Национальный исследовательский ядерный университет МИФИ

Отделение Интеллектуальных кибернетических систем

Направление подготовки Информатика и вычислительная техника

**Лабораторная работа №1**

«Разработка и реализация программного

компонента компьютерной технологии

криптографической защиты файловых

структур в каналах хранения и передачи»

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

Лутфиллаев Абдугаффор Самадович

**Проверил**: доцент кафедры ОИКС

Мышев Алексей Владимирович

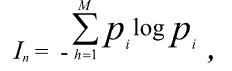
Обнинск 2023

**Основные понятия и определения**

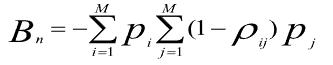
**Бит** – базовая единица измерения информации в каналах хранения и передачи, имеющая в них соответственно логические и физические прототипы. Логическим прототипом бита является символическое обозначение его образа, т.е. множество символов базового алфавита, в физической среде информационного канала. 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.

В качестве количественной меры информации IO используем следующие оценки. Во-первых, шенноновское определение энтропии, которое задается в виде выражения

где p i – это вероятность появления i-ой буквы алфавита N m в информационном объекте IO, |М| - мощность алфавита Nm.

Во-вторых, В – энтропия, которая в отличии от шенноновской определена на метрическом вероятностном пространстве и задается следующим выражением:

0 < ρij < 1 - рандомизированная мера на метрическом вероятностном пространстве, которая определяется выражением (хотя можно использовать множество и других)

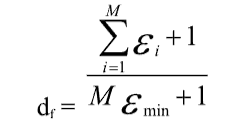
Количественная характеристика информации на основе В – энтропии отражает степень насыщенности IO буквами N m с учетом их близости по вероятности, т.е. она позволяет оценить «информационную геометрию» IO.

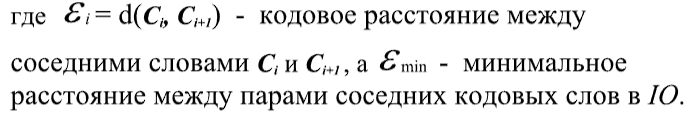
Под количественной мерой неопределенности представления информационного объекта IO фиксированной длины, определенного на <Х,Nm>, понимается ансамбль, порождаемый IO на <Х,Nm>. Ансамбль в этом случае представляет собой дискретное распределение вероятностей возможного представления IO элементами его таблицы информационной насыщенности. В качестве выражений для получения оценок количественной меры неопределенности используются также выражения (1.1) и (1.2), в которых p i -вероятность i-го представления IO в таблице возможных представлений, а М – это множество возможных представлений IO или объем таблицы представления. Поясним это на следующем примере.

**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.

Тогда в качестве интегральной характеристики IO можно использовать геометрическую информационную размерность, которая определяется следующим выражением:



**Цель работы:**

Разработка и реализация компьютерных технологий криптографической защиты информации в каналах передачи и хранения в виде готового программного продукта, приобретение базовых знаний построения моделей алгоритма и процедур программных компонент таких технологий, освоение навыков анализа и оценки сложности защиты и надежности технологии продукта.

**Входные данные**

|  |  |
| --- | --- |
| Input.txt | Файл с исходными данными |
| n | Длина входного слова |
| Segment\_length | Длина выходного слова |

**Листинг программы:**

#include <iostream>

#include <fstream>

#include <string>

#include <map>

#include <bitset>

#include <vector>

#include <set>

#include <cmath>

#include <Windows.h>

// ANSI управляющие последовательности 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"

// Функция для преобразования текста в двоичное представление

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

    std::string binary;

    for (char c : text) {

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

    }

    return binary;

}

// Функция для разделения двоичной строки на сегменты фиксированной длины

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;

}

// Функция для вычисления количества уникальных сегментов

int numberOfUniqueSegments(std::vector<std::string>& binarySegments) {

    std::set<std::string> Unique;

    for (std::string& segment : binarySegments) {

        Unique.insert(segment);

    }

    std::cout << "NumberOfUniqueElements: " << Unique.size() << std::endl;

    return (ceil(log2(Unique.size())));

}

int main() {

    // Пути к файлам

    setlocale(LC\_ALL, "ru\_RU.UTF-8");

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

    // Открываем файл с входными данными

    std::ifstream inputFile(inputPath);

    if (!inputFile) {

        std::cerr << "Error opening input.txt" << std::endl;

        return 1;

    }

    // Считываем данные из файла в строку

    std::string inputText;

    std::string line;

    while (std::getline(inputFile, line)) {

        inputText += line;

    }

    inputFile.close();

    std::cout << "Input data from the file " << inputPath << ": \n" << inputText << std::endl;

    // Преобразуем входные данные в двоичное представление

    std::string binaryText = textToBinary(inputText);

    std::cout << "Input data from the file " << inputPath << " in binary form: \n" << binaryText << std::endl;

