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: Метод Хаффмана. Статический.

Студент гр. 9304 _____ Боблаков Д.С.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы

Изучить методы кодирования и декодирования. Научиться реализовывать различные алгоритмы кодирования и декодирования информации.

Задание

Вариант 4.

Метод Хаффмана. Статический.

Программа должна иметь следующие ключи запуска: encode (включить режим«кодирование»), decode (включить режим «декодирование»), file (указать названиевходного файла, иначе ожидается ввод с консоли), o (указать название выходного файла,иначе на консоль), debug (включить режим отладки). Файл, полученный кодированием,должен быть расшифрован программой обратно.

Описание алгоритма работы

Сначала алгоритм считывает и записывает из CLI ключи и их аргументы в структуру config. Затем проходится проверка на корректность введенных данных. В случае некорректного ввода данных программа завершит свою работу с выводом сообщения об ошибке. Далее если был введен ключ --encode программа сначала считывает данные из файла или из консоли, а затем создает вектор с узлами для будущего дерева. Затем строится дерево Хаффмана из этих узлов. После этого программа создает файл, в который она записывает получившиеся пары из символа и его кода. Далее программа создает новый файл и записывает в него преобразованный текст в бинарном виде.

При запуски в режиме декодирования (--decode) программа считывает из бинарного файла последовательность нулей и единиц, а затем считывает из

другого файла пары ключей и значения символов. Затем в цикле с помощью итератора программа проходится по бинарному файлу и при совпадении ключа и кода в новый файл записывается результат декодирования.

При запуске программы с ключом --help программа выводит подсказку для корректного запуска программы.

Разработанный код см. в приложении А.

Формат входных и выходных данных

Программа не принимает входные данные во время выполнения программы. Однако программа принимает данные во время запуска программы через CLI (Command Line Interface). Данный интерфейс имеет следующие ключи: -e (--encode) – активация режима кодирования, -d (--decode) [<имя файла>] – активация режима декодирования, -f (--file) [<имя файла>] – имя файла из которого будет считаны исходные данные, -o (--output) [<имя файла>] – имя файла с выходными данными, -D (--debug) – активация режима отладки, -h (--help) – вызов подсказки. В случае некорректного ввода данных программа выведет следующую строку: «Error: incorrect input».

На выходе программа выведет в два файла (по умолчанию pairs.txt и out.txt) получившиеся пары (ключ — значение) и результат кодирования или декодирования.

Рисунок 1 – Пример работы программы.

Описание основных структур данных и функций

Класс Node

Данный класс является узлом для дерева Хаффмана.

Структура Config

Данная структура отвечает за конфигурацию параметров запуска программы.

Структура Pair

Данная структура является узлом словаря, где есть ключ code и значение symbol.

void print_help()

Данная функция печатает подсказку.

std::string load_from_file(const std::string& filename)

Данная функция считывает из файла текст для дальнейшей его обработки.

void printTree(std::shared_ptr<Node> node, int n =0)

Данная функция печатает дерево Хаффмана.

bool comp(std::shared_ptr<Node> &left, std::shared_ptr<Node> &right)

Данная функция передается как аргумент для сортировки в std::sort().

std::vector<std::shared_ptr<Node>> count_symbols(const std::string& string) Данная функция считает количество встречнных одинаковых символов и возвращает вектор с узлами Node.

std::shared_ptr<Node> consTree(const std::shared_ptr<Node>& a, const std::shared_ptr<Node>& b) Данная функция объединяет узлы Node и возвращает указатель на дерево.

std::shared_ptr<Node> make_tree(std::vector<std::shared_ptr<Node>>& vector) Данная функция строит дерево Хаффмана и возвращает указатель на его голову.

void make_pairs(std::shared_ptr<Node>& head, std::vector<Pair>& pairs,std::string string) Данная функция проходится по дереву и каждому листу присваивает код. Пару код – символ записывает в вектор.

void unload_pairs(const std::vector<Pair>& pairs, const std::string& filename= "pairs.txt") Данная функция записывает в файл пары, полученные на предыдущем шаге.

