ГЛОССАРИЙ

Критерий оценки избыточности k = I/(I+C), где I – информационные (informational) биты, а C – проверочные (checking)

ОСНОВНЫЕ ИДЕИ

10% ошибок в передаваемой информации – это довольно много и исправить возникшие ошибки в данном (или большем) количестве почти невозможно. Поэтому мы перемежаем передаваемую информацию таким образом, чтобы даже при поступлении данных, подвергнутых пачечным ошибкам, у нас не было более 10% ошибок (в идеале ошибки д.б. единичными).

Для всех примеров будет браться следующий исходный код (27 символов):
101 011 110 011 000 111 010 001 110

Не все коды могут работать с указанным кол-вом символов: в таком случае мы будем дополнять код до нужного числа различными способами (*подразумевается, что приёмная сторона знает про эту особенность и при/после декодирования учтёт это*).

БЛОЧНЫЕ КОДЫ

1) *Дополнение до чётности: трёхмерный вариант*

27 символов для кодирования:

101 011 110 - 011 000 111 - 010 001 110

Первые 4 матрицы дополнены до чётности по строкам и столбцам. Четвёртая матрица – это дополнение до чётности элементов «в глубину»

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 1 & 1 & 1 & \end{pmatrix}$$

Перепишем нашу последовательность в закодированном виде: сначала – исходные символы, после – дополнения (парами: по вертикали и горизонтали) и в конце – тройка дополнения «по глубине» (сверху вниз по строкам):

101 011 110 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

Декодирование следующее: снова дополняем до чётности. В строках и столбцах, где не ноль – возможна ошибка. По глубине – также *(это доп. параметр – если в каждой группе по одной ошибке, то это не нужно).*

Вносим ошибки и ДЕкодируем:

ВАРИАНТ 1: в каждой группе по ошибке

101 111 110 - 011 010 111 - 110 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix}1&0&1&0&0\\1&1&1&0&1\\1&1&0&0&0\\0&0&0&&\\1&0&0&&\end{pmatrix}\begin{pmatrix}0&1&1&0&0\\0&1&0&0&1\\1&1&1&1&0\\1&0&0&&\\0&1&0&&\end{pmatrix}\begin{pmatrix}1&1&0&1&1\\0&0&1&1&0\\1&1&0&0&0\\1&0&1&&\\1&0&0&&\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\1&1&1\end{pmatrix}\begin{pmatrix}1&0&0\\1&1&0\\0&0&0\end{pmatrix}$$

ВАРИАНТ 2: в одной группе две ошибки, в другой - одна

111 010 110 - 011 000 011 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix}1&\mathbf{1}&1&0&1\\0&1&\mathbf{0}&0&1\\1&1&0&0&0\\0&0&0&&\\0&1&1&&\end{pmatrix}\begin{pmatrix}0&1&1&0&0\\0&0&0&0&0\\\mathbf{0}&1&1&1&1\\1&0&0&&\\1&0&0&&\end{pmatrix}\begin{pmatrix}0&1&0&1&0\\0&0&1&1&0\\1&1&0&0&0\\1&0&1&&\\0&0&0&&\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\1&1&1\end{pmatrix}\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}$$

Каждый бит имеет номер, состоящий из 3 чисел: его координат по X (строка), Y (столбец), Z (слой). Начало отсчёта строк и столбцов – сверху-слева. Слои отсчитываются слева направо.

Красным отмечены подозрительные на нарушение биты:

1-ый слой: 1,2,1 | 1,3,1 | 2,2,1 | 2,3,1

2-ой слой: 3,1,2

По проверки чётности на глубину мы можем сказать, что ошибкам м.б. подвержены биты с X и Y, равными 1,2 | 2,3 | 3,1 . При сопоставлении со слоями 1 и 2 видно, что данному условию удовлетворяют биты: 1,2,1 | 2,3,1 | 3,1,2