    // Запрашиваем у пользователя новую длину битсета

    int n;

    std::cout << "Bits of entire alphabet: ";

    std::cin >> n;

    if (n <= 0) {

        std::cerr << "Cannot encode 0 or fewer bits." << std::endl;

        return 1;

    }

    // Разделяем двоичное представление на сегменты

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

    // Запрашиваем у пользователя длину сегмента

    int segmentLength;

    std::cout << "Bits of output alphabet: ";

    std::cin >> segmentLength;

    if (segmentLength <= 0 || segmentLength < numberOfUniqueSegments(binarySegments)) {

        std::cerr << "Привет мир" << std::endl;

        return 1;

    }

    // Вычисляем количество уникальных сегментов

    // numberOfUniqueSegments(binarySegments);

    // Инициализируем словарь кодов

    std::map<std::string, std::string> codeMap;

    std::string currentCode;

    std::set<std::string> codes;

    if(segmentLength == 1){

        currentCode = std::string(1, '0');

    }

    else{

        for (int i = 0; i < segmentLength; i++) {

        int r = std::rand() % 2;

        currentCode += char(r + '0');

        }

    }

    int zeros = 0;

    // Выводим словарь кодов на экран

    std::cout << "Dictionary:\n";

    for (std::string& segment : binarySegments) {

        while (segment.size() < n) {

            segment += '0';

            zeros++;

        }

        if (codeMap.find(segment) == codeMap.end()) {

            codeMap[segment] = currentCode;

            codes.insert(currentCode);

            std::string newCurrentCode = "";

            if(segmentLength == 1){

                newCurrentCode = std::string(1, '1');

            }

            else{

                while (codes.find(newCurrentCode) != codes.end() || newCurrentCode == ""){

                newCurrentCode.clear();

                for (int i = 0; i < segmentLength; i++) {

                    int r = std::rand() % 2;

                    newCurrentCode += char(r+'0');

                }

                }

            }

        currentCode = newCurrentCode;

        }

        std::cout << ANSI\_COLOR\_GREEN << "Input:" << segment << '\t'  << ANSI\_COLOR\_YELLOW << "Output:" << codeMap[segment] << std::endl;

    }

    std::cout << ANSI\_COLOR\_RESET;

    std::string encodedText;

    for (const std::string& segment : binarySegments) {

        encodedText += codeMap[segment];

    }

    std::cout << ANSI\_COLOR\_RESET;

    // Запись закодированных данных в файл

    std::ofstream encodedFile("encoded.txt", std::ios::binary);

    int outputZeros = 0;

    if (encodedText.length() % 8 != 0) {

        for (size\_t j = 0; encodedText.length() % 8 != 0; j++) {

            encodedText += '0';

            outputZeros++;

        }

    }

    std::cout << "\nEncoded data: " << std::endl << encodedText << std::endl;

    for (size\_t i = 0; i < encodedText.length(); i += 8) {

        std::bitset<8> byte(encodedText.substr(i, 8));

        char byteChar = static\_cast<char>(byte.to\_ulong());

        encodedFile.write(&byteChar, 1);

    }

    encodedFile.close();

    // Открытие файла с закодированными данными для декодирования

    std::ifstream decodedInputFile("encoded.txt", std::ios::binary);

    if (!decodedInputFile) {

        std::cerr << "Error opening encoded.txt" << std::endl;

        return 1;

    }

    // Чтение и перевод закодированных данных в бинарную систему

    std::string decodedByteText;

    char byteChar;

    std::string binaryByte;

    std::cout << "\nReading data from the encoded file:\n";

    while (decodedInputFile.read(&byteChar, 1)) {

        binaryByte = std::bitset<8>(byteChar).to\_string();

        std::cout << ANSI\_COLOR\_MAGENTA << std::bitset<8>(byteChar).to\_string() << std::endl;

        decodedByteText += binaryByte;

    }

    std::cout << ANSI\_COLOR\_RESET;

    decodedInputFile.close();

    if (decodedByteText.empty()) {

        std::cerr << "No data to decode." << std::endl;

        return 1;

    }

    // Декодирование

    std::string decodedOutputText;

    if (outputZeros != 0) {

        decodedByteText.erase(decodedByteText.length() - outputZeros);

    }

    for (size\_t i = 0; i < decodedByteText.length(); i += segmentLength) {

        for (auto pair : codeMap) {

            if (i + segmentLength <= decodedByteText.size()) {

                if (pair.second == decodedByteText.substr(i, segmentLength)) {

                    decodedOutputText += pair.first;

                    break;

                }

            }

        }

    }

    if (zeros != 0) {

        decodedOutputText.erase(decodedOutputText.length() - zeros);

    }

    std::vector<unsigned char> decodedBytes;

    std::cout << "\nDecoded binary sequence:\n" << decodedOutputText << std::endl;

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

    for (size\_t i = 0; i < decodedOutputText.size(); i += 8) {

        std::string byteStr = decodedOutputText.substr(i, 8);

        unsigned char byte = static\_cast<unsigned char>(std::bitset<8>(byteStr).to\_ulong());

        decodedBytes.push\_back(byte);

