МИНИСТРЕСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**Отчет**

Лабораторная работа №1

*по дисциплине «Кодирование и передача информации»*

**Помехоустойчивое кодирование двоичных сообщений с использованием кодов Хэмминга**

|  |  |  |  |
| --- | --- | --- | --- |
| выполнил: | Симонов В.С. | проверил: | Катасонов Д.Н. |
| факультет: | АВТ | дата сдачи: | 31.03.2018 |
| группа: | АВТ-509 |  |  |

Новосибирск 2018

**Оглавление**

[**Цель работы** 3](#_Toc510226303)

[**1.** Постановка задачи 3](#_Toc510226304)

[**2.** Исходные данные 3](#_Toc510226305)

[**3.** Ход работы 4](#_Toc510226306)

[**3.1.** Описание кодирования по Хэммингу 4](#_Toc510226307)

[**3.2**. Описание декодирования и исправления ошибки по Хэммингу 7](#_Toc510226308)

[**3.3.** Невозможность коррекции двойных ошибок и любых ошибок большей кратности 8](#_Toc510226309)

[**3.4**. Основные недостатки кода Хэмминга 9](#_Toc510226310)

[**3.5**. Демонстрационная программа 12](#_Toc510226311)

[**3.5.1.** Состав демонстрационной программы 12](#_Toc510226312)

[**3.5.2.** Описание основных функций демонстрационной программы 12](#_Toc510226313)

[**3.5.3.** Исходный код демонстрационной программы с комментариями 12](#_Toc510226314)

[**Заключение** 17](#_Toc510226315)

**Цель работы**

Изучения кодирования и декодирования по Хэммингу. Выявление достоинств и недостатков данного вида кодирования. Программная реализация помехоустойчивого кодирования данных. Получение практических навыков в ходе реализации.

**1. Постановка задачи**

Сообщение должно быть защищено от ошибок при его передаче. Размер сообщения заранее не известен.

Требуется произвести помехоустойчивое кодирование исходных данных.

**2. Исходные данные**

Сообщение, состоящее из n байт информации.

**3. Ход работы**

## 3.1. Описание кодирования по Хэммингу

Кодирование по Хэммингу весьма несложный процесс. Достоинство кода в том, что реализация алгоритма требует небольших ресурсов и может быть выполнена аппаратно.

Исходными данными для кодирования является произвольная двоичная последовательность, например как это видно из таблицы 1.

Таблица 1 – Исходная битовая последовательность

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № бита | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | … | n |
| значение бита | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

Прежде всего, двоичная последовательность разделяется на куски размером в m бит. Размеры кусков не произвольны, их длина определяются формулой

m = 2r ‑ r - 1,

где r – любое целое число большее 2. Куски исходной двоичной последовательности будем называть «блоки исходного кода» и обозначать ai. Рассмотрим для определенности r = 4, тогда m = 11. Cоответствующее деление исходной битовой последовательности на блоки исходного кода изображено на рис. 1.

Далее исходные коды расширяют до n бит каждый, дополняя r контрольными битами. Полученные n-битные коды образуются так:

* Позиции с номерами 2i (i = 1, 2, …r) резервируются под контрольные биты;
* в остальные биты копируется исходный код в порядке следования его битов.

Расширенные блоки будем называть «блок кода» и обозначать bi. Расширение исходного кода для r = 4 и n = 15 продемонстрировано на рис. 2.3.

Затем вычисляют контрольные разряды. Для вычисления контрольных разрядов нужна вспомогательная матрица M размером (2r – 1) строк и r столбцов. Матрица заполняется по строкам, в каждую строку записывают двоичное представление чисел от 1 до 2r – 1. Такая матрица для r = 4 показана на рис. 2.4. Далее вычисляются контрольные разряды ci, для этого из матрицы M выбираются и суммируются по колонкам все строки номера которых совпадают с ненулевыми битами блока кода bi. Полученная строка из r битов записывается в контрольные разряды блока кода bi в порядке следования битов, как показано на рис. 2.4. Вычисление контрольных разрядов ci можно представить матричным умножением

ci = (bi)T⋅M,

здесь (bi)T — строка (вместо столбца) расширенного кода, где контрольные биты равны 0.

