**Код Хэмминга**

**Ко́ды Хэ́мминга** — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся [кодов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4). Построены применительно к [двоичной системе счисления](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Позволяет исправлять одиночную ошибку (ошибка в одном бите) и находить двойную.

Названы в честь американского математика [Ричарда Хэмминга](https://ru.wikipedia.org/wiki/%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3,_%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4_%D0%A3%D1%8D%D1%81%D0%BB%D0%B8), предложившего их.

**Самоконтролирующиеся коды**

Коды Хэмминга являются самоконтролирующимися кодами, то есть кодами, позволяющими автоматически обнаруживать ошибки при передаче данных. Для их построения достаточно приписать к каждому слову один добавочный (контрольный) двоичный разряд и выбрать цифру этого разряда так, чтобы общее количество единиц в изображении любого числа было, например, нечетным. Одиночная ошибка в каком-либо разряде передаваемого слова (в том числе, может быть, и в контрольном разряде) изменит четность общего количества единиц. Счетчики по модулю 2, подсчитывающие количество единиц, которые содержатся среди двоичных цифр числа, могут давать сигнал о наличии ошибок.

При этом невозможно узнать, в каком именно разряде произошла ошибка, и, следовательно, нет возможности исправить её. Остаются незамеченными также ошибки, возникающие одновременно в двух, четырёх, и т. д. — в четном количестве разрядов. Впрочем, двойные, а тем более четырёхкратные ошибки полагаются маловероятными.

**Немного о коде Хэмминга (7; 4).**

### Пример на круге

Пусть дана последовательность из 4-х битов, например 1011.

Нарисуем три кружочка, так, как изображено на картинке.

0

1

1

У нас получается 7 областей трех типов.

1) В 3-х из них есть точки только одного круга.

2) В других 3-х есть точки сразу 2-х кругов.

3) И в последней есть точки всех кругов.

Поместим биты в области 2-го и 3-го типа.

Области 1 типа будем заполнять следующим образом: берем все остальные области, в которые входит данный круг, и суммируем значения из них.

0

1

1

Теперь для каждой области справедливо следующее: значение в ней равно сумме значений из остальных областей круга, в который входит данная область. Соответственно, если у нас при передаче данных значение 1-го любого бита будет передано неправильно, мы сможем обнаружить ошибку и восстановить правильное значение.

### Пример применения алгоритма Хемминга.

Пусть дана последовательность бит 1110110. Пронумеруем все биты.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |

Перепишем следующим образом

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 1 | 1 | \* | 0 | 1 | 1 | \* | 0 | \* | \* |

Т. е. пометим биты с номерами, являющимися степенью 2, как зарезервированные. После чего перепишем последовательность с учетом этого. Далее в зарезервированных битах будет сохранена контрольная сумма.

Перепишем номера битов, которые имеют значение 1 в столбик в двоичном виде и просуммируем номера битов по модулю 2.

|  |  |
| --- | --- |
| 5 | 0101 |
| 6 | 0110 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| + | 1011 |

Значение, которое получилось в качестве суммы и является контрольной суммой. Запишем его на место контрольных битов.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

Если теперь мы получаем последовательность битов, мы можем ее проверить, для этого повторим указанную процедуру, т. е. просуммируем номера битов, у которых значение 1.

|  |  |
| --- | --- |
| 1 | 0001 |
| 2 | 0010 |
| 5 | 0101 |
| 6 | 0110 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| + | 0000 |

О чудо, получилось 0. Это означает, что все правильно.

Допустим, что мы получили последовательность, у которой 1 бит неправильный. Допустим изменился 9-й бит.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

|  |  |
| --- | --- |
| 1 | 0001 |
| 2 | 0010 |
| 5 | 0101 |
| 6 | 0110 |
| 8 | 1000 |
| 10 | 1010 |
| 11 | 1011 |
| + | 1001 |

В результате суммы получилось число 9. Как мы видим, это число совпадает с номером измененного бита.

Очевидно, что данный подход можно применять к битовой последовательности любой длины. Можно также разбить последовательность на блоки фиксированного размера и обрабатывать каждый блок отдельно.

**Другой пример.**

Предположим, что нужно сгенерировать код Хемминга для некоторого информационного кодового слова. В качестве примера возьмём 15-битовое кодовое слово **x**1…**x**15, хотя алгоритм пригоден для кодовых слов любой длины. В приведённой ниже таблице в первой строке даны номера позиций в кодовом слове, во второй — условное обозначение битов, в третьей — значения битов.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** | **x9** | **x10** | **x11** | **x12** | **x13** | **x14** | **x15** |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Вставим в информационное слово контрольные биты **r**0…**r**4 таким образом, чтобы номера их позиций представляли собой целые степени двойки: 1, 2, 4, 8, 16… Получим 20-разрядное слово с 15 информационными и 5 контрольными битами. Первоначально контрольные биты устанавливаем равными нулю. На рисунке контрольные биты выделены розовым цветом.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **r0** | **r1** | **x1** | **r2** | **x2** | **x3** | **x4** | **r3** | **x5** | **x6** | **x7** | **x8** | **x9** | **x10** | **x11** | **r4** | **x12** | **x13** | **x14** | **x15** |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

