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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Кодирование и декодирование

Студент гр. 9382 _____ Демин В.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с базовыми методами кодирования и декодирования.

Задание.

Вариант 3.

В вариантах заданий 1-ой группы (кодирование и декодирование) на вход подаётся файл с закодированным или незакодированным содержимым. Требуется раскодировать или закодировать содержимое файла определённым алгоритмом.

Кодирование: статическое Хаффмана

Алгоритм.

1. Формирование алфавита символ, в этом шаге происходит подсчет весов каждого символа в тексте.
2. На основе полученного алфавита создается бинарное дерево. Сначало создается массив узлов. В каждом узле хранится вес и символ из алфавит. Далее берется два наименьших узла по весу. Объединям их в дерево, корнем которого будет узел с суммой весов данных узлов. Массив узлов сортируется по убыванию. Алгоритм повторяется. Пока в массиве не останется один узел, которой будет корнем бинарного дерева символов из текста.
3. Из полученного дерево формируются коды символов. Идет поиск узла в котором хранится символ алфавита. Запоминается путь до него, если веть левая, то 0, если правое, то 1.
4. Далее происходи кодирование текста, каждый символ заменяется на его код.

Функции и структуры данных.

1. Бинарное дерево

   Используется в кодирование Хаффмана, с помощью дерева формируется коды символов. Содержит в себя поля: weight – вес узла, ch – символ, left – левое поддерево, right – правое поддерево.

Методы: node *cons(node *a, node *b) – из полученных узлов формирует новый узел, поддеревьями которого будут полученные узлы. Узел у которого вес больше станет правым поддеревом.

2. Кодирование
    a. void enter(string &str) - считывает текст для кодирования из файла или из консоли, и записвывает его в строку str.
    b. void formAlpabet(string const &str, vector<pair<char, int>> &alphabet)- функция формирования алфавита из строки. Алфавит представлен вектором пар символа и его веса.
    c. node *makeBinaryTree(vector<pair<char, int>> alphabet) – функция, которая возвращает корень бинарного дерева созданное из элементов алфавита(принимаемоего вектора пар весов и символов).
    d. void makeCode(node *tree, char ch, string &code, bool &flag) – функция формирования кода символа из бинарного дерева, функция рекурсивная. Полученный код сохраняется в строку, если символ есть в дереве, то полученный флаг останется false.
    e. vector<pair<char, string>> coding(string &str) – основная функция кодирования, которая принимает текст для кодирования, а возвращает вектор пар символа и его кода.
    f. int write(string text, vector<pair<char, std::string>> code) – функция записи текста в закодированном виде.

Выводы.

В данной задаче была создана программа для кодирования Хаффмана.

Тестирование.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Modern world is becoming smaller all the time. Every day distances between different countries seem less. For this reason it's becoming more and more important to know different languages, especially English. One billion people speak English today. That's about 20% of the world's population. 400 million people speak English as their first language. For the other 600 million people it's either a second language or a foreign language. English is the first language in the United Kingdom, the United States of America, Australia and New Zealand. It is one of the official languages in Canada, the Irish Republic and the Republic of South Africa. As a second language English is spoken in more than 60 countries. It is used by the government,	111011010101010 11000001110010111110001 10100101111000010011000 11010100111111000110000 01111010101111100101010 01110010110011111 110010000010001 00001110011010000010000 10110010000110100011001 00101011110000000011101 10001111100111100000011 10010011101100110 010001001110110 01100101001110100100001 01111110100001111100011 00000010000110100000000 10111100110010101001100 1001100001110000 010110100110111 10101011111111011010011 10010100000111110011100 00000111100110001000000 11101110011101101110110 000101111001100 100011011010011 11101110000010000011101 01101111010100100111010 00101111100011000001111 10101011111001010101111 00101101111000101 111000001101000 10110110011011111000101	

