МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы сжатия

Студент гр. 7383 _____ Александров Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы.

Познакомиться с алгоритмом сжатия – кодом Фано-Шеннона – и его реализацией на языке программирования C++.

Постановка задачи.

Кодирование: Фано-Шеннона.

Реализация задачи.

Для решения поставленной задачи в работе были использованы класс Main, ряд функций и две структуры для работы с алгоритмом.

Структура CodeTree:

```
struct CodeTree {
   Symbol s;
   CodeTree *parent;
   CodeTree *left;
   CodeTree *right;
};
```

Структура Symbol:

```
struct Symbol {
   char c;
   int weight;
};
```

В классе Main определяется функции считывания текста:

1) void fileRead() – из файла;
2) void consoleRead() – из консоли.

Пользователю предлагается либо ввести текст, либо указать текстовый файл, в котором он находится.

Функции для работы с бинарным деревом в алгоритме:

- bool symbol_greater(const Symbol &l, const Symbol &r) – сравнивает количество найденных символов;

- CodeTree *make_leaf(const Symbol &s) создает создает лист без потомков в дереве;
- CodeTree *make_node(int weight, CodeTree *left, CodeTree *right) создает элемент дерева;
- bool is_leaf(const CodeTree *node) проверяет, является ли элемент листом;
- bool is_root(const CodeTree *node) проверяет, является ли элемент корнем;
- static void fill_symbols_map(const CodeTree *node, const CodeTree **symbols_map) заполняет 2Д массив структуры CodeTree;
- char *encode(const CodeTree *tree, const string &message) получает на вход созданное построенное дерево и исходную строку, возвращает закодированный массив;
- char *resize(char *prevArr, int sizeOfCode) изменяет первоначальный размер закодированного массива;
- void destroy(CodeTree *tree) удаляет дерево;
- CodeTree *fanno_shannon(const string &message), CodeTree *fanno_shannon(const Symbol *symbols, int len), CodeTree *fanno_shannon(const Symbol *symbols, int l, int r, int sum) – 3 перегруженные функции строят дерево по введенной строке;
- static int middle(const Symbol *symbols, int l, int sum, int &lsum, int &rsum) делит символы полученной строки на 2 части, сумма которых максимально близка друг к другу.

Тестирование программы.

Программа собрана и проверена в операционных системах Xubuntu 18.04 с использованием компилятора g++ и Windows с использованием MinGW. В других ОС и компиляторах тестирование не проводилось. Тесты находятся в приложении А.

Вывод.

В ходе лабораторной работы были получены основные навыки программирования алгоритма Фано-Шеннона на языке C++. Результатом стала программа, которая кодирует введенный текст.

ПРИЛОЖЕНИЕ А
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Таблица 1 – Тестирование программы

Input	Output
In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal.	11111110100100011110010 11011001101001100100110 01111101011111111111101 10111100110110100100100 01110111001110101011010 01111000111010000100011 01100100000111111111011 10011100001101110111000 01101010010000011010101 01001101001111000010110 10000101101001000001101 01101001010100001111000 10100011110010001110010 01011110000001101111010 00111000110110010111010 00010000010001011101001 10011110000001101111010 00111000110110010111010 11101000001101001110100 00100011011001010010001 10101011011111010101101 10100101101000010110100 11000001000100011101100 01000011101010000111000 11101101101100010111010 00010000011000011010111 00011011110100011100011 01100101101011110110100

	0101100100111000011010 1 001000000110011100011101 111011100101110100000 11 001110001101110010111 01 111011011001011101000 01 000001000110001010110 10 011110000000101111011 11 1111000110101111111 1011
The Shannon–Fano algorithm doesn't always generate an optimal code.	111110100101110011111 11 100100000100100110010111 001110110000001001100010 001010011001011010111 10 101100010011011000111 01 001100111110000101110 00 100000100010100111011 00 011110011000000111001011 101001111011100010000 11 100100000100010110111101 100010101101100000101000 011101110110110100111 11 1010