ВАРИАНТ 3: три ошибки в одной группе

111 010 100 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix}1&1&1&0&1\\0&1&0&0&1\\1&0&0&0&1\\0&0&0&&\\0&0&1&&\end{pmatrix}\begin{pmatrix}0&1&1&0&0\\0&0&0&0&0\\1&1&1&1&0\\1&0&0&&\\0&0&0&&\end{pmatrix}\begin{pmatrix}0&1&0&1&0\\0&0&1&1&0\\1&1&0&0&0\\1&0&1&&\\0&0&0&&\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\1&1&1\end{pmatrix}\begin{pmatrix}0&1&0\\0&0&1\\0&1&0\end{pmatrix}$$

При декодировании видно, что группы 2 и 3 ошибок не содержат, а вот группа 1 – подвержена им крайне сильно. В таком случае мы просто смотрим на матрицу чётности по глубине – в местах, где стоят единицы, мы имеем ошибки в 1-ой группе.

Посмотрим этот алгоритм с ошибками в другой группе:

101 011 110 - 001 100 011 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix}1&0&1&0&0\\0&1&1&0&0\\1&1&0&0&0\\0&0&0&&\\0&0&0&&\end{pmatrix}\begin{pmatrix}0&0&1&0&1\\1&0&0&0&1\\0&1&1&1&1\\1&0&0&&\\0&1&0&&\end{pmatrix}\begin{pmatrix}0&1&0&1&0\\0&0&1&1&0\\1&1&0&0&0\\1&0&1&&\\0&0&0&&\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\1&1&1\end{pmatrix}\begin{pmatrix}0&1&0\\1&0&0\\1&0&0\end{pmatrix}$$

Как и ранее, мы видим, что ошибки возникают только во второй группе. И, как ранее, посмотрев на матрицу чётности по глубине мы обнаружим биты из второй группы, подверженные ошибке.

ВАРИАНТ 4: две ошибки в одной строке / одном столбце

110 011 110 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 1 & 1 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Как мы видим, чётность в строках не нарушена. Зато чётность по столбцам говорит, что во 2 и 3 столбцах есть ошибки. Сопоставив с матрицей «по глубине», мы их сразу найдём.

Аналогично – для ошибок по столбцу:

111 001 110 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

ВАРИАНТ 5: четыре ошибки: алгоритм ломается!

- В одной группе (попытка 1):

110 101 110 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 1 & 0 & 1 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Просто наложив матрицу чётности «по глубине», мы найдём ошибки.

- В одной группе (попытка 2):

101 011 110 - 111 101 110 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Просто наложив матрицу чётности «по глубине», мы найдём ошибки.

- В одной группе (попытка 3):

101 011 110 - 111 011 110 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & & \\ 1 & 1 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Просто наложив матрицу чётности «по глубине», мы найдём ошибки.

- В **разных** группах (по две – случай 2+1+1 не отличается от варианта 2; попытка 4):

001 011 100 - 111 001 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-111

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & & \\ 1 & 1 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 1 & 0 & 1 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Выпишем все подозреваемые на ошибку биты:

1-ый слой: 1,1,1 | 1,2,1 | 3,1,1 | 3,2,1

2-ой слой: 1,1,2 | 1,3,2 | 2,1,2 | 2,3,2

Слой по глубине говорит нам, что ошибки м.б. в битах с X,Y = 2,3 | 3,2

Это нам ничего не даст, т.к. ошибки в битах 1,1,1 | 3,2,1 | 1,1,2 | 2,3,2

ВАРИАНТ 6: ошибка в контрольном бите

101 011 110 - 011 000 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-110

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Проверка внутри слоёв не дала обнаружения ошибок. Далее, в зависимости от реализации алгоритма мы или игнорируем указатель на ошибку в матрице «по глубине», или продолжаем работу исходя из того, что **число и расположение сломанных бит неизвестно**.

ВАРИАНТ 7: ошибка в контрольном бите и информационном

101 011 110 - 011 010 111 - 010 001 110 – 000-000 – 001-100 – 110-101 – 100-010-110

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & & \\ 0 & 1 & 0 & & \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Проверка внутри слоёв дала ошибку во втором слое. Однако проверка матрицей по глубине сообщает, что есть неустановленная ошибка в другом месте.