Полученные блоки кода можно вновь преобразовать в битовую последовательность и передавать по каналу связи. Код Хемминга способен детектировать и исправлять 1 (одну) ошибку на блок.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № бита | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | … | n |
| значение бита | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

Блоки исходного кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № бита  исх. кода | Блоки | | | |
| a1 | a2 | … | an |
| 1 | 0 | 0 |  |  |
| 2 | 1 | 1 |  |  |
| 3 | 1 | 0 |  |  |
| 4 | 1 | 0 |  |  |
| 5 | 0 | 1 |  |  |
| 6 | 1 | 1 |  |  |
| 7 | 0 | 1 |  |  |
| 8 | 0 | 0 |  |  |
| 9 | 1 | 1 |  |  |
| 10 | 1 | 0 |  |  |
| 11 | 1 | 0 |  |  |

Рис. 1 – разделение битовой последовательности на блоки исходного кода

Расширенные блоки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № бита  расш. кода | № бита  исх. кода | Блоки | | | |
| b1 | b2 | … | bn |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 | 1 | 0 | 0 |  |  |
| 4 |  |  |  |  |  |
| 5 | 2 | 1 | 1 |  |  |
| 6 | 3 | 1 | 0 |  |  |
| 7 | 4 | 1 | 0 |  |  |
| 8 |  |  |  |  |  |
| 9 | 5 | 0 | 1 |  |  |
| 10 | 6 | 1 | 1 |  |  |
| 11 | 7 | 0 | 1 |  |  |
| 12 | 8 | 0 | 0 |  |  |
| 13 | 9 | 1 | 1 |  |  |
| 14 | 10 | 1 | 0 |  |  |
| 15 | 11 | 1 | 0 |  |  |

Блоки исходного кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № бита  исх. кода | Блоки | | | |
| a1 | a2 | … | an |
| 1 | 0 | 0 |  |  |
| 2 | 1 | 1 |  |  |
| 3 | 1 | 0 |  |  |
| 4 | 1 | 0 |  |  |
| 5 | 0 | 1 |  |  |
| 6 | 1 | 1 |  |  |
| 7 | 0 | 1 |  |  |
| 8 | 0 | 0 |  |  |
| 9 | 1 | 1 |  |  |
| 10 | 1 | 0 |  |  |
| 11 | 1 | 0 |  |  |

Рис.2 – расширение блоков исходного кода контрольными разрядами

## 3.2. Описание декодирования и исправления ошибки по Хэммингу

Переданная по информационному каналу в приемник битовая последовательность делится на куски по n = 2r - 1 бит — получаются блоки кода. С каж­дым таким блоком выполняется операция

ci = (bi)T⋅M,

здесь (bi)T — строка (вместо столбца) расширенного кода. Пример приведен на рис. 2.5. Причем здесь контрольные разряды участвуют в вычислении суммы.

Если получено, что все биты ci равны нулю, то значит ошибок нет и коррекция не нужна, см. рис.3. Если хотя бы один бит ci не равен нулю (см. рис.4), то имела место ошибка. Значение ci преобразуют из битового представления в десятичное число i и бит блока кода с номером i — ошибочный бит (передан с ошибкой). Для исправления значение бита инвертируют: заменяют ноль на единицу, а единицу на ноль. В результате получаем правильное значение блока кода.

В заключение, контрольные разряды удаляются из блока и получается блок исходного кода. Эту операцию (проверка-коррекция) проводят с каждым блоком кода.