void encode(const std::string string, std::vector<Pair> pairs, std::string filename) Данная функция записывает в файл закодированный текст.

std::vector<Pair> read_pairs(const std::string& filename)

Данная функция считывает из файла пары значений (ключ — символ) и записывает их в вектор, который затем возвращает.

void decode(std::string encodedText, const std::string& decoded, const std::vector<Pair>& codes) Данная функция проходится по тексту из нулей и единиц и по полученным ключам декодирует текст.

void process(const Config& config, int argc , char** argv)

Данная функция запускается после работы CLI. Она является основной, то есть в ней прописана основная логика работы программы.

Тестирование

Тестирование программы проводится с помощью bash-скрипта tests_script. Для запуска тестирования необходимо выполнить команду ./tests_scripts, предварительно собрав программу с помощью команды make.

Для тестирования было написано 10 тестов, первые 8 из которых являются корректными для программы. Последние 2 теста являются некорректными данными для программы.

Результаты тестирования см. в приложении Б.

Выводы

Были изучены различные методы кодирования и декодирования информации. Была реализована программа, которая кодирует входящий текст по алгоритму Хаффмана, а затем декодирует его. Для данной программы был реализован CLI.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

```
#include <iostream>
#include <iostream>
#include <memory>
#include <string>
#include <getopt.h>
#include <fstream>
#include <vector>
#include <algorithm>


class Node{
public:
    char c{};
    int count{};
    std::string string;
    std::shared_ptr<Node> left {nullptr};
    std::shared_ptr<Node> right {nullptr};

};

struct Config{

    int encode;
    int decode;
    std::string pairs;
    int debug;
    int from_file;
    std::string ifile;
    std::string ofile;
};
```

```
    struct Pair{
        std::string symbol;
        std::string code;
    };


    void print_help(){
        std::cout<<"to encode: -e -f input_file\nto decode: -d
pairs_file -f binary_file [-o <file_name> to save output]\n";
    }

    std::string load_from_file(const std::string& filename){
        std::string data;
        std::string string;
        std::ifstream file;
        file.open(filename);
        while(getline(file, string)){
            data+=string;
        }
        file.close();
        return data;
    }

    void print_vector(const std::vector<std::shared_ptr<Node>>&
vector)
    {
        for (auto & i : vector) {
            std::cout<<i->string<<"="<<i->count<<" ";
        }
        std::cout<<"\n";
    }
    void printTree(std::shared_ptr<Node> node, int n =0 ){
        if (node!= nullptr) {
            std::cout << ' ' << node->string<<"="<<node->count;
            if(node->right!= nullptr) {
```

```
                printTree (node->right,n+1);
            }
            else std::cout <<"\n";

            if(node->left!= nullptr) {
                for (int i=1;i<=n;i++){
                    std::cout << "  ";
                }
                printTree (node->left,n+1);
            }
        }
        else {
            std::cout<<"Tree is empty!";
        }
    }

    bool comp(std::shared_ptr<Node> &left, std::shared_ptr<Node>
&right){
        return left->count > right->count;
    }

    std::vector<std::shared_ptr<Node>> count_symbols(const
std::string& string){
        std::vector<std::shared_ptr<Node>> vector;

        bool exist;

        if (string.size()>1){
            for (int i = 1; i < string.size(); ++i) {
                exist= false;
                for (auto & j : vector) {
                    if (string[i]==j->c){
                        exist= true;
                        j->count++;
                    }
```

```
                }
                if (!exist){
                    std::shared_ptr<Node> node;
                    node = std::make_shared<Node>();
                    node->c=string[i];
                    node->count=1;
                        vector.push_back(node);
                }
            }
        }
        for (auto & i : vector) {
            i->string=i->c;
        }
        std::sort(vector.begin(), vector.end(),comp);
        //print_vector(vector);
        return vector;
    }

    std::shared_ptr<Node> consTree(const std::shared_ptr<Node>& a,
const std::shared_ptr<Node>& b){
        std::shared_ptr<Node> head = std::make_shared<Node>();

        head->left=a;
        head->right=b;
        head->count=a->count+b->count;
        head->string=a->string+b->string;

        return head;
    }

    std::shared_ptr<Node>
make_tree( std::vector<std::shared_ptr<Node>>& vector){
       std::shared_ptr<Node> head;

       while (vector.size() >= 2){
```

```
            head=consTree(vector[vector.size()-1],
vector[vector.size()-2]);
            vector.pop_back();
            vector.pop_back();
            vector.push_back(head);
            std::sort(vector.begin(), vector.end(), comp);

        }

         return head;
     }

     void make_pairs( std::shared_ptr<Node>& head, std::vector<Pair>&