Далее, в зависимости от реализации алгоритма мы или игнорируем второй указатель на ошибку в матрице «по глубине» (первый совпадает с проверкой внутри слоя), или продолжаем работу исходя из того, что **число и расположение сломанных бит неизвестно**.

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: Данный алгоритм способен обнаружить ошибки в коде и исправить их, если допущено не более 3 ошибок. При наличии ошибок в контрольных битах мы можем или отбросить их (на свой страх и риск), или запросить последовательность заново.

ПЛЮСЫ: простота алгоритма; исправляет до 3 ошибок в информационных битах.

МИНУСЫ: избыточность не спасает от ошибки в контрольном бите.

2) Линейный блочный код Хэмминга

Будем считать, что мы передаём информационные слова по четыре бита: 1010 – первое слово нашей последовательности, 1111 – второе, 0011 – третье и т.д.
Обозначим исходное слово как u; оно связано с закодированным словом C через порождающую матрицу как C = u * G;
Порождающая матрица G(k, n) и проверочная матрица Хэмминга H(n-k, n), где k – число информационных бит (у нас 4) и n – число бит кодового вектора (n=7)
К слову: они обе имеют в себе единичную матрицу!

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Как определить k и n: берём число-степень двойки (показатель степени больше 1) вычитаем из него 1; получившееся число – всего бит, информационных – получившееся число минус степень двойки. Т.е.: $n = 2^2-1 = 3 \to k = 3-1 = 2$ $n = 2^3-1 = 7 \to k = 7-3 = 4$ $n = 2^4-1 = 15 \to k = 15-4 = 11$ $n = 2^5-1 = 31 \to k = 31-5 = 26$ $n = 2^6-1 = 63 \to k = 63-6 = 57$ … ***Мы берём (4,7), т.к. это наиболее оптимальный вариант.***

Кодируем:

$$C = u*G = (1\ 0\ 1\ 0) * \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} = |\text{перемножение – AND, сложение – XOR; всё остальное – как в линейной алгебре}| = (1\ 1\ 0\ 1\ 0\ 1\ 0)$$

Последние 4 символа совпадают с информационными! Первые 3 символа – проверочные. Их можно сосчитать через информационные, сложив соответствующие биты по модуль 2. Обозначив за B передаваемые 7 бит, и помня, что 0-2 – это контрольные, а 3-6 – информационные, используя матрицу G мы получим следующие уравнения для контрольных бит: B0 = B3+B4+B6 B1 = B4+B5+B6 B2 = B3+B5+B6 Т.е. каждое уравнение под контрольный бит t строится по столбцу t матрицы G (исключая столбцы единичной матрицы), где 1 в строке r означает, что для получения указанного контрольного бита t мы должны сложить информационный бит под номером r+2 (1-ая строка - 3-ий–бит, 4-ая строка – 6-ой бит и т.п.) с другими инф. битами, которым соответствует 1 в строке.

Допустим, что при передаче ошибок не было.

Декодируем с помощью транспонированной матрицы Хэмминга; D – проверочное уравнение (указывает на разряд одиночной ошибки).

По факту: код, соответствующий ошибочному разряду – это содержимое строки транспонированной матрицы Хэмминга:

Код ошибки	000	100	010	001	101	110	011	111
№ разряда	-	0	1	2	3	4	5	6

$$D = C*H^T = (1\ 1\ 0\ 1\ 0\ 1\ 0) * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = (0\ 0\ 0) \text{ – ошибок нет.}$$

Если допущена ошибка в 3-ем разряде:

$$D = C*H^T = (1\ 1\ 0\ \mathbf{0}\ 0\ 1\ 0) * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = (1\ 0\ 1) \text{ – код ошибки в третьем разряде.}$$

Другой способ декодирования – с использованием контрольных бит. Если есть ошибка, то ни одно из равенств для B0-2 не будет выполняться. В таком случае суммы контрольных бит с составляющими его информационными дадут нам код, показывающий позицию ошибки.
Т.е. если мы получили 110**0**010:
B0 = B3+B4+B6 = **0** + 0 + 0 = 0 – а д.б. равно единице. Равенство НЕ выполняется!
B1 = B4+B5+B6 = 0 + 1 + 0 = 1
B2 = B3+B5+B6 = **0** + 1 + 0 = 1 – а д.б. равно нулю. Равенство НЕ выполняется!
Ищем место ошибки:
Err0 = B0 + B3+B4+B6 = 1 + 0 = 1
Err1 = B1 + B4+B5+B6 = 1 + 1 = 0
Err2 = B2 + B3+B5+B6 = 0 + 1 = 1
Код ошибки: Err2Err1Err0 = 101 – по таблице это разряд номер три.

