Национальный исследовательский ядерный университет МИФИ

Отделение Интеллектуальных кибернетических систем

Направление подготовки Информатика и вычислительная техника

**Лабораторная работа №3**

«Разработка и реализация программного

компонента компьютерной технологии

помехоустойчивого кодирования в

каналах передачи и хранения»

**Выполнил**: студент 2 курса группы ИBT1-Б22

Лутфиллаев Абдугаффор Самадович

**Проверил**: доцент кафедры ОИКС

Мышев Алексей Владимирович

Обнинск 2023

**Основные понятия и определения**

**Бит** – базовая единица измерения информации в каналах хранения и передачи, имеющая в них соответственно логические и физические прототипы. Логическим прототипом бита является символическое обозначение его образа, т.е. множество символов базового алфавита, в физической среде информационного канала. Например, для булевой логики – это ноль и единица, символические образы которых определяются символами 0 и 1.

**Бинарное информационное множество** – это множество элементов базового алфавита, образующих логическую структуру в виде сегмента памяти в физической среде канала хранения или поток логически и функционально связанных данных в канале передачи. В каналах передачи и хранения любой информационный объект (файлы и другие логические структуры) рассматриваются как бинарные информационные множества, на которых определяются информационные пространства, задаваемые в виде кортежа <Х,Nm>, где Х - множество символьных цепочек алфавита Nm, который представляет собой множество бинарных цепочек длиной m битов.

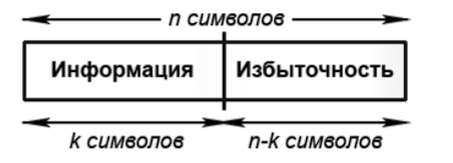
**Информационное пространство** в виде кортежа <Х,Nm> является математической моделью информационного объекта, определенного как бинарное множество. Тогда вероятностно-статистической характеристикой информационного объекта (IO) в <Х,Nm> для заданного алфавита Nm будет таблица информационной насыщенности IO, которая представляет собой дискретное распределение вероятностей букв Nm в IO.

**Определение 1.** Кодовым бинарным словом длины m называется цепочка из m битов.

**Определение 2.** Кодом называется множество кодовых слов.

**Определение 3.** Кодовое расстояние E1(Ci,Ci+1) по Хеммингу между кодовыми словами Ci и Ci+1 определяется как количество позиций, в которых значения битов не совпадают.

**Кодирование для исправления ошибок: основные положения**

Все коды, исправляющие ошибки, основаны на одной общей идее: для исправления ошибок, которые могут возникнуть в процессе передачи или хранения информации, к ней добавляется некоторая избыточность. По основной схеме, используемой на практике, избыточные символы дописываются за информационными, образуя кодовую последовательность или **кодовое слово (codeword).** На рисунке №1 показана геометрическая иллюстрация процедуры формирования **блокового кода (bloch code).** Такое кодирование называют **систематическим (systematic).**

Это означает, что информационные символы всегда появляются на первых **k** позициях кодового слова. Символы на оставшихся **(n - k)** позициях являются различными функциями от информационных символов, обеспечивающими тем самым избыточность, необходимую для обнаружения или исправления ошибок. Множество всех кодовых последовательностей называют **кодом,** **исправляющим ошибки** **(error correcting code).**

**Цель работы**

Разработка и реализация компьютерных технологий помехоустойчивого кодирования информации на основе синтеза традиционных методов избыточного кодирования и способов виртуализации каналов передачи и систем хранения в виде готового программного продукта; приобретение базовых знаний применения методов избыточного кодирования для создания моделей алгоритмов и процедур программных компонент таких технологий; освоение навыков применения теории кодов, исправляющих ошибки для решения практических задач защиты (физической и конфиденциальной) информации в каналах передачи и системах хранения на основе технологий виртуализации; освоение методологии анализа и оценки эффективности и степени надежности помехоустойчивой защиты разрабатываемой технологии.

**Этапы выполнения работы**

1. Разработка модели алгоритмов и процедур программных компонентов помехоустойчивого кодирования информационного объекта в кодировании каналов передачи и хранения согласно схеме, определенной в задании.

2. Реализация (см. лаб. раб. №1 и №2) программного интерфейса ввода - вывода файловых структур с ВУ в ОЗУ и обратно с учетом особенностей периферии.

3. Реализация моделей алгоритмов и процедур технологий кодера и декодера на бинарных полях в виде программных компонент.