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include "main.h"

using namespace std;

void Main::menu() {
    cout << "1. Enter a binary codeTree from the console" << endl;
    cout << "2. Enter a binary codeTree from the text file" << endl;
    cout << "0. Exit" << endl;
}

void Main::fileRead() {
    string fileName;
    string message;
    cout << "What`s the file name?" << endl;
    cin >> fileName;
    cout << "--------------" << endl;
    cout << "Reading from " << fileName << endl;
    cout << "--------------" << endl;
    ifstream inFile;
    inFile.open(fileName);
    if (!inFile) {
        cout << "Cannot find this file" << endl;
        cout << endl;
        return;
    }
    ofstream outFile;
    outFile.open("result.txt");
    cout << "--------" << endl;
    while (!inFile.eof()) {
        getline(inFile, message);
        if (message.empty()) continue;
        CodeTree *ct = fanno_shannon(message);
        char *resultCode = encode(ct, message);
        cout << "Your code " << endl;
        cout << resultCode << endl;
        outFile << resultCode << endl;
    }
    cout << "--------" << endl;
    inFile.close();
    outFile.close();
}
```

```
void Main::consoleRead() {
    string message;
    cout << "Enter string" << endl;
    cin.ignore();
    getline(cin, message);
    if (message.empty())
        return;
    CodeTree *ct = fanno_shannon(message);
    char *resUlt = encode(ct, message);
    cout << "Your code " << endl;
    cout << resUlt << endl;
}

int main() {
    Main main;
    while (true) {
        main.menu();
        cin >> main.choice;
        switch (main.choice) {
            case 1:
                main.consoleRead();
                break;
            case 2:
                main.fileRead();
                break;
            case 0:
                exit(1);
        }
    }
}
```

main.h

```
#include "fs.h"

class Main {
public:
    Main() {}

    unsigned int choice;

    void fileRead();

    void consoleRead();

    void menu();
};
```

codetree.cpp

```
#include "codetree.h"
```

```
#include <climits>
#include <cstring>
#include <string>
#include <vector>
#include <iostream>

using namespace std;


bool symbol_greater(const Symbol &l, const Symbol &r) {
    return l.weight > r.weight;
}

CodeTree *make_leaf(const Symbol &s) {
    return new CodeTree{s, nullptr, nullptr, nullptr};
}

CodeTree *make_node(int weight, CodeTree *left, CodeTree *right) {
    Symbol s{0, weight};
    return new CodeTree{s, nullptr, left, right};
}

bool is_leaf(const CodeTree *node) {
    return node->left == nullptr && node->right == nullptr;
}

bool is_root(const CodeTree *node) {
    return node->parent == nullptr;
}

static void fill_symbols_map(const CodeTree *node, const CodeTree
**symbols_map);

char *encode(const CodeTree *tree, const string &message) {
    unsigned int firstLength = 1000;
    char *code = new char[firstLength];
    const CodeTree **symbols_map = new const CodeTree *[UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols_map[i] = nullptr;
    }
    fill_symbols_map(tree, symbols_map);
    int len = message.size();
    unsigned int index = 0;
    char path[UCHAR_MAX];
```

```
    for (int i = 0; i < len; ++i) {
        const CodeTree *node = symbols_map[message[i] - CHAR_MIN];
        int j = 0;
        while (!is_root(node)) {
            if (node->parent->left == node)
                path[j++] = '0';
            else
                path[j++] = '1';
            node = node->parent;
        }
        while (j > 0) {
            if (index >= firstLength) {
                code = resize(code, firstLength);
            }
            code[index++] = path[--j];
        }
    }
    code[index] = 0;
    delete[] symbols_map;
    return code;
}

char *resize(char *prevArr, unsigned int &sizeOfCode) {
    unsigned int newSize = sizeOfCode * 2;
    char *newArr = new char[newSize];
    for (int i = 0; i < sizeOfCode; i++) {
        newArr[i] = prevArr[i];
    }
    sizeOfCode = newSize;
    return newArr;
}

void destroy(CodeTree *tree) {
    if (tree == nullptr) return;
    destroy(tree->left);
    destroy(tree->right);
    delete tree;
    tree = nullptr;
}

void fill_symbols_map(const CodeTree *node, const CodeTree **symbols_map)
{
    if (is_leaf(node))
        symbols_map[node->s.c - CHAR_MIN] = node;
```