	businessmen and universities. English is the language of politics and diplomacy, science and technology, business and trade, sport and pop music.	1110000011010101111001 1111001011110001001000 1011010011001000101110 11101000101101010 011010110011001 0101001101001100001110 0000101101001100010100 0101110010111111100010 0100000111111011111000 00111111110000111 101101010000010 0010100111011000111111 0111001000101010011101 1010011101001111011101 1000110001100101000100 01010100101101111 011111000001011 1111000100001100111111111 1000010001110100011000 1111110111001000101010 0111011011100100010101 1001000100111000 111011010011110 1001101100001001110100 1011111010000011000101 1111110100110111010101 0001101110011110011100 10110011011001000 110100011000110 1001011110000100110011 1010010111110111110010 1111110111111001010000 1 0010100101101100111011 0100111100000110 110011011110111 1001010001000101010010 1101111011111000001011 1 1110001000011001111111 1 0000100011101000110001 111110111001000 101010011101101	

		1101000011111001000110 1000101011100110100110 1010111000111010011000 1010001011100101111111 00010010000001110 110111011000010 1111001100100011010001 1001010100011010001110 0110111011001001101100 1101111011110010100010 00101010010110111 101111100000101 1111100010000110101001 0011101001011111000010 1001000110100011100110 1000110011100011110101 01101101100110001 010001011100101 1111110001001000011001 0111100110100011010011 0010111100000101010010 1011110001010001011100 10111111100010010 000001110001111 1101110010001010100111 0110111010100111110010 0011010001101001101010 1110001110100110001010 00101110010111111 100010010000110 1010101111001000110100 0110100111111101110100 1000000110011011101010 1110101011100100110001 01111100111011111 001000110100011 0100111111101110100100 0000110011010011111001 0010000100000011111001 0110011011011101011111 0000011100101011	

		110110001110111 1101110101111111011101 0011100100000101010100 0110100010110110011011 1010100000000110101101 1101010010001000 001010001011011 0000111011000111101010 0110101001111100101101 1000110010110011011001 0001101000110010110011 01001101010111101 101010000010110 0010100010111001011111 1100010010000011111010 1010111101110110100100 0101110000110010001110 11111001000110100 011000111101111 0010100111011011101110 1101100011111011111001 1000010101011110111010 0010110110011001000110 1000110111011011 000111110111111 0011000010101011110111 0010110011011010011111 0010111111101000110111 0111010111001101110010 10111101100000111 011101011011111 0100011001110001111010 1011011011001100010100 0101110010111111100010 0100001100011111101110 01000101010011101 101110101001111 1001111111110010111101000 0000101111010101011110 1111000101111000001100 1000110110001011110111	

		0110010011011110 111101010111111 1101101001110010100000 1110011101100011110101 0011010100111110111110 1110000110011000110010 0111011001000110 100011010010010 1001111000001110010111 1110000010110100111011 1110001100111111010111101 0101100001110111111100 0001011110100010 110110011011111 1101110100011110000011 1000111101001001010000 0111001110001111110111 0 0100010101001110110111 0101001111100100 011010001100010 1000101110010111111100 0100100001100101100110 1101111100101001010100 1001010111101011111010 00101101100110011 001010111110001 0010111110010001111011 0011101110111110011111 1 1011010000101111110100 0110100010110110011001 0000011110101101 101101010010010 1100101001110111011111 0001100111111011110101 0110000111011111010001 0110110011001001110010 00011000001110111 110011111111001 0111100010011010001011 0110011011111001011111 1 0110111100111111010111 01	

		0111101001110 The size of the text in bytes before encoding:7072 The size of the text in bytes after encoding:3936 Intermeiate values write to file "intermeiate.txt" Process finished with exit code 0	
2.	Hello,world!	0100110101000011111100001111101100110 1 The size of the text in bytes before encoding:96 The size of the text in bytes after encoding:37	
3.	aaaaaaabbbbbb	1111111000000 the size of the text in bytes before encoding:104 The size of the text in bytes after encoding:13	
4.	1111111000000	1111111000000 The size of the text in bytes before encoding:104 The size of the text in bytes after encoding:13	
5.	abcfgklmnoprctyuio	101010111111100 010010010001100000001 1 100110011111110100010 1 1110011101110 The size of the text in bytes before encoding:144 The size of the text in bytes after encoding:72	

