МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №4

«Построение кодов Хаффмана»

Выполнил:

Студент 2 курса
группы ПО-9
Харитонович Захар Сергеевич

Проверила:

Глущенко Т. А.

Брест 2022

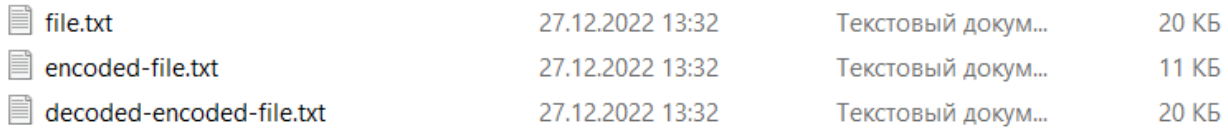
Задание.
Дан текстовый файл размером не менее 5 кбайт. Построить для данного текста коды Хаффмана. Написать программу для кодировки и раскодировки заданного файла. Указать размеры файла до и после сжатия алгоритмом Хаффмана.
Для наглядности текстовый файл можно создать самостоятельно и меньшего размера.

```
main.cpp
#include <iostream>
#include "huffman.h"

int main() {
    encode("file.txt");
    decode("encoded-file.txt");
    return 0;
}

huffman.h
#ifndef MAIN_CPP_HUFFMAN_H
#define MAIN_CPP_HUFFMAN_H

#include <bits/stdc++.h>

using namespace std;

struct node {
    char ch;
    int freq;
    node *left;
    node *right;
};

struct comp {
    bool operator()(node *a, node *b) {
        return a->freq > b->freq;
    }
};

void encode(string fileName);

void decode(string fileName);

void getHuffmanCodes(node *root, map<char, vector<bool>> &HuffmanCode,
vector<bool> code);

node *createNode(char ch, int freq, node *left, node *right);

void fillFrequenciesFromDecodedFile(map<char, int> &frequencies, string
fileName);

node *buildHuffmanTree(const map<char, int> &frequencies);

void fillFrequenciesFromEncodedFile(map<char, int> &frequencies, string
fileName);

#endif

huffman.cpp
#include "huffman.h"

void encode(string fileName) {
    map<char, vector<bool>> HuffmanCode;
    vector<bool> empty;
```

```
    map<char, int> frequencies;

    fillFrequenciesFromDecodedFile(frequencies, fileName);
    node *root = buildHuffmanTree(frequencies);
    getHuffmanCodes(root, HuffmanCode, empty);

    ifstream fin(fileName, ios::out | ios::binary);
    ofstream fout("encoded-" + fileName, ios::out | ios::binary);

    fout << frequencies.size() << " ";
    for (auto i: frequencies) {
        fout << (int) i.first << " " << i.second << " ";
    }
    fout << '\0';

    int count = 0;
    char buf = 0;
    while (!fin.eof()) {
        char c = fin.get();
        vector<bool> tmp = HuffmanCode[c];
        for (int n = 0; n < tmp.size(); n++) {
            buf = buf | tmp[n] << (7 - count);
            count++;
            if (count == 8) {
                fout << buf;
                count = 0;
                buf = 0;
            }
        }
    }

}

void getHuffmanCodes(node *root, map<char, vector<bool>> &HuffmanCode,
vector<bool> code) {
    if (root->left == nullptr && root->right == nullptr) {
        HuffmanCode[root->ch] = code;
        return;
    }
    if (root->left != nullptr) {
        code.push_back(0);
        getHuffmanCodes(root->left, HuffmanCode, code);
        code.pop_back();
    }
    if (root->right != nullptr) {
        code.push_back(1);
        getHuffmanCodes(root->right, HuffmanCode, code);
        code.pop_back();
    }
}

node *buildHuffmanTree(const map<char, int> &frequencies) {
    priority_queue<node *, vector<node *>, comp> pq;
    for (auto i: frequencies) {
        pq.push(createNode(i.first, i.second, nullptr, nullptr));
    }
    while (pq.size() != 1) {
        node *left = pq.top();
        pq.pop();
        node *right = pq.top();
        pq.pop();
        int totalFreq = right->freq + left->freq;
        pq.push(createNode('\0', totalFreq, left, right));
```

```
    }
    return pq.top();
}

node *createNode(char ch, int freq, node *left, node *right) {
    node *nd = new node;
    nd->ch = ch;
    nd->freq = freq;
    nd->left = left;
    nd->right = right;
    return nd;
}

void fillFrequenciesFromDecodedFile(map<char, int> &frequencies, string
fileName) {
    ifstream fin(fileName, ios::out | ios::binary);

    char c = fin.get();
    while (!fin.eof()) {
        frequencies[c]++;
        c = fin.get();
    }

    fin.close();
}

void decode(string fileName) {
    map<char, vector<bool>> HuffmanCode;
    vector<bool> empty;
    map<char, int> frequencies;


    fillFrequenciesFromEncodedFile(frequencies, fileName);
    node *root = buildHuffmanTree(frequencies);
    getHuffmanCodes(root, HuffmanCode, empty);

    ifstream fin(fileName, ios::in | ios::binary);
    ofstream fout("decoded-" + fileName, ios::out | ios::binary);

    while (fin.get() != '\0');

    char byte;
    bool curBit;
    node *ptrNode = root;
    while (!fin.eof()) {
        byte = fin.get();
        for (int i = 0; i < 8; ++i) {
            curBit = byte & (1 << (7 - i));
            if (curBit == 0) ptrNode = ptrNode->left;
            else ptrNode = ptrNode->right;
            if (ptrNode->left == nullptr && ptrNode->right == nullptr) {
                fout << ptrNode->ch;
                ptrNode = root;
            }
        }
    }

    fout.close();
    fin.close();
}

void fillFrequenciesFromEncodedFile(map<char, int> &frequencies, string
fileName) {
```

```
    ifstream fin(fileName);
    int frequency, ch, count; fin >> count;
    for (int i = 0; i < count; ++i) {
        fin >> ch >> frequency;
        frequencies[(char) ch] = frequency;
    }
    fin.close();
}
```



