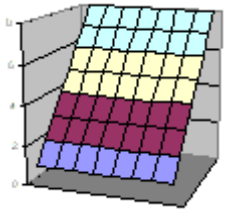
[1,1] -> (1-й элемент (1-я базисная функция) - константа)



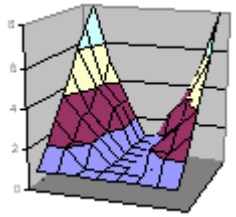
[2,1] ->
(2-й элемент - половина периода синусоиды, горизонтальная)



[3,1] ->
(3-й элемент - синусоида, горизонтальная)

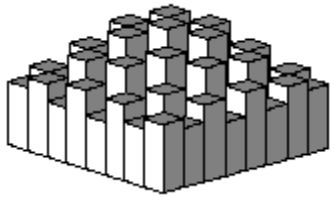


[1,2] ->
(9-й элемент - половина периода синусоиды, вертикальная)



[3,2] ->
(11-й элемент - комбинация вертикальной составляющей 9-го элемента и горизонтальной 3-го)
[прим.: Изображения функций **схематично**. 9-й и 2-й элемент - это вообще-то синусоидальные, то есть немного изогнутые поверхности, а не плоскости. :)
Общая форма, однако, именно такая]

В принципе, ничто в самом принципе сжатия не заставляло нас располагать базисные функции в двухмерную схему 8x8 - мы вполне могли бы просто запомнить, что их 64 штуки, и мы после DCT имеем разложение изображения 8x8 на эти функции. Но такое двухмерное представление дает заметить интересные особенности - видно, что горизонтальная координата положения базисной функции характеризует горизонтальную составляющую изменений изображения в исходном квадрате, вертикальная координата - вертикальную составляющую. Чем больше, к примеру, коэффициент перед базисной функцией, расположенной более справа, тем больше резких переходов изображения в горизонтальной плоскости мы имеем.



Например, если коэффициент перед 8-й базисной функцией (координаты [8,1]) максимален - мы имеем на изображении **вертикальную** сетку с шириной линий и промежутков в 1 пиксель (пиксели в **горизонтальной** плоскости чередуются с максимальной частотой), тогда как максимальный коэффициент 57-й базисной функции (координаты [1,8]) означал бы горизонтальную сетку изображения. Важно понимать (если вы еще что-то понимаете :), что функции, характеризующие оба направления колебаний сразу, представляет собой не простое сочетание горизонтальных и вертикальных составляющих колебаний - это **умножение** 'горизонтальных' и 'вертикальных' косинусов. К примеру, 64-я базисная функция (координаты [8,8]) представляет собой максимальные колебания максимальной частоты в центре квадрата, затухающие до нуля к краям, и выглядит это дело так, как изображено на сером рисунке справа (где-то в интернете нашел готовую диаграмму и сам рисовать не стал :). Это **не** готовая сетка сразу по обоим направлениям, как можно было подумать.

Итак, что мы в результате имеем? Мы разбили изображение на фрагменты 8x8 пикселей, в каждом из этих фрагментов, состоящим из 64-х точек, преобразовали изображение в базисные функции с помощью DCT, характеризующие горизонтальные и вертикальные синусоиды. Если разместить результат разложения - коэффициенты базисных функций - обратно в квадрат 8x8, мы получим такую тенденцию: чем правее координата, тем больше частота в горизонтальной плоскости, чем ниже - тем больше частота в вертикальной плоскости. Базисные функции ближе к центру, и особенно в правом нижнем углу представляют собой менее внятные наборы характеристик, но всё равно имеется тенденция - чем ближе базисная функция к правому нижнему углу квадрата, тем более высокие частоты в обоих плоскостях она содержит.

Как это дело считать? Очень просто - например, методом корреляций. Мы имеем всего 64 базисные функции, и самый простой, а заодно и практически самый быстрый, способ состоит в следующем - умножить каждый элемент исходного квадрата 8x8 на соответствующий элемент какой-либо базисной функции, затем сложить все 64 элемента произведения, поделить на какое-то число и записать частность в коэффициент к базисной функции, с которой мы комбинировали исходные данные. Так надо сделать 64 раза, чтобы получить все 64 коэффициента. А можно делать вышеописанным способом - зеркально двоить данные, затем производить FFT и выкидывать половину результата - нулевые синусы. В общем - как быстрее на данной аппаратуре, так и делают.