Вспомогательная матрица M для r = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

Блоки кода

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № бита  расш. кода | № бита  исх. кода | Блоки | | | |
| b1 | b2 | … | bn |
| 1 |  | 0 |  |  |  |
| 2 |  | 1 |  |  |  |
| 3 | 1 | 0 | 0 |  |  |
| 4 |  | 0 |  |  |  |
| 5 | 2 | 1 | 1 |  |  |
| 6 | 3 | 1 | 0 |  |  |
| 7 | 4 | 1 | 0 |  |  |
| 8 |  | 0 |  |  |  |
| 9 | 5 | 0 | 1 |  |  |
| 10 | 6 | 1 | 1 |  |  |
| 11 | 7 | 0 | 1 |  |  |
| 12 | 8 | 0 | 0 |  |  |
| 13 | 9 | 1 | 1 |  |  |
| 14 | 10 | 1 | 0 |  |  |
| 15 | 11 | 1 | 0 |  |  |

Строки M, соответствующие ненулевам битам блока

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |
| Сумма по столбцам | 0 | 1 | 0 | 0 |

Рис. 3 – вычисление контрольных разрядов

## 3.3. Невозможность коррекции двойных ошибок и любых ошибок большей кратности

Две и более ошибки в блоке кода Хэмминга невозможно исправить, хуже того невозможно отличить ошибку в одном бите и ошибку в двух и более битах.

Невозможность исправления двойной ошибки следует и простого факта – процедура коррекции (см. пункт 4.2.) всегда вычисляет один номер ошибочного бита, следовательно, второй и т.д. ошибочные биты просто невозможно указать. Пример двойной ошибки приведен на рис. 2.7.

Невозможность отличить ошибку в одном блоке от ошибки в двух и более блоках следует из факта - процедура коррекции (см. пункт 4.2.) всегда вычисляет один номер ошибочного бита в пределах допустимых номеров, следовательно, отличить случай одиночной ошибки от кратной невозможно.

Вспомогательная матрица M для r = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

Блоки кода

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № бита  расш. кода | № бита  исх. кода | Блоки | | | |
| b1 | b2 | … | bn |
| 1 |  | 0 |  |  |  |
| 2 |  | 1 |  |  |  |
| 3 | 1 | 0 |  |  |  |
| 4 |  | 0 |  |  |  |
| 5 | 2 | 1 |  |  |  |
| 6 | 3 | 1 |  |  |  |
| 7 | 4 | 1 |  |  |  |
| 8 |  | 0 |  |  |  |
| 9 | 5 | 0 |  |  |  |
| 10 | 6 | 1 |  |  |  |
| 11 | 7 | 0 |  |  |  |
| 12 | 8 | 0 |  |  |  |
| 13 | 9 | 1 |  |  |  |
| 14 | 10 | 1 |  |  |  |
| 15 | 11 | 1 |  |  |  |

Строки M, соответствующие ненулевам битам блока

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 2 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |
| Сумма по столбцам | 0 | 0 | 0 | 0 |

Ошибок нет.

Рис.4 – проверка на ошибки

## 3.4. Основные недостатки кода Хэмминга

Главным недостатком кода Хэмминга является не кратность размера исходного блока кода и блока кода степени двойки. Это обстоятельство затрудняет обработку кодов Хэмминга на компьютерах, которые оперируют порциями данных кратными степени двойки (8, 16, 32, 64 бит и т.д.).

Другим недостатком является невозможность создавать код для исправления двойных ошибок или ошибок большей кратности.

Вспомогательная матрица M для r = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

Блоки кода (с ошибкой)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № бита  расш. кода | № бита  исх. кода | Блоки | | | |
| b1 | b2 | … | bn |
| 1 |  | 0 |  |  |  |
| 2 |  | 1 |  |  |  |
| 3 | 1 | 0 |  |  |  |
| 4 |  | 0 |  |  |  |
| 5 | 2 | 1 |  |  |  |
| 6 | 3 | 1 |  |  |  |
| 7 | 4 | 1 |  |  |  |
| 8 |  | 0 |  |  |  |
| 9 | 5 | 1 | ошибка | | |
| 10 | 6 | 1 |  |  |  |
| 11 | 7 | 0 |  |  |  |
| 12 | 8 | 0 |  |  |  |
| 13 | 9 | 1 |  |  |  |
| 14 | 10 | 1 |  |  |  |
| 15 | 11 | 1 |  |  |  |

Строки M, соответствующие ненулевам

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 2 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |
| Сумма по столбцам | 1 | 0 | 0 | 1 |

Ошибка в бите 9.