Контрольные вопросы

1. Что такое сжатие без потерь?

Сжатие данных без потерь — класс алгоритмов сжатия данных, при использовании которых закодированные данные однозначно могут быть восстановлены с точностью до бита, пикселя и т.д. При этом оригинальные данные полностью восстанавливаются из сжатого состояния.

2. Опишите алгоритм построения кодов Шеннона-Фано для сжатия данных.

Код Шеннона — Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на

два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

3. Что такое префиксные коды, являются ли в данных алгоритмах коды префиксными и для чего они используются?

Префиксный код — код со словом переменной длины, имеющий такое свойство: если в код входит слово a, то для любой непустой строки b слова ab в коде не существует. В алгоритмах Хаффмана и Шеннона-Фано получаемые коды являются префиксными. Эти коды обеспечивают однозначное декодирование принятых кодовых слов без введения дополнительной информации для их разделения.

4. Благодаря каким принципам происходит сжатие данных в указанных алгоритмах?

Данная категория алгоритмов способна сжимать информацию за счет неравномерности частот, с которыми разные символы могут встречаться в сообщении. Так, часто встречающийся символ кодируется кодом меньшей длины, а редко встречающийся — кодом большей длины.

5. Укажите недостатки указанных кодов, средние коэффициенты сжатия для указанных алгоритмов.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого декодер должен знать таблицу частотностей, которой пользовался кодер, что может свести на нет все усилия по сжатию сообщения. Необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению. Во-вторых, например, для двоичного источника, непосредственное применение кода Хаффмана бессмысленно. Текстовый файл может быть сжат на значение около 50% (обычно 40-45%)/

Алгоритм Шеннона-Фано не является оптимальным.

Текстовый файл может быть сжат на значение около 28-30% (меньше 50%)