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: Данный алгоритм способен обнаружить одну ошибку в коде и исправить её. Ошибка в контрольном бите при этом не фатальна.

ПЛЮСЫ: невосприимчив к ошибке в контрольном бите, можно составить свою таблицу кодирования/декодирования.

МИНУСЫ: не самый простой алгоритм из-за перемножения матриц (в общем случае).

3) Реализация кода Хэмминга, предложенная нам на телекоммуникациях (от 22.05.2015) – частный случай.

Как и ранее, используем по 4 бита: первое слово – 1010, и т.д.
В чём суть: из 7 передаваемых бит B(1 .. 7) биты, номер которых степень двойки (1,2,4) – не информационные, а контрольные (обозначим их как С). Они определяются так:

1. Сначала заполним позиции, не являющиеся степенью 2:
   __1_010
2. Теперь можем сосчитать контрольные биты:
   C1 = B3+B5+B7
   C2 = B3+B6+B7
   C4 = B5+B6+B7
3. Результат: 1011010

Если при передаче не было ошибок, то код S, указывающий на ошибку, будет равен нулю.

Представим, что была допущена ошибка в 5-ом бите: 1011**1**10

S1 = C1 + B3+B5+B7 = 1 + 1+1+0 = 1
S2 = C2 + B3+B6+B7 = 0 + 1+1+0 = 0
S4 = C4 + B5+B6+B7 = 1 + 1 +1 0 = 1
S = S4S2S1 = 101 – ошибка в пятом разряде.

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: алгоритм аналогичен указанному в п.2 и его можно считать частным случаем – порождающая матрица здесь подобрана таким образом, что получив код ошибки мы можем не смотреть в таблицу соответствий, а сразу сказать, в каком разряде ошибка. Это значительно повышает простоту алгоритма, т.к. не требуется перемножать матрицы и лезть в таблицу соответствия между кодом ошибки и позицией в принятой последовательности.

4) Циклические коды

Рассмотрим код (4,7), где 4 – число инф. бит, а 7 – число бит для передачи.

Порождающий многочлен выглядит как $g(x) = x^3+x+1$

Первый передаваемый блок информации: 1010. Ему соответствует многочлен $d(x) = x^3+x$

Для нахождения кодового слова перемножим g(x) на d(x):

$c(x) = x^6+x^4+x^4+x^2+x^3+x = x^6+x^3+x^2+x$ (т.е. 1001110)

Если код принят без ошибок, то при делении c(x) на g(x) мы получим закодированное слово d(x):

$x^6+x^3+x^2+x$ | x^3+x+1
$\underline{x^6+x^4+x^3}$ x^3+x
x^4+x^2+x
$\underline{x^4+x^2+x}$
0

Общий алгоритм обнаружения и исправления ошибок:

1) Принятая комбинация делится на образующий многочлен g(x). Если остаток R(x)!=0, то определяется вес остатка w. Если вес остатка равен или меньше числа исправляемых ошибок t (w<=t), то принятую комбинацию складывают по модулю 2 с остатком и получают исправленную комбинацию.
2) Если w>t, то производится циклический сдвиг на один символ влево и полученная после такого сдвига комбинация снова делится на образующий многочлен. Если вес полученного остатка w<=t, то циклически сдвинутую комбинацию складывают с остатком и затем после

сложения циклически сдвигают в обратную сторону вправо на один символ (возвращают на прежнее место). В результате получаем исправленную комбинацию.