В общем случае количество контрольных бит в кодовом слове равно двоичному логарифму числа, на единицу большего, чем количество бит кодового слова (включая контрольные биты); логарифм округляется в большую сторону. Например, информационное слово длиной 1 бит требует двух контрольных разрядов, 2-, 3- или 4-битовое информационное слово — трёх, 5…11-битовое — четырёх, 12…26-битовое — пяти и т. д.

Добавим к таблице 5 строк (по количеству контрольных битов), в которые поместим матрицу преобразования. Каждая строка будет соответствовать одному контрольному биту (нулевой контрольный бит — верхняя строка, четвёртый — нижняя), каждый столбец — одному биту кодируемого слова. В каждом столбце матрицы преобразования поместим двоичный номер этого столбца, причём порядок следования битов будет обратный — младший бит расположим в верхней строке, старший — в нижней. Например, в третьем столбце матрицы будут стоять числа 11000, что соответствует двоичной записи числа три: 00011.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **r0** | **r1** | **x1** | **r2** | **x2** | **x3** | **x4** | **r3** | **x5** | **x6** | **x7** | **x8** | **x9** | **x10** | **x11** | **r4** | **x12** | **x13** | **x14** | **x15** |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | **r**0 |  |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | **r**1 |  |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | **r**2 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | **r**3 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | **r**4 |  |

В правой части таблицы мы оставили пустым один столбец, в который поместим результаты вычислений контрольных битов. Вычисление контрольных битов производим следующим образом. Берём одну из строк матрицы преобразования (например, r0) и находим её скалярное произведение с кодовым словом, то есть перемножаем соответствующие биты обеих строк и находим сумму произведений. Если произведение получилось больше единицы, находим остаток от его деления на 2. Иными словами, мы подсчитываем сколько раз в кодовом слове и соответствующей строке матрицы в одинаковых позициях стоят единицы и берём это число по модулю 2.

Если описывать этот процесс в терминах матричной алгебры, то операция представляет собой перемножение матрицы преобразования на матрицу-столбец кодового слова, в результате чего получается матрица-столбец контрольных разрядов, которые нужно взять по модулю 2.

Например, для строки r0:

r0 = (1·0+0·0+1·1+0·0+1·0+0·0+1·1+0·0+1·0+0·0+1·1+0·0+1·1+0·1+1·1+0·0+1·0+0·0+1·0+0·1) mod 2 = 5 mod 2 = 1.

Полученные контрольные биты вставляем в кодовое слово вместо стоявших там ранее нулей. По аналогии находим проверочные биты в остальных строках. Кодирование по Хэммингу завершено. Полученное кодовое слово — 11110010001011110001.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **r0** | **r1** | **x1** | **r2** | **x2** | **x3** | **x4** | **r3** | **x5** | **x6** | **x7** | **x8** | **x9** | **x10** | **x11** | **r4** | **x12** | **x13** | **x14** | **x15** |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | **r**0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | **r**1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | **r**2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | **r**3 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | **r**4 | 1 |

**Алгоритм декодирования.**

Алгоритм декодирования по Хэммингу абсолютно идентичен алгоритму кодирования. Матрица преобразования соответствующей размерности умножается на матрицу-столбец кодового слова и каждый элемент полученной матрицы-столбца берётся по модулю 2. Полученная матрица-столбец получила название «матрица синдромов». Легко проверить, что кодовое слово, сформированное в соответствии с алгоритмом, описанным в предыдущем разделе, всегда даёт нулевую матрицу синдромов.

Матрица синдромов становится ненулевой, если в результате ошибки (например, при передаче слова по линии связи с шумами) один из битов исходного слова изменил своё значение. Предположим для примера, что в кодовом слове, полученном в предыдущем разделе, шестой бит изменил своё значение с нуля на единицу (на рисунке обозначено красным цветом). Тогда получим следующую матрицу синдромов.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **r0** | **r1** | **x1** | **r2** | **x2** | **x3** | **x4** | **r3** | **x5** | **x6** | **x7** | **x8** | **x9** | **x10** | **x11** | **r4** | **x12** | **x13** | **x14** | **x15** |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | **s**0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | **s**1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | **s**2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | **s**3 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | **s**4 | 0 |

Заметим, что при однократной ошибке матрица синдромов всегда представляет собой двоичную запись (младший разряд в верхней строке) номера позиции, в которой произошла ошибка. В приведённом примере матрица синдромов (01100) соответствует двоичному числу 00110 или десятичному 6, откуда следует, что ошибка произошла в шестом бите.