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
//
// Created by vikto on 20.11.2020.
//
// add logger

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <conio.h>

using namespace std;
typedef struct nodeTree node;

struct nodeTree {//узел бинарного дерева
    nodeTree(int w, char c) {
        this->weight = w;
        this->ch = c;
        this->left = nullptr;
        this->right = nullptr;
    }

    nodeTree() {
        weight = 0;
        ch = -1;
        left = nullptr;
        right = nullptr;
    }

    int weight;//вес узла
    char ch;//символ в узле
    node *left;//левое поддерево
    node *right;//правое поддерево
};
```

```
    //класс для вывода промежуточных значений
    class logger {
        ofstream file;

    public:
        logger() {
            file.open("intermeiate.txt");
            if (!file.is_open())
                throw runtime_error("file open failure");
        }

        void logAlphabet(vector<pair<char, int>> &alphabet) {
            for (auto it = alphabet.begin(); it != alphabet.end();
++it) {
                file << "Symbol: " << it->first << " -> Weight:" << it-
>second << "\n";
            }
        }

        void logTree(node *tree, int level = 0) {
            if (tree) {
                logTree(tree->left, level + 1);
                for (int i = 0; i < level; i++) file << "   ";
                if (tree->ch == -1) {
                    file << '<' << "\n";
                } else {
                    file << tree->ch << "\n";
                }
                logTree(tree->right, level + 1);
            }
        }

        void logCode(vector<pair<char, string>> &code) {
            for (auto it = code.begin(); it != code.end(); ++it) {
                file << "Symbol: " << it->first << " -> Code:" << it-
>second << "\n";
            }
        }
```

```
    //закрытие файла
        ~logger() {
            std::cout << "\nIntermeiate values write to file
\"intermeiate.txt\"";
            file.close();
        }
    };

    static logger *log = new logger();

    //функция для объединения двух узлов
    node *cons(node *a, node *b) {
        node *c = new node();
        c->weight = a->weight + b->weight;
        c->ch = -1;
        if (a->weight > b->weight) {//узел у которого вес больше станет
правым поддеревом
            c->right = a;
            c->left = b;
        } else {
            c->right = b;
            c->left = a;
        }
        return c;
    }

    //ввод текста для кодирования
    void enter(string &str) {
        int type = 0;
        cout << "Enter from console -1\n";
        cout << "Enter from file -2\n";
        cin >> type;

        ifstream file;
        string name;
        string temp;
```

```
        switch (type) {
            case 1:
                std::cin.ignore(32767, '\n');
                getline(cin, str);
                break;
            case 2:
                cout << "Enter file name\n";
                cin >> name;
                file.open(name);
                if (file.is_open()) {
                    while (file.good()) {
                        getline(file, temp);
                        str.append(temp);
                    }
                } else cout << "Unable to open file";
                break;
            default:
                cout << "incorrect type\n";
        }

    }

    //формирование алфавит текста для кодирования
    //считает вес символа
    void formAlpabet(string const &str, vector<pair<char, int>>
&alphabet) {

        bool flag = false;
        for (int i = 0; i < str.size(); ++i) {
            flag = false;
            for (auto it = begin(alphabet); it != end(alphabet); ++it)
{
                if (str[i] == it->first) {
                    flag = true;
                    it->second++;
                    break;
                }
            }
```

```
            if (flag) {
                continue;
            }
            alphabet.emplace_back(make_pair(str[i], 1));
        }
    };