Рис. 5 – проверка на ошибки и коррекция ошибок

Вспомогательная матрица M для r = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

Блоки кода (с ошибкой)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № бита  расш. кода | № бита  исх. кода | Блоки | | | |
| b1 | b2 | … | bn |
| 1 |  | 0 |  |  |  |
| 2 |  | 1 |  |  |  |
| 3 | 1 | 0 |  |  |  |
| 4 |  | 1 | ошибка | | |
| 5 | 2 | 1 |  |  |  |
| 6 | 3 | 1 |  |  |  |
| 7 | 4 | 1 |  |  |  |
| 8 |  | 0 |  |  |  |
| 9 | 5 | 1 | ошибка | | |
| 10 | 6 | 1 |  |  |  |
| 11 | 7 | 0 |  |  |  |
| 12 | 8 | 0 |  |  |  |
| 13 | 9 | 1 |  |  |  |
| 14 | 10 | 1 |  |  |  |
| 15 | 11 | 1 |  |  |  |

Строки M, соответствующие ненулевам

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № строки | № колонки | | | |
| 1 | 2 | 3 | 4 |
| 2 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |
| Сумма по столбцам | 1 | 0 | 1 | 1 |

Ошибка в бите 13. А ошибки совсем не там.

Рис.6 – проверка на ошибки и невозможность коррекции ошибок кратности два и выше

## 3.5. Демонстрационная программа

### 3.5.1. Состав демонстрационной программы

Для лучшего понимания процесса кодирования разработана демонстрационная программа (далее просто «программа»). Программа реализована с помощью C++ и может быть скомпилирована на любой более новой версии компилятора GNU C++11.

Программа реализована для целей обучения поэтому целесообразность ее применения (технический смысл) принесена в жертву максимальной скорости работы и наглядности процесса.

### 3.5.2. Описание основных функций демонстрационной программы

Программа состоит из следующих функций:

1) getSyndrome() – получает позицию ошибки для представленной последовательности бит. Если ошибки нет, то возвращает 0.

2) hamming() – кодирование фиксированной части сообщения.

3) setCode() – кодирование всего сообщения по частям

4) getMessage() – декодирование сообщения.

### 3.5.3. Исходный код демонстрационной программы с комментариями

#define TRUE 1;

#define FALSE 0;

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <malloc.h>

#include <string.h>

#include <math.h>

#include <locale.h>

const int byte\_size = 8; // колличество бит в байте

// подсчет количества контрольных бит

long countKbits(long length)

{

int sum = 0;

for(int i=0; length>0; i++){

sum++;

length-=pow(2,i)-1;

}

return sum;

}

// вывод данных в двоичном представлении

void coutt(char\* byte, int size)

{

unsigned char mask = 0x80;

char zn;

int count = 0;

for(int j=0; j<size; j++)

{

if(count==4)

{

count=0;

printf("\n");

}

count++;

for(int i=0; i<8; i++)

{

zn = byte[j] & mask >> i;

if(zn!=0) printf("%c", '1');

else printf("%c", '0');

}

printf(" ");

}

printf("\n");

}

// Смещение массива вправо

char\* setOffset(char\* message, int offset)

{

char\* result = NULL;

int size = 0;

if (offset<9 && offset>0){

size = strlen(message);

result = (char\*)malloc(size+1);

for(int i=0; i<=size; ++i) result[i] = 0;

for(int i=0; i<size; ++i)

{

result[i] |= (unsigned char)message[i] >> offset;

result[i+1] = (unsigned char)message[i] << (byte\_size - offset);