3) Если после циклического сдвига на один символ по-прежнему w>t, то производят дополнительные циклические сдвиги влево. При этом после каждого сдвига осуществляется деление сдвинутой комбинации на g(x) и проверяется вес остатка. При w<=t сдвинутую комбинацию складывают с остатком и производят столько обратных циклических сдвигов вправо, сколько было сделано влево.

Допустим, что у нас **ошибка в 1-ом** разряде: $x^6+x^3+x^2$ (т.е. 1001100, верно 1001110)

$x^6+x^3+x^2$ | x^3+x+1

$\underline{x^6+x^4+x^3}$ $\quad x^3+x$

x^4+x^2

$\underline{x^4+x^2+x}$

x

Складываем принятую комбинацию по модуль 2 с остатком и получаем верную комбинацию: $x^6+x^3+x^2+x$

- Допустим, что у нас **ошибка в 5-ом** разряде: $x^6+x^5+x^3+x^2+x$ (т.е. 1101110, верно 1001110)

$x^6+x^5+x^3+x^2+x$ | x^3+x+1

$\underline{x^6+x^4+x^3}$ $\quad x^3+x^2+x+1$

$x^5+x^4+x^2$

$\underline{x^5+x^3+x^2}$

x^4+x^3+x

$\underline{x^4+x^2+x}$

x^3+x^2

$\underline{x^3+x+1}$

x^2+x+1

Делаем первый циклический сдвиг влево (1101110 –> 1011101):

$x^6+x^4+x^3+x^2+1$ | x^3+x+1

$\underline{x^6+x^4+x^3}$ $\quad x^3$

$0+x^2+1$

Делаем второй циклический сдвиг влево (1011101–> 0111011):

$x^5+x^4+x^3+x+1$ | x^3+x+1

$\underline{x^5+x^3+x^2}$ $\quad x^2+x$

x^4+x^2+x

$\underline{x^4+x^2+x}$

$0+1$

Складываем с остатком:

0111011 + 1 = 0111010

Теперь делаем циклические сдвиги вправо (2 раза): 0111010 -> 0011101 -> 1001110

Ошибка исправлена!

- Допустим, что у нас **ошибка в 1-ом и 2-ом** разрядах: x^6+x^3 (т.е. 1001**00**0, верно 1001110)

$$\begin{array}{l|l} x^6+x^3 & x^3+x+1 \\ \underline{x^6+x^4+x^3} & x^3+x \\ \quad x^4 & \\ \quad \underline{x^4+x^2+x} & \\ \qquad x^2+x & \end{array}$$

Складываем принятую комбинацию по модуль 2 с остатком и получаем верную комбинацию: $x^6+x^3+x^2+x$

- Допустим, что у нас **ошибка в 1-ом и 3-ем** разрядах: x^6+x^2 (т.е. 100**0**1**0**0, верно 1001110)

$$\begin{array}{l|l} x^6+x^2 & x^3+x+1 \\ \underline{x^6+x^4+x^3} & x^3+x \\ \quad x^4+x^3+x^2 & \\ \quad \underline{x^4+x^2+x} & \\ \qquad x^3+x & \end{array}$$

Складываем принятую комбинацию по модуль 2 с остатком и получаем верную комбинацию: $x^6+x^3+x^2+x$

- Допустим, что у нас **ошибка в 1-ом и 4-ом** разрядах: $x^6+x^4+x^3+x^2$ (т.е. 10**1**11**0**0, верно 1001110)

$$\begin{array}{l|l} x^6+x^4+x^3+x^2 & x^3+x+1 \\ \underline{x^6+x^4+x^3} & x^3 \\ \qquad 0+x^2 & \end{array}$$

Складываем принятую комбинацию по модуль 2 с остатком и получаем НЕверную комбинацию: $x^6+x^4+x^3$

- Допустим, что у нас **ошибка во 2-ом и 4-ом** разрядах: $x^6+x^4+x^3+x$ (т.е. 10**1**1**0**10, верно 1001110)

$$\begin{array}{l|l} x^6+x^4+x^3+x & x^3+x+1 \\ \underline{x^6+x^4+x^3} & x^3 \\ \qquad 0+x & \end{array}$$

Складываем принятую комбинацию по модуль 2 с остатком и получаем НЕверную комбинацию: $x^6+x^4+x^3+x^2$

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: Данный алгоритм способен обнаружить одну ошибку в коде и исправить её.