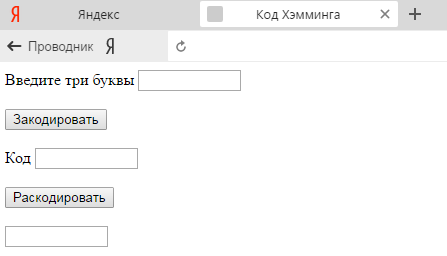
**Постановка задачи**

Закодируйте при помощи Кода Хэмминга последовательность из трех букв.

Необходимо представить строку в бинарном виде. Например:  
  
Длина строки из нулей и единиц, которые будете кодировать, равна 24.

Исправив возможные одиночные ошибки, декодировать последовательность.

Сделайте в виде:



Объяснения.

Напишем сценарий, который будет рассчитывать площадь прямоугольника по введенным пользователем длине и ширине. Для этого сначала разместим на html-странице нужные элементы формы:

<html>

<head>

<title>Расчет площади прямоугольника</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma1">

Введите длину прямоугольника <input type="text" name="t1" size="10"><br><br>

Введите ширину прямоугольника <input type="text" name="t2" size="10"><br><br>

<input type="button" name="button" value="Вычислить"><br><br>

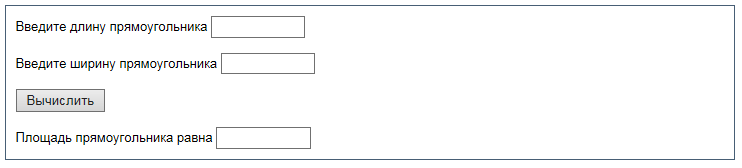
Площадь прямоугольника равна <input type="text" name="res" size="10">

</form>

</body>

</html>

В браузере наша страница будет выглядеть так:

Если вы забыли, как задаются элементы форм, то обратитесь к [HTML - работа с формами](http://www.site-do.ru/html/html11.php).  
  
Итак, пользователь вводит значения ширины и длины и нажимает на кнопку "Вычислить". После чего, в поле площадь должен появиться результат. Таким образом, событие наступает при нажатии на кнопку "Вычислить", значит именно к ней мы и привяжем обработчик события. Функцию вычисления площади назовем "areaRectangle":

.......

<input type="button" name="button" value="Вычислить"

onClick="areaRectangle();"><br><br>

.......

Теперь пришло время написать саму функцию "areaRectangle". Для этого откроем страницу script.js и напишем заготовку для функции:

function areaRectangle(){

}

Теперь надо написать тело функции. Для начала объявим три переменные: *a* - значение длины прямоугольника, *b* - значение ширины прямоугольника, *s* - площадь прямоугольника:

function areaRectangle(){

var a;

var b;

var s;

}

Значение (value) *a* должно браться из текущей страницы (document), из формы с именем "forma1", из текстового поля с именем "t1". Так это и записывается *document.forma1.t1.value*, т.е. перечисляются через точку имена объектов от родительского до нужного. Последним указывается необходимое свойство объекта (value).  
Аналогично и для значения *b* - *document.forma1.t2.value*.  
А наша переменная *s* - есть произведение *a* на *b*. Запишем это в тело функции:

function areaRectangle(){

var a=document.forma1.t1.value;

var b=document.forma1.t2.value;

var s=a\*b;

}

Осталось только написать инструкцию записи вычисленной площади в текстовое поле с именем "res" нашей формы. Т.е нам надо, чтобы в текщую страницу, в форму с именем "forma1", в текстовое поле с именем "res", в качестве значения (value) было присвоено значение *s*. Так и запишем:

function areaRectangle(){

var a=document.forma1.t1.value;

var b=document.forma1.t2.value;

var s=a\*b;

document.forma1.res.value=s;

}

Иными словами, мы сначала присвоили нашим переменным *a* и *b* значения из формы, затем произвели необходимые расчеты, а после этого присвоили некоторому элементу формы полученное значение *s*.  
Проверьте работу нашей html-страницы у себя в браузере.

Еще пример без пояснений.

<html>

<body>

<input maxlength=4 type="edit" id="sender" onkeypress="Filter();"/>

<input type=button id=send onclick="Send();" value="Send"/>

<BR>

<input maxlength=7 type="edit" id="channel" onkeypress="Filter();"/>

<input type=button id=recieve onclick="Recieve();" value="Recieve"/>

<BR>

<input type="edit" id="reciever" readonly/>

</body>

<script>

function Send()

{

alert("Send call. sender.value = " + sender.value);

}

function Recieve()

{

alert("Recieve call. channel.value = " + channel.value);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Пример фильтрации ввода //

function Filter()

{

var filter = "01";

window.event.returnValue = (filter.indexOf(String.fromCharCode(window.event.keyCode)) != -1);

}

</script>

</html>