Национальный исследовательский ядерный университет МИФИ

Отделение Интеллектуальных кибернетических систем

Направление подготовки Информатика и вычислительная техника

**Лабораторная работа №2**

«Разработка и реализация программного

компонента компьютерной технологии

сжатия файловых структур в каналах

хранения и защиты»

**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.

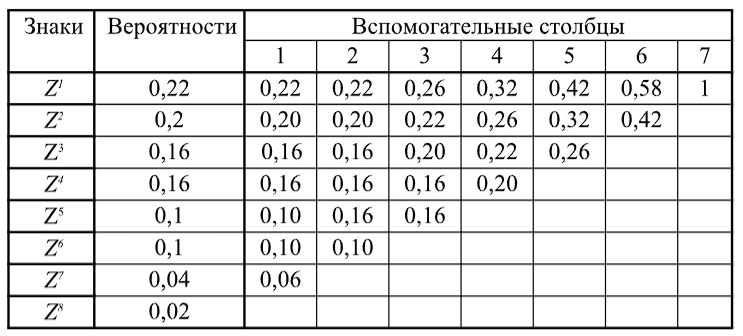
**Информационное пространство** в виде кортежа <Х,Nm> является математической моделью информационного объекта, определенного как бинарное множество. Тогда вероятностно-статистической характеристикой информационного объекта (IO) в <Х,Nm> для заданного алфавита Nm будет таблица информационной насыщенности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

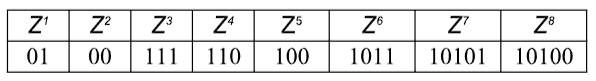
**Определение 1.** Кодовым бинарным словом длины m называется цепочка из m битов.

**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.

**Метод Хаффмена**

Методика разработки алгоритмов и процедур технологий сжатия по этому методу достаточно проста и включает в себя формирование двух элементов: треугольную матрицу вероятностей и построение кодового древа. Треугольная матрица вероятностей имеет размерность (L,L), где L –количество элементов входного алфавита. Элементы столбцов этой матрицы – это вероятности. Первый столбец задается как вероятности элементов входного алфавита, а элементы последующих столбцов формируются по методике Хаффмена. Процесс формирования матрицы поясняется таблицей.

Кодовое древо для формирования элементов выходного алфавита строится на основе матрицы вероятностей. Из точки (первый элемент последнего столбца), соответствующий вероятности 1, направляем две ветви, причем ветви с большей вероятностью присваиваем – 1, а с меньшей – 0. Такое последовательное ветвление продолжаем до тех пор, пока не дойдем до вероятности каждого элемента входного алфавита. На рис. 3.1 показана иллюстрация этого процесса. После построения кодового древа, двигаясь сверху вниз записываем, (определяем) элементы выходного алфавита. Словарь будет иметь следующий вид, в котором символы Z 1 ,Z 2 ,Z 3 ,Z 4 ,Z 5 ,Z 6 ,Z 7 ,Z 8 первой строки обозначают буквы входного алфавита и соответствующие им бинарные цепочки фиксированной длины, а бинарные цепочки переменной длины во второй строке – это буквы выходного алфавита

**Цель работы**

Разработка и реализация компьютерных технологий сжатия информации на основе методов кодирования и способов виртуализации в каналах передачи и хранения в виде готового программного продукта, приобретение базовых знаний построения моделей алгоритмов и процедур программных компонент таких технологий, освоение навыков интеллектуального анализа информационных объектов на уровне их «физического и виртуального» представления, оценки эффективности и конкурентоспособности технологии продукта.

**Этапы выполнения работы**

1. Разработка моделей алгоритмов и процедур программного компонента сжатия информационного объекта (файлы и другие логические структуры) согласно заданию.

2. Реализация программного интерфейса ввода-вывода файловых структур с внешних устройств (ВУ) в ОЗУ и обратно с учетом особенностей периферии.

3. Разработка логических схем моделей построения заголовков словарей и алгоритмы их реализации.

4. Реализация п.3 в виде элемента технологической цепочки общего программного компонента технологии сжатия.

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 <map>

#include <queue>

#include <vector>

#include <bitset>

#include <sstream>

#include <string>

#include <clocale>

#include <Windows.h>

using namespace std;

struct Node

{

    string data;

    int freq;

    Node\* left;// 0

    Node\* right;// 0

    Node(const string& data, int freq) :

        data(data),

        freq(freq),

        left(nullptr),

        right(nullptr)

    {}

    Node(Node\* left, Node\* right):

        data(""),

        freq(left->freq + right->freq), left(left), right(right)

    {}

};

struct CompareNodes{

    bool operator()(Node\* left, Node\* right)

    {

        return left->freq > right->freq;

    }

};

string readBinaryFile(const string& inputPath)

{

    ifstream input(inputPath, ios::binary);

    if (input.is\_open())