pairs,std::string string){

         if(head->left == nullptr &&  head->right == nullptr){
             pairs.push_back({head->string, string});
         }
         if(head->left != nullptr) {
             make_pairs(head->left, pairs, string + "0");
         }
         if(head->right != nullptr) {
             make_pairs(head->right, pairs, string + "1");
         }
     }
     void unload_pairs(const std::vector<Pair>& pairs, const
std::string& filename= "pairs.txt"){
         std::ofstream file;
         file.open(filename);
         for(auto & pair : pairs){
             file<<pair.symbol<<" - "<< pair.code <<"\n";
         }
         file.close();
     }
```

```
    void encode(const std::string string, std::vector<Pair> pairs,
std::string filename){
        std::ofstream file;
        file.open(filename);
        for(char i : string){
            for(auto & pair : pairs){
                if(pair.symbol[0] == i)
                    file << pair.code;
            }
        }
        file<<"\n";
        file.close();
    }

    std::vector<Pair> read_pairs(const std::string& filename){
        std::vector<Pair> vector;
        std::ifstream file(filename);
        std::string string;
        std::string symbol;
        std::string code;
        while(getline(file, string, '\n')){
            code.clear();
            symbol = string[0];
            for(int i = 4; i < string.size(); ++i){
                code.push_back(string[i]);
            }
            vector.push_back({symbol, code});
        }
        return vector;
    }
    void decode(std::string encodedText, const std::string& decoded,
const std::vector<Pair>& codes){
        std::ofstream file;
        file.open(decoded);
```

```
    std::string code;
    std::string::iterator iterator = encodedText.begin();
    while(encodedText.end() != iterator){
        code += *iterator;
        for(auto & i : codes){
            if(i.code == code){
                file << i.symbol;
                code.clear();
            }
        }
        iterator++;
    }
    file << "\n";
    file.close();
}


void process(const Config& config, int argc , char** argv){

    if (config.encode !=1 && config.decode != 1){
        std::cout<<"Error: incorrect input\n";
        return;
    }
    std::string string;
    if (config.from_file==0){
        if (argc!=0) {
            string = argv[0];
        } else {
            std::cout<<"Error: incorrect input\n";
            return;
        }
    }
    if (config.from_file==1){
        string = load_from_file(config.ifile);
    }
```

```
    if (config.encode==1){
        std::vector<std::shared_ptr<Node>> vector;
        vector= count_symbols(string);
        std::shared_ptr<Node> head;
        head = make_tree(vector);
        std::vector<Pair> pairs;
        make_pairs(head, pairs,"");
        unload_pairs(pairs);
        encode(string,pairs,config.ofile);
    }
    if (config.decode==1){
        string = load_from_file(config.ifile);
        std::vector<Pair> pairs = read_pairs(config.pairs);
        decode(string,config.ofile, pairs);
    }

}


int main(int argc, char ** argv) {


    char options[] = "ed:Df:o:h";
    struct option longopts[] = {
            { "encode", no_argument, nullptr, 'e' },
            { "decode", required_argument, nullptr, 'd'},
            { "help", no_argument, nullptr, 'h' },
            {"debug",no_argument, nullptr, 'D'},
            { "file", required_argument, nullptr, 'f' },
            { "output", required_argument, nullptr, 'o' },

            { nullptr, 0, nullptr, 0 }
    };
    Config config = {
```

```
                0,              // encode
                0,           //decode
                "",
                0,           // debug
                0,              // from_file

                "",       // ifile
                "out.txt",           // ofile

        };

        int opt;
        int longidx;


        while (true)
        {
            opt = getopt_long(argc, argv, options, longopts,
&longidx);
            if (opt == -1)
                break;

            switch (opt)
            {
                case 'e':
                    config.encode=1;
    //                std::cout<<"=encode\n";
                    break;

                case 'h':
                    print_help();

                    return 0;

                case 'd':
```

```
                config.decode=1;
                config.pairs=optarg;
//                std::cout<<"=decode\n";
                break;

            case 'f':
                config.ifile = optarg;
                config.from_file=1;
//                std::cout<<"=infile\t"<<optarg<<"\n";
                break;

            case 'o':
                config.ofile = optarg;
//                std::cout<<"=outfile\t"<<optarg<<"\n";
                break;

            case 'D':
                config.debug=1;
//                std::cout<<"=DEBUG\n";
                break;
            default:
                break;
        }
    }
    argc -= optind;
    argv += optind;

    process(config,argc,argv);

    return 0;
}
```