ПЛЮСЫ: можно составить свою таблицу кодирования.

МИНУСЫ: алгоритм не самый ресурсоёмкий и не самый сложный, однако требует многократные операции сложения по модуль два и циклические сдвиги. Если ошибок несколько, то они НЕ будут обнаружены.

СВЁРТОЧНЫЕ КОДЫ

Основная идея – кодируются не блоки информации, а сразу вся последовательность.

1) ***Контрольное суммирование***

Это простейший случай, когда между информационными битами вставляются проверочные (они представляют собой сумму по модуль два соседних бит).

27 символов для кодирования:

1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 0

Первый проверочный бит – это сумма первого и второго информационного по модуль два: 1 + 0 = 1

Второй проверочный бит – это сумма по модуль два второго и третьего информационных бит:

0 + 1 = 1

Ниже первая последовательность – информационные биты; вторая – проверочные:

1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 0
 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1

В итоге мы получим следующий код (синий – проверочные биты):
1 **1** 0 **1** 1 **1** 0 **1** 1 **0** 1 **0** 1 **0** 1 **1** 0 **0** 0 **1** 1 **0** 1 **1** 0 **0** 0 **0** 0 **1** 1 **0** 1 **0** 1 **1** 0 **1** 1 **1** 0 **0** 0 **0** 0 **1** 1 **0** 1 **0** 1 **1** 0

Декодирование следующее: суммируются соседние биты исходных данных и сравниваются с их проверочным битом. Если для двух соседних проверочных битов была зафиксирована ошибка, то общий информационный бит для этих двух проверочных битов - неверен. Для исправления ошибки необходимо заменить его на противоположный.
Если для одного **проверочного** символа была зафиксирована ошибка, а два соседних проверочных символа ошибку не показали, это означает, что сбой произошел в проверочном символе, а информационные биты корректны.
Рассмотрим **простой** пример (далее для удобства изложения И1 – информационный бит-1, П3 – проверочный бит-3 и т.п.)**:**

1 **1** 0 **1** 0 **1** 0 **1** 1 **0** 1 …

Мы видим, что П2 и П3 не получаются, если сделать И2+И3 и И3+И4 – на месте П2 и п3 д.б. ноль. Это значит, что ошибка в И3.

1 **1** 0 **0** 1 **1** 0 **1** 1 **0** 1 …

Мы видим, что И2+И3 не дают нужный П2. Однако И1+И2 и И3+И4 дают нам верные П1 и П3 соответственно. Значит, ошибка была в П2.

При проведении анализа принятых данных, если И-биты не совпадают с П-битом, то мы берём и анализируем следующую пару бит.

Очевидное слабое место данного кода – это ошибка в крайних И/П битах (верно определить сломанный бит – невозможно! Можно только выяснить, что есть ошибка). **Существует способ решения проблемы крайних бит**: добавлять в начало и конец последовательности *дополнительные* биты, значение которых нам всегда известно: скажем, 1 или 0. Независимо от принятых данных, мы будем декодировать код исходя из того, что крайние биты – это наши константы. После декодирования *дополнительные* биты извлекаются и у нас остаются только информационные.

Наш код принимает следующий вид (дополнительные биты – это «1 0 1», т.е. «$И_{-1}$ $П_{-1}$ И0», а также «ПN $И_{N+1}$ $П_{N+1}$ $И_{N+2}$», выделены зелёным цветом; жёлтым цветом выделены П0 и ПN– образуются за счёт И0+И1 и ИN+ $И_{N+1}$):

1 0 1 0
1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0
1 1 0 1

Общий алгоритма декодирования остаётся тем же: сначала проверяют $И_n+И_{n+1}=П_n$. Если совпадения нет – смотрят следующую пару: $И_{n+1}+И_{n+2}=П_{n+1}$. Совпадения **нет**? Ошибка в $И_{n+1}$. Совпадение **есть**? Проверяют предыдущую пару $И_{n-1}+И_n=П_{n-1}$ Совпадение есть? Значит, ошибка в $И_n$

ВАРИАНТ 1: сломан крайний слева И-бит

1 0 1 0 0 1 0 1 1 0 1 1 0 1 0 1 0 1 …

И0+И1 = 1 – не совпадает с П0. Проверяем следующую пару: И1+И2 = 0 – НЕ совпадает с П1. ***Т.к. нет совпадения между рядом стоящими П-битами, то ошибка в их общем И-бите.*** Значит, ошибка в И1!