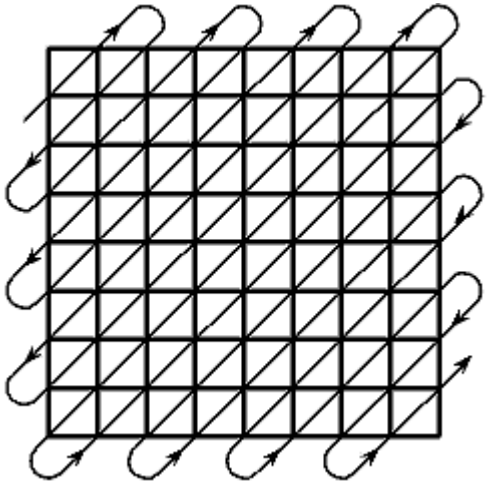
Почему выбрали DCT? Законный вопрос, почему было просто не взять тот же FFT? В основном, потому, что любое сжатие через разложение в базисы будет максимально эффективно, если использовать базисные функции, наиболее полно отвечающие наши сигналы. Преобразование Фурье, например, имеет только периодические функции, тогда как у DCT самая низкочастотная функция проходит только половину периодического цикла - а это очень частый элемент изображений. Также оказалось, что лучше иметь в два раза большее разрешение по частоте, чем фазовую информацию о сигналах. Для FFT, к примеру, мы не смогли бы сразу построить такую же изящную двухмерную модель результатов разложения - а DCT подошел тут как нельзя лучше.

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

Вообще говоря, и DCT был далеко не лучшим вариантом для сжатия, но компромисс с простотой привел именно к использованию DCT. Более сложные базисные функции работают намного удачнее - например, сжатие на основе Wavelet функций - формат файлов с расширение .wi - работает ощутимо лучше JPEG, но использовать их очень сложно и разложение в них приводит к долгой работе сжимателя, поэтому я видел этот формат сжатия только в Corel Photo-Paint. Есть преобразования, которые вообще слишком сложны для использования в потребительских приложениях, но дали бы просто огромный прирост сжатия - эксперименты с преобразованием Karhunen-Loeve дают сжатие в несколько раз лучше чем у JPEG (при том же качестве - размер файла в три раза меньше), но вычислять их с хоть сколь приемлемой скоростью еще не научились. Что это такое я, кстати, даже и не знаю. :)

Итак, подготовительные операции закончены. Мы преобразовали информацию в форму коэффициентов базисных функций. Пока никаких сжимающих операций мы не производили - если сложить обратно все базисные функции с соответствующими коэффициентами, получится в точности исходное изображение. Теперь же настало время уменьшать количество информации. Мы перешли в пространство базисных функций, где сделать это незаметным для человека образом будет гораздо легче.

Далее в основном используется тот факт, что в реальных изображениях высокочастотные составляющие встречаются редко - ну не всегда же есть сетка в один пиксель шириной? Правильнее будет сказать, что, наоборот, такое почти никогда не встречается. Поэтому кодировать высокочастотные компоненты можно более схематично. И тут нас еще раз выручает наше ненужное в общем то, но наглядное представление результатов DCT в двухмерном виде. Справа сверху изображена так называемая таблица квантования JPEG. Вернее, один из её вариантов - он соответствует слабому сжатию (то есть практически оригинальной картинке). Цифры на ней означают, что информацию, затрачиваемую на кодирование соответствующей базисной функции, надо понизить в **n** раз. Например, правая нижняя, 64-я базисная функция - целое шахматное поле из белых и черных клеток - практически никогда не встречается в изображении, поэтому её наличие (если уж оно всё таки произошло) можно характеризовать схематично.



Насколько сильно мы снижаем точность представления высоких частот разложения - зависит от желаемого качества сжатия. После того, как мы выбрали качество, синтезируется таблица квантования, на основании которой происходит дальнейшее кодирование. Например, единица в какой-либо клетке означает, что на кодирование этого коэффициента пойдет 8 бит, а цифра 16 - значит, точность в 16 раз меньше - уже всего 4 бита. Современные алгоритмы сжатия JPEG могут менять таблицу квантования в соответствии с реальной картинкой, делая её разной для разных частей изображения - это называется оптимизированный JPEG.
Дальнейшая судьба коэффициентов, уже закодированных с какой-либо точностью (каким-либо количеством бит) - быть построенными в непрерывный поток данных в соответствии со схемой слева. Такая система позволяет дальнейшему алгоритму сжатия действовать более эффективно - родственные группы частот в получающемся потоке данных следуют друг за другом. Это особенно важно при сильном сжатии, когда высокочастотные коэффициенты часто нулевые, и получается много нулей подряд - а по сути именно такое сжатие почти всегда применяется в реальных приложениях.