}

return result;

}

return message;

}

// получение синдрома для заданной последовательности бит

int getSyndrome(char\* message, int part, int begin)

{

unsigned char mask = 0x80;

int sum = 0, temp;

int kbits = countKbits(part);

part+=kbits;

int syndrome = 0;

// установка контрольных бит

for(int i=0; i<kbits; i++)

{

sum = 0; // pos\_p - используется дважды, чтобы не использовать еще одну переменную

temp = pow(2,i)-1; // позиция контрольного бита

for(int j = temp; j<part; j+=(i+1)\*2)

{

for(int k = 0; k<=i; k++) // подгруппа контролируемых битов

{

if(1)//j+k!=temp) // кроме первого контролируемого бита

{

//printf("%i\n", ((j + k + begin) % byte\_size));

if((message[(j + k + begin) / byte\_size] & (mask >> ((j + k + begin) % byte\_size))) != 0)

{

sum++; // подсчет контрольной суммы

}

}

}

}

if(sum%2!=0) // значение контр.бита - инвертированная кольцевая сумма

{

syndrome+=i+1;

}

}

return syndrome;

}

// кодирование сообщения длиной size бит,

// память: 61+size байт

char\* hemming(char\* part, int size, int pos\_p)

{

unsigned char mask = 0x80; // маска для чтения "10000000"

int temp, kbits=countKbits(size); // количество контрольных бит

int enc\_size = size+kbits;

int part\_size = (size+kbits+1)/byte\_size+((size+kbits+1)%byte\_size>0?1:0); // размер закодированной части в байтах

char buf, \*result = (char\*)malloc(part\_size), \*buff;

int r\_offset=0; // pos\_p - позиция в part, r\_offset - смещение для вставки в result

//printf("%i\n", part\_size);

for(int i=0; i<part\_size;i++) result[i]=0; //обнуление массива

// Заполнение массива исходными данными

for(int i=0; i<kbits; i++)

{

temp = pow(2,i); // позиция контрольного бита

for(int pos\_r = temp; pos\_r<temp\*2-1 && pos\_r<enc\_size; ++pos\_r) // от текущего до следующего контрольного бита

{

buf = part[pos\_p / byte\_size] & (mask >> pos\_p % byte\_size); // получение бита из part

r\_offset = pos\_r % byte\_size - pos\_p % byte\_size; // смещение этого бита

result[pos\_r / byte\_size] |= (r\_offset>=0)? buf >> r\_offset : buf << -r\_offset; // запись бита в result

pos\_p++;

}

}

//coutt(result, 2);

// установка контрольных бит

for(int i=0; i<kbits; i++)

{

pos\_p = 0; // pos\_p - используется дважды, чтобы не использовать еще одну переменную

temp = pow(2,i)-1; // позиция контрольного бита

for(int j = temp; j<enc\_size; j+=(i+1)\*2)

{

for(int k = 0; k<=i; k++) // подгруппа контролируемых битов

{

if(j+k!=temp) // кроме первого контролируемого бита

{

if((result[(j + k) / byte\_size] & (mask >> (j + k) % byte\_size)) != 0)

{

pos\_p++; // подсчет контрольной суммы

}

}

}

}

if(pos\_p%2!=0) // значение контр.бита - инвертированная кольцевая сумма

{

result[temp/byte\_size] |= mask >> temp % byte\_size;

}

}

// установка бита четности

buff = result;

result = setOffset(result, 1); // единичное смещение

free(buff);

pos\_p = 0; // повторное использование переменной

for(int i=1; i<=enc\_size; ++i)

{

if((result[i/byte\_size] & (mask >> (i%byte\_size))) != 0) pos\_p++; // подсчет контрольной суммы

}

if(pos\_p%2!=0) result[0] |= mask; // установка контрольного бита

return result;