ВАРИАНТ 2: сломан нулевой П-бит

1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 …

И0+И1 = 0 – не совпадает с П0. Проверяем следующую пару: И1+И2 = 1 – совпадает с П1. Значит, надо проверить и предыдущую пару: $И_{-1}$+И0 = 0 – совпадение с $П_{-1}$ есть ВСЕГДА. ***Т.к. проверка смежных проверочных бит прошла успешно, то ошибка в проверочном бите.*** Значит, ошибка в П0.

ВАРИАНТ 3: сломан первый П-бит

1 0 1 0 1 0 0 1 1 1 0 1 1 …
И0+И1 = 0 – совпадает с П0
И1+И2 = 1 – не совпадает с П1; проверяем следующую пару И-бит: И2+И3 = 1 – совпадение с П2 есть. Проверка предыдущей пары уже была шагом ранее – совпадение было. ***Т.к. проверка смежных проверочных бит прошла успешно, то ошибка в проверочном бите.*** Значит, ошибка в П1.

ВАРИАНТ 4: сломаны 2 произвольных бита: алгоритм ломается (на расстоянии 1)

- Расстояние между битами – 4 (попытка 1):

1 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 0 1 …
И1+И2 = 0 – нет совпадения с П1. Проверяем сл. пару: И2+И3 = 0 – нет совпадения с П2. Значит, ошибка в И2! Заменяем на верный бит:
1 0 1 0 1 1 0 1 1 1 0 0 1 0 1 0 1 0 1 …
…
И4+И5 = 1 – не совпадает с П4. Проверяем следующую пару: И5+И6 = 0 – совпадение с П5 есть. Предыдущая пара: И3+И4 = 1 – совпадение с П3 есть. Значит, ошибка в П4!
1 0 1 0 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 …

- Расстояние между позициями бит – 1, И-биты (попытка 2):

1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1 …

И1+И2 = 0 – нет совпадения с П1. Проверяем сл. пару: И2+И3 = 1 – совпадение с П2 есть. Проверяем предыдущую пару: И0+И1 = 0 – совпадение с П0 есть. Значит, ошибка в П1. *Но П1 верное!* Меняем «сломанный бит» и идём дальше:

1 **0** 1 **0** 1 **0** 1 **1** 0 **1** 0 **1** 1 **0** 1 **0** 1 **0** 1 …

И2+И3 = 1 – совпадение с П2 есть.

И3+И4 = 0 – совпадения с П3 нет.

Проверяем следующую пару: И4+И5 = 1 – совпадение с П4 есть. Предыдущая пара даёт совпадение. Значит, ошибка в П3. *Но П3 верное!* Меняем сломанный бит и идём дальше. **Далее ошибку не обнаружить.**

- Расстояние между позициями бит – 1, П-биты (попытка 3):

1 **0** 1 **0** 1 **1** 0 **0** 1 **0** 0 **1** 1 **0** 1 **0** 1 **0** 1 …

И2+И3 = 1 – совпадения с П2 нет. Проверяем сл. пару: И3+И4 = 1 – совпадения с П3 нет. Значит, ошибка в И3. *Но И3 верное!* Меняем сломанный бит и идём дальше. **Далее ошибку не обнаружить.**

- Расстояние между позициями бит – 2 (попытка 4):

1 **0** 1 **0** 1 **1** 1 **1** 1 **0** 0 **1** 1 **0** 1 **0** 1 **0** 1 …

И1+И2 = 0 – не совпадает с П1. Проверяем сл. пару бит: И2+И3 = 0 – нет совпадения с П2. Значит, ошибка в И2.