Файл: Makefile

```
        g++ -std=c++17 Source/main.cpp -o lab5
```

```
run_tests: lab4
        ./tests_script
```

Файл: tests_scripts.sh

```
#! /bin/bash

printf "\nLaunching the tests\n\n"
for n in {1..4}
do
    printf "Test$n:\n"
    ./lab5 -e -f Tests/test$n.txt -o out$n.txt
    cat "./Tests/test$n.txt" #| tr -d '\n'
    if cmp "./out$n.txt" "./Tests/true_out$n.txt" > /dev/null;
then
            printf "*** - Passed\n"
    else
            printf "*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "./Tests/true_out$n.txt"
    printf "Actual result:\n"
    cat "./out$n.txt"
    printf "\n==============\n"

done
for n in {5..8}
do
    printf "Test$n:\n"
    ./lab5 -d Tests/pairs$n.txt -f Tests/test$n.txt -o out$n.txt
    cat "./Tests/test$n.txt" #| tr -d '\n'
    if cmp "./out$n.txt" "./Tests/true_out$n.txt" > /dev/null;
then
            printf "*** - Passed\n"
```

```
        else
                printf "*** - Failed\n"
        fi
        printf "Desired result:\n"
        cat "./Tests/true_out$n.txt"
        printf "Actual result:\n"
        cat "./out$n.txt"
        printf "\n==============\n"

    done
    for n in {9..9}
    do
        printf "Test$n:\n"
        ./lab5 -e > out$n.txt
        printf "./lab5 -e\n"
        if cmp "./out$n.txt" "./Tests/true_out$n.txt" > /dev/null;
then
                printf "*** - Passed\n"
        else
                printf "*** - Failed\n"
        fi
        printf "Desired result:\n"
        cat "./Tests/true_out$n.txt"
        printf "Actual result:\n"
        cat "./out$n.txt"
        printf "\n==============\n"

    done
    for n in {10..10}
    do
        printf "Test$n:\n"
        ./lab5 abcqwezxc > out$n.txt
        printf "./lab5 abcqwezxc\n"
        if cmp "./out$n.txt" "./Tests/true_out$n.txt" > /dev/null;
then
```

```
            printf "*** - Passed\n"
    else
            printf "*** - Failed\n"
    fi
    printf "Desired result:\n"
    cat "./Tests/true_out$n.txt"
    printf "Actual result:\n"
    cat "./out$n.txt"
    printf "\n==============\n"

done



rm ./out*.txt
#rm ./pairs*.txt
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	aaabbcccccccccc	0101010000111111111111	Кодирование
2.	There those thousand thinkers were thinking how did the other three thieves go through	1101011001101111010110 0100000011011110101100 1000001010001001101000 0001111111010110011000 0000010010110010001111 1001111011001101111010 0110011000000001000110 0000011101111101000000 1111110111101100111111 1101011010111110000100 1101011001111010110100 0110110111101011001100 1010011001010001111010 1101000111010110100110 0000010101110110	Кодирование
3.	Black back bat, black back bat, black back bat	1100011011000011101100 1100001110111011110100 0011111000110110000110 1011011000011101110110 1101000111110001101100 0001110110110000111010 11011	Кодирование
4.	How many cans can a cannibal nibble,	0111011100000000011110 1111110011110000101110 1100110000101111100010	Кодирование

