Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Кафедра электронных вычислительных машин

Лабораторная работа №3

**МЕТОДЫ ЗАЩИТЫ ОТ ОШИБОК ИСПОЛЬЗУЕМЫ В СЕТЯХ**

Выполнил: Проверил:

ст. гр. №750502 Глоба А. А.

Альховик Д. Г.

Минск, 2019

**1**. **Код Хемминга.**

Коды Хэмминга — наиболее известные и, вероятно, первые из самоконтролирующихся и самокорректирующихся кодов. Построены они применительно к двоичной системе счисления.

Другими словами, это алгоритм, который позволяет закодировать какое-либо информационное символ определённым образом и после передачи (например по сети) определить появилась ли какая-то ошибка в этом сообщении (к примеру из-за помех) и, при возможности, восстановить это символ. Сегодня, я опишу самый простой алгоритм Хемминга, который может исправлять лишь одну ошибку.

Также стоит отметить, что существуют более совершенные модификации данного алгоритма, которые позволяют обнаруживать (и если возможно исправлять) большее количество ошибок.

Сразу стоит сказать, что Код Хэмминга состоит из двух частей. Первая часть кодирует исходное символ, вставляя в него в определённых местах контрольные биты (вычисленные особым образом). Вторая часть получает входящее символ и заново вычисляет контрольные биты (по тому же алгоритму, что и первая часть). Если все вновь вычисленные контрольные биты совпадают с полученными, то символ получено без ошибок. В противном случае, выводится символ об ошибке и при возможности ошибка исправляется.

Рассмотрим пример.

Допустим, у нас есть символ «D», который необходимо передать без ошибок. Для этого сначала нужно наш символ закодировать при помощи Кода Хэмминга. Нам необходимо представить его в бинарном виде.

Символ: “D” – ASCII-код: 68 – Бинарное представление: 01000100

Прежде всего, необходимо вставить контрольные биты. Они вставляются в строго определённых местах — это позиции с номерами, равными степеням двойки. В нашем случае (при длине информационного слова в 8 бит) это будут позиции 1, 2, 4, 8. Соответственно, у нас получилось 4 контрольных бит:

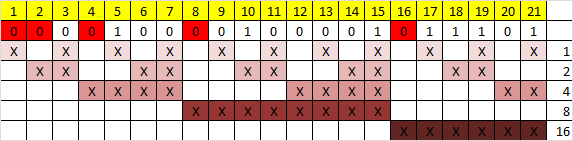
Было: 01000100

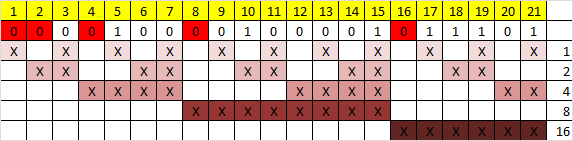
Стало: 000010000100

Таким образом, длина всего сообщения увеличилась на 4 бит. До вычисления самих контрольных бит, мы присвоили им значение «0».

Вычисление контрольных бит.

Теперь необходимо вычислить значение каждого контрольного бита. Значение каждого контрольного бита зависит от значений информационных бит, но не от всех, а только от тех, которые этот контрольных бит контролирует. Для того, чтобы понять, за какие биты отвечает каждых контрольный бит необходимо понять очень простую закономерность: контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N.





Здесь знаком «X» обозначены те биты, которые контролирует контрольный бит, номер которого справа. То есть, к примеру, бит номер 12 контролируется битами с номерами 4 и 8. Ясно, что чтобы узнать какими битами контролируется бит с номером N надо просто разложить N по степеням двойки.

Вычисление значения контрольного бита: берём каждый контрольный бит и смотрим сколько среди контролируемых им битов единиц, получаем некоторое целое число и, если оно чётное, то ставим ноль, в противном случае ставим единицу. Можно конечно и наоборот, если число чётное, то ставим единицу, в противном случае, ставим 0. Главное, чтобы в «кодирующей» и «декодирующей» частях алгоритм был одинаков. Высчитав контрольные биты для нашего информационного слова получаем следующее:

Было: 000010000100

Стало: 110110010100

Первая часть алгоритма завершена.

Декодирование и исправление ошибок.

Теперь, допустим, мы получили закодированное первой частью алгоритма символ, но он пришёл к нам с ошибкой. К примеру, мы получили такое (11-ый бит передался неправильно):

110110010110.

Вся вторая часть алгоритма заключается в том, что необходимо заново вычислить все контрольные биты и сравнить их с контрольными битами, которые мы получили. Так, посчитав контрольные биты с неправильным 11-ым битом мы получим такую картину:

000110000110.

Как мы видим, контрольные биты под номерами: 1, 2, 8 не совпадают с такими же контрольными битами, которые мы получили. Теперь просто сложив номера позиций неправильных контрольных бит (1 + 2 + 8 = 11) мы получаем позицию ошибочного бита. Теперь просто инвертировав его и отбросив контрольные биты, мы получим исходное символ в первозданном виде.