1 **0** 1 **0** 1 **1** 0 **1** 1 **0** 0 **1** 1 **0** 1 **0** 1 **0** 1 …

…

И3+И4 = 1 – нет совпадения с П3. Проверяем сл. пару: И4+И5 = 1 – совпадение с П4 есть. Смотрим предыдущую пару: И2+И3 = 1 – совпадение с П2 есть. Значит, ошибка в П3. *Всё верно.*

ВАРИАНТ 5: сломаны 2 рядом стоящих бита: алгоритм ломается!

- Первый сломанный бит - проверочный (попытка 1):

1 **0** 1 **1** 0 **1** 0 **1** 1 **1** 0 **1** 1 **0** 1 **0** 1 **0** …

И0+И1 = 1 – совпадение с П0 есть

И1+И2 = 0 – совпадения с П1 нет; проверяем следующую пару бит: И2+И3 = 1 – совпадает с П2. Предыдущая пара дала совпадение с П0. Значит, ошибка в П1. *Однако у нас ошибка в П0 и И0!* Заменяем сломанный (по нашему мнению) бит:

1 **0** 1 **1** 0 **0** 0 **1** 1 **1** 0 **1** 1 **0** 1 **0** 1 **0** …

Далее ошибку не обнаружить.

- Первый сломанный бит - информационный (попытка 2):

1 **0** 1 **0** 1 **1** 0 **1** 1 **1** 1 **0** 1 **0** 1 **0** 1 **0** …

И3+И4 = 0 – нет совпадения с П3. Проверяем следующую пару бит: И4+И5 = 0 – совпадение с П4 есть. Предыдущая пара бит: И2+И3 = 1 – совпадение с П2 есть. Значит, ошибка в П3 (*но оно верно!*). Заменяем сломанный (по нашему мнению) бит:

1 **0** 1 **0** 1 **1** 0 **1** 1 **0** 1 **0** 1 **0** 1 **0** 1 **0** …

Далее ошибку не обнаружить.

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: Данный алгоритм способен обнаружить ошибки в коде и исправить их, если расстояние между позициями нарушенных бит не меньше 2 (иначе исправляется верный бит и в итоге мы имеем 3 ошибки рядом вместо двух).

ПЛЮСЫ: простота алгоритма; теоретически может исправить ~25% ошибок, если расстояние между позициями сломанных бит будет не меньше 2. Ошибки в контрольных битах м.б. исправлены.

МИНУСЫ: если расстояние между позициями сломанных бит меньше 2 мы не можем это исправить.

2) Свёрточный код с декодером Витерби

Это рассматривается в бакалаврской моего коллеги, поэтому не будем забирать у него хлеб и просто расскажем общую идею.

Кодовое слово получается с помощью n полиномов (число n больше числа входных последовательностей – есть коды 1/2, 2/3, 5/7 и т.п., где первое число – это число битов на входе, а второе – кол-во битов на выходе), которые используя линию задержки складывают по модуль 2 биты в ячейках полинома. Принимающая сторона каждое принятое слово оценивает по расстоянию Хэмминга между битами и переводит конечный автомат декодера в то или иное состояние. После, когда прямой путь переходов построен (для всех возможных переходов) начинается обратный путь: мы идём по наименее тяжёлым дугам и в обратном порядке восстанавливаем закодированную последовательность.

ВЫВОДЫ ПО АЛГОРИТМУ КОДИРОВАНИЯ: Данный алгоритм способен обнаружить ошибки в коде и исправить их, однако реализация его ДЕкодера является крайне сложной. С учётом этого порой проще использовать более простые коды и передавать речь ещё раз, если после декодирования речь собеседника невнятна.

ПЛЮСЫ: простота алгоритма кодирования. При правильном алгоритме ДЕкодирования можно исправлять даже рядом стоящие ошибки.

МИНУСЫ: крайне сложный алгоритм ДЕкодирования. Чем больше степень полинома у кодера, тем больше будет состояний у конечного автомата, который расставляет веса при декодировании. Сам по себе алгоритм декодирования является крайне ресурсоёмким и на определённом этапе включает в себя повторное кодирование (чтобы сравнить полученную на входе и после декодирования последовательности).