    {

        stringstream buffer;

        buffer << input.rdbuf();

        return buffer.str();

    }

    return "0";

}

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

    std::string binary;

    for (char c : text) {

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

    }

    return binary;

}

string binaryToString(const string& binary) {

    string text = "";

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

    {

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

        text += c;

    }

    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;

}

map<string, int> buildFrequencyTable(std::vector<std::string>& splittedBinary)

{

    map<string, int> frequencyTable;

    for (auto block : splittedBinary)

    {

        frequencyTable[block]++;

    }

    return frequencyTable;

}

Node\* buildHaffmanTree(const map<string, int>& frequencyTable)

{

    priority\_queue<Node\*, vector<Node\*>, CompareNodes> priorityQueue;

    for (auto& obj:frequencyTable)

    {

        priorityQueue.push(new Node(obj.first, obj.second));

    }

    if (priorityQueue.size() == 1)

    {

        Node\* root = new Node("", priorityQueue.top()->freq);

        root->left = priorityQueue.top();

        return root;

    }

    else

    {

        while (priorityQueue.size() > 1)

        {

            Node\* left = priorityQueue.top();

            priorityQueue.pop();

            Node\* right = priorityQueue.top();

            priorityQueue.pop();

            Node\* parent = new Node(left, right);

            priorityQueue.push(parent);

        }

        return priorityQueue.top();

    }

}

map<string, string> buildDict(Node\* root, string code = "")

{

    map<string, string> Dict;

    if (root->left)

    {

        map<string, string> leftDict = buildDict(root->left, code+"0");

        Dict.insert(leftDict.begin(), leftDict.end());

    }

    if (root->right)

    {

        map<string, string> rihgtDict = buildDict(root->right, code + "1");

        Dict.insert(rihgtDict.begin(), rihgtDict.end());

    }

    if (!root->data.empty())

    {

        Dict[root->data] = code;

    }

    return Dict;

}

string compress(map<string, string> dict, const vector<string>& splittedBinary)

{

    string compressedString = "";

    for (auto obj: splittedBinary)

    {

        compressedString += dict[obj];

    }

    return compressedString;

}

string decompress(Node\* rootHaffman, const string& compressed)

{

    Node\* currentNode = rootHaffman;

    string decompressed;

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

    {

        if (compressed[i] == '0')

        {

            currentNode = currentNode->left;

        }

        else

        {

            currentNode = currentNode->right;

        }

        if (!currentNode->data.empty())

        {

            decompressed += currentNode->data;

            currentNode = rootHaffman;

        }

    }

    return decompressed;

}

void printTree(Node\* root, int level = 0)

{

    if (root == nullptr)

    {

        return;

    }

    printTree(root->right, level + 1);

    for (size\_t i = 0; i < level; i++)

    {

        cout << "\t ";

    }

    if (root->data.empty())

    {

        cout << '(' << root->freq << ')';

    }

    else

    {

        cout << '[' << root->data << ']';

    }

    cout << '|' << level << '|' << endl;

    printTree(root->left, level + 1);

}

int main()

{

    setlocale(LC\_ALL,"Russian");

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    string inputPath = "input.txt";

    string inputFile = readBinaryFile(inputPath);//Считываем файл(1)

    cout << "Input file:\n" << inputFile << endl;

    string binaryFile = textToBinary(inputFile);//Приводим файл к бинарному виду(2)

    int bitsSplit;

    do{

        cout << "Vvedite kolichestvo bit dlia razbienia na segmenti: ";

        cin >> bitsSplit;

    } while (bitsSplit < 0);

    vector<std::string> splittedBinary = splitBinary(binaryFile, bitsSplit);//Разбиваем строку по n бит(3)

    map<string, int> frequencyTable = buildFrequencyTable(splittedBinary);//Формируем таблицу частотности(4)

    Node\* rootHaffman = buildHaffmanTree(frequencyTable);//Формируем дерево(5)

    cout << "Tablitsa chastotnosti: \n";

    for (auto cail : frequencyTable)

    {

        cout << cail.first << "\t" << cail.second << endl;

    }

    map<string, string> dict = buildDict(rootHaffman);//Формируем словарь(6)

    cout << "Slovarik: \n";

    for (auto cail : dict)

    {

        cout << cail.first << "\t\t" << cail.second << endl;

    }

    cout << "Derevo: \n";

    printTree(rootHaffman);

    string compressed = compress(dict, splittedBinary);

    cout << compressed << endl;

    int zeros = 0;

    if (compressed.length() % 8 != 0)

    {

        zeros = 8 - (compressed.length() % 8);

        compressed += string(zeros, '0');

    }

    string compressedBinString = binaryToString(compressed);

    ofstream output("output.bin", ios::binary);

    output.write(compressedBinString.c\_str(), compressedBinString.length());

    output.close();

    string compressedFile = readBinaryFile("output.bin");

    compressedFile = textToBinary(compressedFile);

    if (zeros != 0)