}

// кодрование по хеммингу

char\* setCode(char\* message, int part){

int mes\_p=0, res\_p=0;

int mes\_size =strlen(message);

int mes\_bits = mes\_size\*byte\_size;

int count\_parts = (mes\_bits % part>0) ? mes\_bits / part + 1 : mes\_bits / part;

int kbits = countKbits(part);

int part\_bytes = ((part+kbits+1)%byte\_size>0) ? (part+kbits+1)/byte\_size+1 : (part+kbits+1)/byte\_size;

int res\_size = ((kbits+1)\*count\_parts%byte\_size>0)? (kbits+1)\*count\_parts/byte\_size+1: (kbits+1)\*count\_parts/byte\_size;

res\_size+=mes\_size;

char\* temp=NULL, \*buf;

char\* result=(char\*)malloc(res\_size);

for(int i=0; i<res\_size; i++) result[i]=0;

for(int i=0; i<count\_parts; ++i)

{

temp = hemming(message, part, mes\_p);

buf = temp;

temp = setOffset(temp, res\_p % byte\_size);

for(int j=0; j<=part\_bytes; ++j)

{

result[res\_p/byte\_size+j] |= temp[j];

}

mes\_p+=part;

res\_p+=part+kbits+1;

free(buf);

}

return result;

}

// декодирование кода

char\* getMessage(char\* code, int part){

unsigned char mask = 0x80;

unsigned char temp;

int syndrome, kbits = countKbits(part);

int part\_size = part+1+kbits;

int count\_parts = strlen(code)\*byte\_size/(part+1+kbits);

int count = 0, pos\_c = 0, pos\_r=0, r\_offset;

int res\_size = (count\_parts\*part%byte\_size>0)? count\_parts\*part/byte\_size+1 : count\_parts\*part/byte\_size;

char\* result = (char\*)malloc(res\_size);

for(int i=0; i<res\_size; i++) result[i]=0;

for(int j=0; j<count\_parts; ++j)

{

count = 0;

for(int i=0; i<part\_size; ++i)

{

if((code[(pos\_c+i)/byte\_size] & (mask >> ((pos\_c+i)%byte\_size))) != 0) count++; // подсчет контрольной суммы

}

if(count%2==0) temp=1; // проверка бита четности

else temp = 0;

syndrome = getSyndrome(code, part, pos\_c+1);

if(syndrome!=0 && temp == 0) // одна ошибка

{

code[(pos\_c + syndrome+1)/byte\_size] ^= (mask >> ((pos\_c + syndrome+1) % byte\_size)); // коррекция

}

if(syndrome!=0 && temp == 1 || syndrome==0 && temp == 0) // две ошибки

{

free(result);

return NULL;

}

pos\_c+=part\_size;

}

pos\_c = 0;

// формирование сообщения

for(int i = 0; i<count\_parts; ++i)

{

count = 1;

for(int j = 1; j<part\_size; ++j)

{

if(j==count) count \*= 2;

else

{

r\_offset = pos\_r % byte\_size - (pos\_c+j) % byte\_size; // смещение этого бита

temp = code[(pos\_c+j)/byte\_size] & mask >> (pos\_c+j)%byte\_size;

result[pos\_r/byte\_size] |= (r\_offset>=0)? temp >> r\_offset : temp << -r\_offset;

pos\_r++;

}

}

pos\_c+=part\_size;

}

return result;

}

int main(int argc, char\*\* argv){

setlocale(0,"");

char\* ans = setCode("Hello!", 8); // кодирование

unsigned char t = 0x80 >> 5;

ans[0] ^= t; // добавление ошибки

ans[3] ^= t >>1; // добавление ошибки

char\* temp = getMessage(ans, 8); // декодирование

printf("\n%s\n", temp);

free(ans);

free(temp);

return 0;

}

**Заключение**

Изучено кодирование и декодирование по Хэммингу. Выявлены достоинства и недостатки данного вида кодирования. Создана программная реализация помехоустойчивого кодирования данных. Получены практические навыки в ходе реализации.