	if a cannibal can nibble cans? As many cans as a cannibal can nibble if a cannibal can nibble cans	110001011111011010101 011111100100110101010 111011100110001100000 110101000010011100010 111110110101010111111 001000101111100011010 101011101110011000100 010111110011001110100 111000110000111111111 100111100001011111001 100011101100011100010 111110110101010111111 001000101111100011010 101011101110011000110 101000010011100010111 110110101010111111001 000101111100011010101 011101110011000100010 1111100110	
5.	0110001111110001011010111 0010101100111100011000111 1110001011010111001010110 0111100011000111111000101 1010111001010110011110001 1000111111000101101011100 1010110011110001100011111 1000101101011100101011001 1110001100011111100010110 1011100101011001111000110 0011111100010110101110010 1011001111000110001111110 0010110101110010101100111	billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions and billions	Декодирование

	1000110001111110001011010 1110010101100111100011000 1111110001011010111001010 1100111100011000111111000 1011010111001010110011110 0011000111111000101101011 1001010110011110001100011 1111000101101011100101011 0011110001100011111100010 1101011100101011001111000 1100011111100010110101110 0101011001111000110001111 1100010110101110010101100 1111000110001111110001011 01011		
6.	1100010001110100011110011 0011001100100011110110111 0001101101111000110101111 101110100	so, this is the sushi chef	Декодирование
7.	1011011100110111011111011 0010110010101000110100100 0101111100101011000110011 1101110011111101111101001 1000111000010101011001101 1001111111000010100010011 1111111110111110100110100 0111011110111001000000111 1010101111100000111001101 1111011001110110010101010 1101111110001000011001000 1000101001001110111000111	schwerer panzerspähwagen sieben komma fünf zentimeter sonderkraftfahrzeug zweihundertvierunddreissig vier panzerabwehrkanonenwag	Декодирование

	1001111010101011111000101 1000110101011111110110011 0111010010100010000110010 0111011100001101100100010 1000100001000001011001101 0110101100110100011010111 0000110110010101000110100 1000101111100101001011111 1111011011101010001001001 0001111110001100001111010 0110001		
8.	1100011111111000100100100 1011001010001111101100001 1000110100000100011110010 1101101011110011100110101 1111011101100011010001011 1101110011010111110110001 0100000100001011100110110 0110101111000010010011101 0111011111000010010100001 1111011000101000111110000 0101110011110001111011100 0101111101111001110011011 0100111101100000110101111 1011110100101010111100111 1010001100010010010001111 1101111001011010111111011 1001101000110111110101001 0010100011001010001111001 1110101110111001110111110 0101101111010110101111010 1101010111100100000110101	if donald trump has his way, the complications from covid-19, which are well beyond what they should be -- it's estimated that 200 million people have died -- probably by the time i finish this talk,	Декодирование

	1101000011111011100101101 0111110000001000001000001 1111000011000101010111000 0100100111101000101000100 1000010110101111001011010 0011110101110001110010100 0011110111001110111010000 0001001000111011000111010 1001101110011100110111101 1100110101111011110010000 1010111110011101111111000 1001110011010100111110111 0011100110101111011011001 010000000100010		
9.	./lab5 -e	Error: incorrect input	Запуск на декодирование без входного текста
10.	./lab5 abcqwezxc	Error: incorrect input	Запуск программы без ключей