4. Разработка и реализация программного оконного диалогового интерфейса для ввода исходных данных, записи закодированного информационного объекта на ВУ и чтения в ОЗУ.

5. Реализация алгоритмов программных компонент преобразования файлов на бинарных полях памяти.

6. Отладка программных компонент.

7. Тестирование и контрольные примеры.

8. Расчеты и анализ. Выводы.

9. Прием-сдача программного компонента в диалоговом режиме.

**Входные данные**

|  |  |
| --- | --- |
| Input.txt | Файл с исходными данными |
| n | Длина входного слова |

**Листинг программы:**

#include <iostream>

#include <fstream>

#include <string>

#include <bitset>

#include <vector>

#include <set>

#include <cmath>

#include <sstream>

using namespace std;

int controlBits = 0;

// Управляющие последовательности ANSI для цветов

#define ANSI\_COLOR\_RED     "\x1b[31m"

#define ANSI\_COLOR\_GREEN   "\x1b[32m"

#define ANSI\_COLOR\_YELLOW  "\x1b[33m"

#define ANSI\_COLOR\_BLUE    "\x1b[34m"

#define ANSI\_COLOR\_MAGENTA "\x1b[35m"

#define ANSI\_COLOR\_CYAN    "\x1b[36m"

#define ANSI\_COLOR\_RESET   "\x1b[0m"

std::string textToBinary(const std::string& text) {

    std::string binary;

    for (char c : text) {

        binary += std::bitset<8>(c).to\_string();

    }

    return binary;

}

std::string binaryToText(const std::string& binary)

{

    std::string text;

    for (int i = 0; i < binary.length(); i += 8) {

        text += static\_cast<char>(std::bitset<8>(binary.substr(i, 8)).to\_ulong());

    }

    return text;

}

std::vector<std::string> splitBinary(const std::string& binary, int n) {

    std::vector<std::string> segments;

    for (size\_t i = 0; i < binary.size(); i += n) {

        segments.push\_back(binary.substr(i, n));

    }

    return segments;

}

string readBinaryFile(const string& filename = "input.txt") {

    ifstream file(filename, ios::binary);

    if (!file.is\_open())

    {

        cerr << "Ne udalos schitat dannie iz faila " << filename << endl;;

    }

    else

    {

        stringstream buffer;

        buffer << file.rdbuf();

        return buffer.str();

    }

    return string(8, '0');

}

bool writeBinaryFile(const string& binaryString, const string& filename = "output.bin")

{

    ofstream output(filename, ios::binary);

    if (output.is\_open())

    {

        for (int unsigned i = 0; i < binaryString.length(); i += 8) {

            char c = static\_cast<char>(bitset<8>(binaryString.substr(i, 8)).to\_ulong());

            output.write(&c, 1);

        }

        return true;

    }

    return false;

}

string hammingCoding(const std::vector<std::string>& segments)

{

    bool controlBit = false;

    string hammingCode;

    string a;

    for (const string& segment : segments)

    {

        string codingSegment = segment;

        bool bit = 0;

        for (auto c: codingSegment)

        {

            if (c == '1')

            {

                bit = !bit;

            }

        }

        a += (bit ? '1' : '0');

        hammingCode += codingSegment;

    }

    hammingCode += a;

    return hammingCode;

}

std::string hammingDecoding(const std::vector<std::string>& segments)

{

    string hammingDecodetText;

    for (const string& word : segments) {

        string codeWord = word;

        bool bit = 0;

        bool preBit = (codeWord[codeWord.size()-1] == '0'? 0:1);

        codeWord.erase(codeWord.length() - 1);

        for (auto c : codeWord)

        {

            if (c == '1')

            {

                bit = !bit;

            }

        }

        if (preBit != bit)

        {

            cout << ANSI\_COLOR\_RED << codeWord << ANSI\_COLOR\_RESET << endl;

        }

        else

        {

            cout << codeWord << endl;

        }

        hammingDecodetText += codeWord;

    }

    return hammingDecodetText;

}

int main() {

    setlocale(0, "rus");

    std::string inputPath = "input.txt";

    std::string outputPath;

    std::string inputText = readBinaryFile(inputPath);

    std::cout << "Vkhodnie dannie iz faila " << inputPath << ": \n" << inputText << std::endl;

    std::string binaryText = textToBinary(inputText);

    std::cout << "Vkhodnie dannie iz faila " << inputPath << " v binarnom vvide: \n" << binaryText << std::endl;

    int n;

    std::cout << "Vvedite chislo bit dliya razbieniya na segmenti: ";

    std::cin >> n;

    if (n <= 0)