    {

        compressedFile.erase(compressedFile.end() - zeros, compressedFile.end());

    }

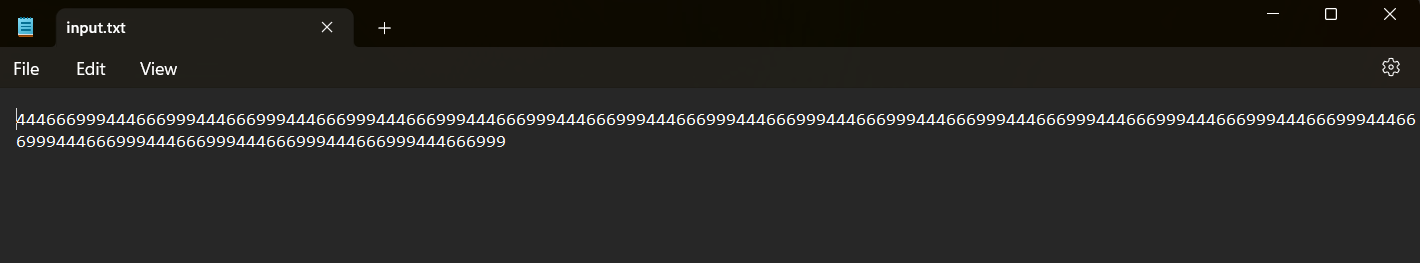
    string decompressed = decompress(rootHaffman, compressedFile);

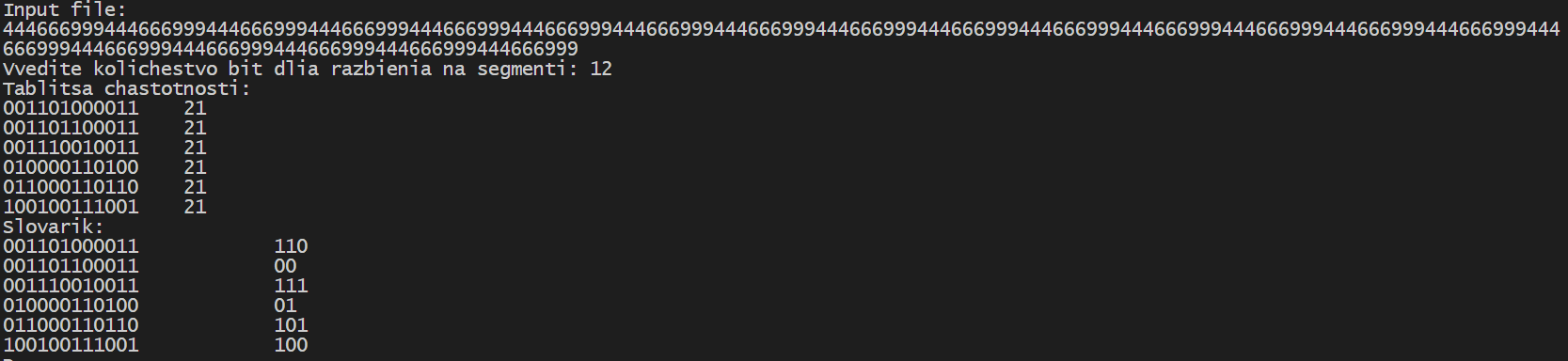
    cout << binaryToString(decompressed) << endl;

}

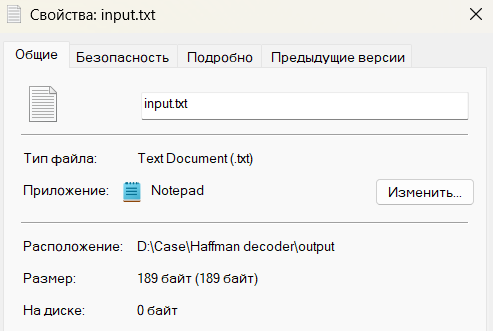
**Тестирование**

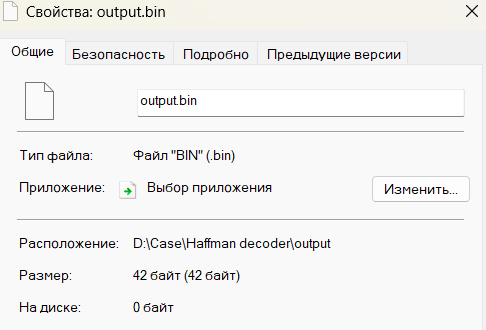
Исходный файл:



Работа программы:

**Результат:**

**До сжатия: После сжатия:**

****

**Вывод**:

Было разработано и реализовано компьютерная технология сжатия информации на основе методов кодирования и способов виртуализации в каналах передачи и хранения в виде готового программного продукта. Были приобретены базовые знания построения моделей алгоритмов и процедур программных компонент таких технологий.