    //  для сортировки алфавита
    bool compair(pair<char, int> a1, pair<char, int> a2) {
        return a1.second > a2.second;
    }

    // для сортировки узлов дерева
    bool compairNode(node *a1, node *a2) {
        return a1->weight > a2->weight;
    }

    //создает бинарное дерево из алфавита
    node *makeBinaryTree(vector<pair<char, int>> alphabet) {
        vector<node *> tree;
        for (auto it = alphabet.begin(); it != alphabet.end(); ++it) {
            tree.push_back(new node(it->second, it->first));
        }
        sort(tree.begin(), tree.end(), compairNode);
        while (tree.size() != 1) {
            tree.push_back(cons(tree[tree.size() - 2], tree[tree.size()
- 1]));
            sort(tree.begin(), tree.end(), compairNode);
            tree.pop_back();
            tree.pop_back();
        }

        return tree[0];
    }

    //формирует код символа из бинарного дерева
    void makeCode(node *tree, char ch, string &code, bool &flag) {
        if (flag) {
```

```
        if (tree != nullptr) {
            if (tree->ch == ch) {
                flag = false;
                return;
            }
            code += "0";
            makeCode(tree->left, ch, code, flag);
            if (flag) {
                code.pop_back();
            } else { return; }
            code += "1";
            makeCode(tree->right, ch, code, flag);
            if (flag) {
                code.pop_back();
            }
        }
    }
}
//Основная функция кодирование в не входит
//формирование алфавита
//формирование бинарного дерева
//создание кодов символов из дерева
vector<pair<char, string>> coding(string &str) {
    vector<pair<char, int>> alphabet;
    vector<pair<char, string>> alphabetWithCode;
    formAlpabet(str, alphabet);
    log->logAlphabet(alphabet);
    sort(alphabet.begin(), alphabet.end(), compair);
    node *binaryTree = makeBinaryTree(alphabet);
    log->logTree(binaryTree);
    string code;
    bool flag = true;
    for (auto it = begin(alphabet); it != end(alphabet); ++it) {
        makeCode(binaryTree, it->first, code, flag);
        alphabetWithCode.emplace_back(make_pair(it->first, code));
        code = "";
        flag = true;
    }
```

```
        return alphabetWithCode;
    }
    //запись полученного текста
    int write(string text, vector<pair<char, std::string>> code) {
        int type = 0;
        cout << "Print to console -1\n";
        cout << "Print to file -2\n";
        cin >> type;
        ofstream file;
        string nameF;
        unsigned int sizeHuffman = 0;
        unsigned int size = 8* text.size();
        switch (type) {
            case 1:
                for (auto ch = text.begin(); ch != text.end(); ++ch) {
                    for (auto it = code.begin(); it != code.end();
++it) {
                        if (it->first == *ch.base()) {
                            cout << it->second;
                            sizeHuffman += it->second.size();
                        }
                    }
                }
                cout << "\nThe size of the text in bytes before
encoding:" << size << "\n";
                cout << "The size of the text in bytes after encoding:"
<< sizeHuffman << "\n";
                break;
            case 2:
                cout << "Enter file name\n";
                cin >> nameF;
                file.open(nameF);
                if (file.is_open()) {
                    for (auto ch = text.begin(); ch != text.end();
++ch) {
                        for (auto it = code.begin(); it != code.end();
++it) {
                            if (it->first == *ch.base()) {
```

```
                                file << it->second;
                                sizeHuffman += it->second.size();
                            }
                        }
                    }
                    file << "\nThe size of the text in bit before encoding:" << size << "\n";
                    file << "The size of the text in bit after encoding:" << sizeHuffman << "\n";
                } else {
                    cout << "file don't open\n";
                }
                break;
            default:
                cout << "error";
        }

        return 0;
    }


    int main() {
        string str;
        enter(str);
        vector<pair<char, string>> code = coding(str);
        log->logCode(code);
        write(str, code);
        delete log;
        return 0;
    }
```