    {

        std::cerr << "Nevozmozhno zakodirovat po 0 i menshe bit!\n";

        return 1;

    }

    std::vector<std::string> binarySegments = splitBinary(binaryText, n);

    std::cout << "Binarnie segmenti: \n";

    cout << ANSI\_COLOR\_CYAN;

    for (auto segment : binarySegments)

    {

        cout << segment << endl;

    }

    cout << ANSI\_COLOR\_RESET << endl;

    string hammingCode = hammingCoding(binarySegments);

    int encodedBitsSplit = n + 1;

    std::vector<std::string> splittedHammingCode = splitBinary(hammingCode, encodedBitsSplit);

    cout << "Segmenti s kontrolnimi bitami (kontrolnie biti otmecheni zelenim):\n";

    for (const string& segment : splittedHammingCode)

    {

        for (size\_t i = 0; i < segment.length(); i++)

        {

            if (i == segment.length()-1)

            {

                cout << ANSI\_COLOR\_GREEN;

            }

            cout << segment[i];

            cout << ANSI\_COLOR\_RESET;

        }

        cout << endl;

    }

    int outputZeros = 8 - hammingCode.length() % 8;

    cout << "Binarnaya stroka s kontrolnimi bitami:" << endl;

    if (outputZeros != 8){

        cout << "(Zholtim otmecheni neznachashie nuli)" << endl;

        string zeros(outputZeros, '0');

        cout << "-";

        for (int i = 0; i < hammingCode.length(); i += 8)

        {

            cout << hammingCode.substr(i, 8) << "-";

        }

        hammingCode += zeros;

        cout << ANSI\_COLOR\_YELLOW;

        cout << zeros;

        cout << ANSI\_COLOR\_RESET << "-\n\n";;

    }

    else {

        cout << "-";

        for (int i = 0; i < hammingCode.length(); i += 8)

        {

            cout << hammingCode.substr(i, 8) << "-";

        }

        cout << endl << endl;

    }

    cin.ignore();

    getline(cin, outputPath);

    if (outputPath.empty())

    {

        writeBinaryFile(hammingCode);

    }

    else

    {

        writeBinaryFile(hammingCode, outputPath);

    }

    cout << ANSI\_COLOR\_GREEN << "Fail sohranion" << ANSI\_COLOR\_RESET << endl;

    cout << "Izmenite binarnie dannie, dliya proverki rabotosposobnosti koda\n";

    system("pause");

    string newHammingCode = textToBinary(readBinaryFile((outputPath.empty() ? "output.bin" : outputPath)));

    if (outputZeros != 8)

        newHammingCode.erase(newHammingCode.end() - outputZeros, newHammingCode.end());

    std::vector<std::string> newSplittedHammingCode = splitBinary(newHammingCode, encodedBitsSplit);

    cout << "Segmenti s kontrolnimi bitami (kontrolnie biti otmecheni zelionim)\n(zholtim otmecheni izmenionnie biti):\n";

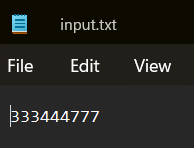
    string decoded = hammingDecoding(newSplittedHammingCode);

    return 0;

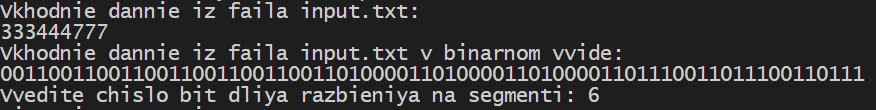
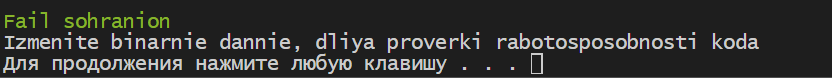
}

**Тестирование**

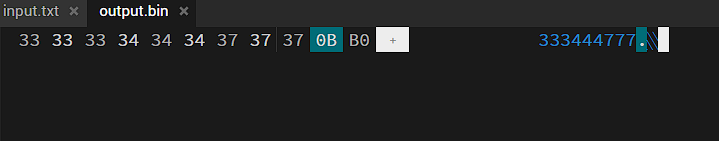
Исходный файл:



Работа программы:



**Результат:**

****

**Вывод**:

Было разработано и реализовано компьютерная технология помехоустойчивого кодирования информации на основе синтеза традиционных методов избыточного кодирования и способов виртуализации каналов передачи и систем хранения в виде готового программного продукта.