```
    else {
        fill_symbols_map(node->left, symbols_map);
        fill_symbols_map(node->right, symbols_map);
    }
}
```

codetree.h

```
#include <string>
#include <vector>

using namespace std;


struct Symbol {
    char c;
    int weight;
};

bool symbol_greater(const Symbol &l, const Symbol &r);

struct CodeTree {
    Symbol s;
    CodeTree *parent;
    CodeTree *left;
    CodeTree *right;
};

char *resize(char *prevArr, unsigned int &sizeOfCode);

CodeTree *make_leaf(const Symbol &s);

CodeTree *make_node(int weight, CodeTree *left, CodeTree *right);

bool is_leaf(const CodeTree *node);

bool is_root(const CodeTree *node);

char *encode(const CodeTree *tree, const string &message);

void destroy(CodeTree *tree);
```

fs.cpp

```
#include "fs.h"
#include <algorithm>
#include <climits>
#include <string>
#include <cstring>

using namespace std;
```

```
static int middle(const Symbol *symbols, int l, int sum, int &lsum, int
&rsum);

CodeTree *fanno_shannon(const Symbol *symbols, int l, int r, int sum) {
    if (l >= r) return nullptr;
    if (r - l == 1) return make_leaf(symbols[l]);
    int lsum, rsum;
    int m = middle(symbols, l, sum, lsum, rsum);
    CodeTree *ltree = fanno_shannon(symbols, l, m + 1, lsum);
    CodeTree *rtree = fanno_shannon(symbols, m + 1, r, rsum);
    CodeTree *node = make_node(sum, ltree, rtree);
    ltree->parent = node;
    rtree->parent = node;
    return node;
}

CodeTree *fanno_shannon(const Symbol *symbols, int len) {
    int sum = 0;
    for (int i = 0; i < len; ++i)
        sum += symbols[i].weight;
    return fanno_shannon(symbols, 0, len, sum);
}

CodeTree *fanno_shannon(const string &message) {
    Symbol symbols[UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols[i].c = i + CHAR_MIN;
        symbols[i].weight = 0;
    }
    int size = message.size();
    for (int i = 0; i < size; ++i)
        symbols[message[i] - CHAR_MIN].weight++;
    std::sort(symbols, symbols + UCHAR_MAX, symbol_greater);
    int len = 0;
    while (symbols[len].weight > 0 && len < UCHAR_MAX) len++;
    return fanno_shannon(symbols, len);
}

int middle(const Symbol *symbols, int l, int sum, int &lsum, int &rsum) {
    int m = l;
    lsum = symbols[m].weight;
    rsum = sum - lsum;
    int delta = lsum - rsum;
    while (delta + symbols[m + 1].weight < 0) {
        m++;
        lsum += symbols[m].weight;
        rsum -= symbols[m].weight;
        delta = lsum - rsum;
    }
    return m;
```

```
}
```

fs.h

```
#include "codetree.h"

#pragma once

#include <string>

CodeTree *fanno_shannon(const std::string &message);

CodeTree *fanno_shannon(const Symbol *symbols, int len);
```

Makefile

```
CXX=g++
RM=rm -f
LDFLAGS=-g -Wall

SRCS=main.cpp codetree.cpp fs.cpp
OBJS=$(subst .cpp,.o,$(SRCS))

all: main

main: $(OBJS)
	$(CXX) $(LDFLAGS) -o main $(OBJS)

main.o: main.cpp main.h

codetree.o: codetree.cpp codetree.h

fs.o: fs.cpp fs.h

clean:
	$(RM) $(OBJS)

distclean: clean
	$(RM) main
```