    }

    std::string decodedText(decodedBytes.begin(), decodedBytes.end());

    // Запись декодированных данных в файл

    std::ofstream decodedFile("decoded.txt");

    decodedFile << decodedText;

    std::cout << "\nDecoded text:\n" << ANSI\_COLOR\_GREEN << decodedText << std::endl;

    decodedFile.close();

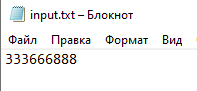
    std::cout << "Encoding and decoding completed" << ANSI\_COLOR\_RESET << std::endl;

    return 0;

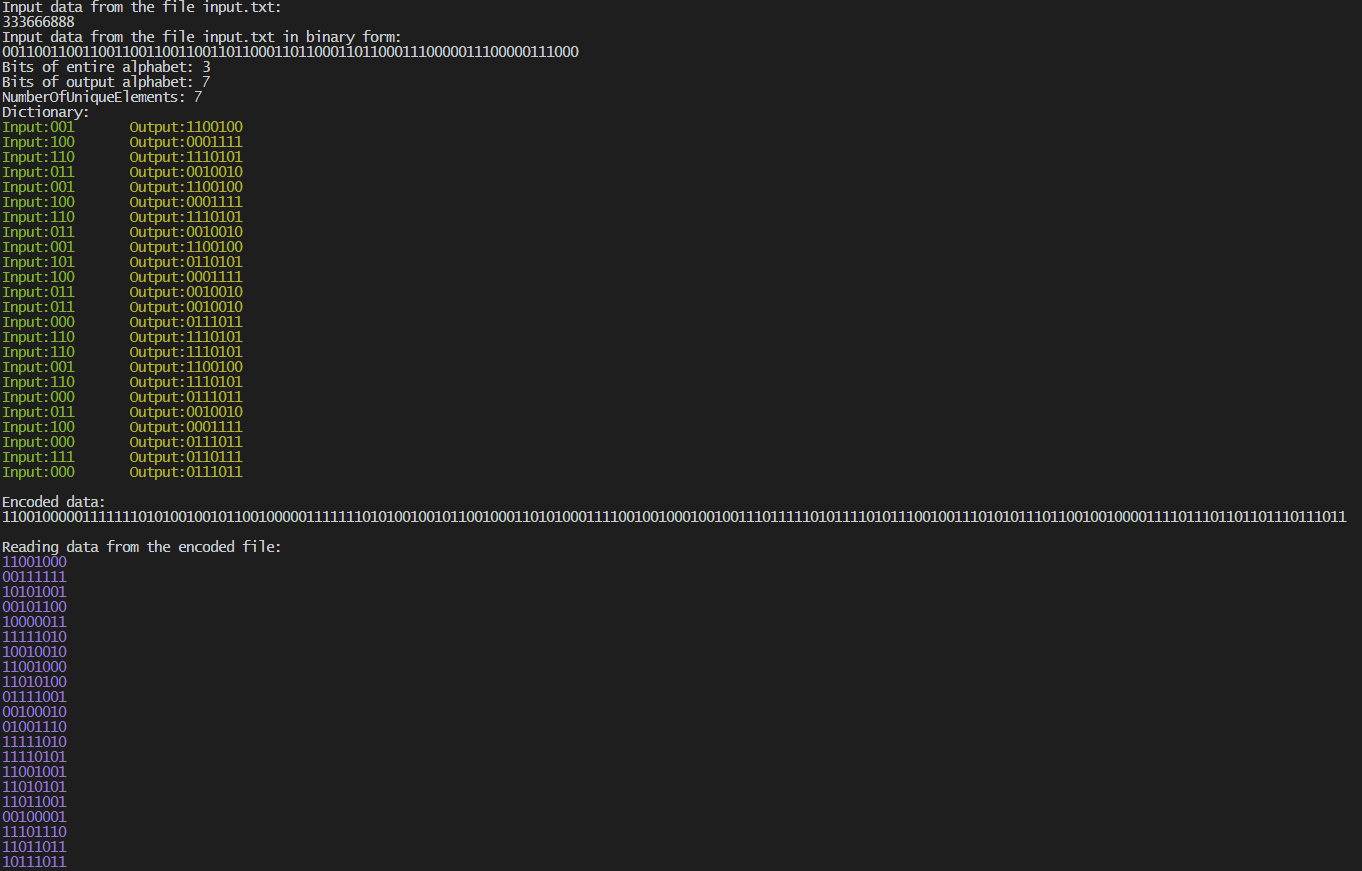
}

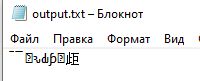
**Тестирование**

Исходный файл:



Работа программы:



Результат в файле:  


**Вывод**:

Было разработана и реализована компьютерная технология криптографической защиты информации в каналах передачи и хранения в виде готового программного продукта. Были приобретены базовые знания построения моделей алгоритма и процедур программных компонент.