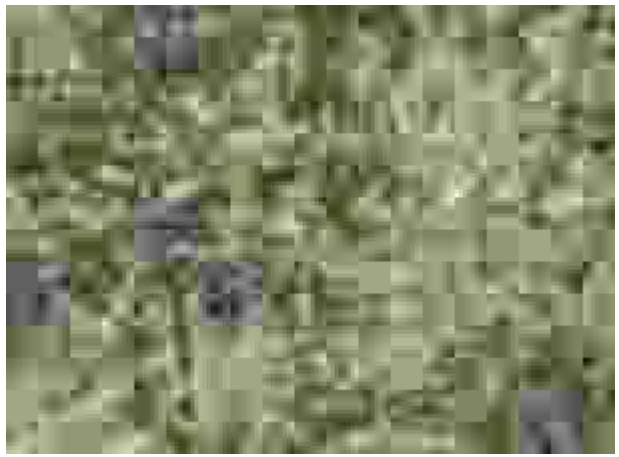
Подготовительные операции JPEG. В большинстве профессиональных программ есть два регулятора, отвечающих за качество сжатия JPEG - собственно сжатие и 'сглаживание' - smoothing. Параметр 'сглаживание' включает предварительную фильтрацию изображения с целью убрать высокие частоты. Идея в том, что если мы сделаем даже самую слабую фильтрацию, то базисных функций, которые, скажем, соответствуют цифре 8 и более на приведенной таблице квантования, в изображении не будет **вообще**. Таким образом после фильтрации в эти ячейки таблицы можно записать 255 - то есть не тратить на представление этих коэффициентов ни одного бита. Этот прием сильно увеличивает сжатие. После такой процедуры мы не только гарантированно не сможем нормально сжать мелкие сетки и особо резкие переходы, но и заранее отказываемся от их передачи, избавляя алгоритм от мучений и выдачи искаженных данных.

Заключительная операция - кодирование всего этого потока данных по Хаффману. То есть в общем то банальная архивация самым слабым алгоритмом.

Немного о цветном сжатии. В общем то идея такая: преобразовать изображение в пространство {яркость, цветовая информация}, затем сжать канал яркости как обычное черно-белое изображение по вышеописанному алгоритму. Цветовой канал переводятся в меньшее разрешение - обычно в два раза меньшее (это соответствует уменьшению информации в 4 раза просто на халяву), и сжимается затем практически так же, как и черно-белый канал. На обратном конце всё возвращается как было - цветное изображение синтезируется обратно.

Вот и вся история... Как видно, сжатие в JPEG устроено не так уж и сложно. Приношу извинения за немного перегруженную часть про двухмерное представление DCT - но для тех, кто хочет действительно хорошо в этом разобраться, надеюсь, оно того стоило.

Ну и в самом конце - демонстрационная картинка, которая позволяет всё увидеть. Это JPEG с очень плохого качества (с сильным сжатием). Можно увидеть и деление на квадраты 8x8, и использование характерных базисных функций - в каждом квадрате сохранялись лишь несколько из них - например, особо много квадратов лишь с функциями [2,1] и [1,2] - изменение интенсивности сверху-вниз или слева-направо, есть даже несколько квадратов, где сохранилась только [1,1] - константа, т.е. просто равномерная закраска.



Вот так. Разглядывайте. А благодарить за эту картинку надо Арсения Хахалина. Можно даже сходить к нему на страничку - http://arsenicum.go.to/. Она хорошая.

Распределение лекций:

№ лекции	Дата	где расположен материал
1	1.09.15	I файл стр. 1-10
2	1.09.15	I файл стр. 11-39
3	8.09.15	II файл стр. 1-13
4	8.09.15	II файл стр. 14-28
5	15.09.15	II файл стр. 29-32
6	28.09.15	II файл стр. 33-34
7	12.10.15	II файл стр. 35. III файл стр. 24-30., стр. 1-2.
8	26.10.15	III файл стр. 3-8
9	9.11.15	III файл стр. 9-16
10	23.11.15	III файл стр. 17-20
11	7.12.15	III файл стр. 21-23