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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Демонстрация динамического метода Хаффмана

Студент гр. 9304 _____ Ламбин А.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Ламбин А.В.

Группа 9304

Тема работы: Демонстрация динамического метода Хаффмана

Исходные данные:

Программе на вход подаётся строка для динамического кодирования или закодированное сообщение для декодирования.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание работы программы
- Примеры работы программы
- Заключение
- Список использованных источников
- Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 28 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 21.12.2020

Дата защиты реферата: 21.12.2020

Студент _____ Ламбин А.В.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В данной работе разработана программа на языке программирования C++, которая принимает на вход строку и кодирует или декодирует её. Функционал программы включает в себя реализацию динамического метода кодирования и декодирования по Хаффману и демонстрацию работы алгоритма на каждой итерации. Реализован CLI-интерфейс, который позволяет выбрать режим работы программы (кодирование или декодирование), указать файлы для чтения и записи и включить режим демонстрации.

SUMMARY

In this work, a program in the C++ programming language has developed that takes a string as input and encodes or decodes it. The functionality of the program includes the implementation of an adaptive Huffman coding method and a demonstration of the algorithm's operation at each iteration. CLI interface was implemented, which allows selecting the program operation mode (encoding or decoding), specifying files for reading and writing, and enabling the demo mode.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Динамический (или адаптивный) алгоритм Хаффмана является модификацией статического алгоритма Хаффмана сжатия информации. Он позволяет ограничиться лишь одним проходом по строке, как при кодировании, так и при декодировании. Суть динамического кодирования состоит в том, что на каждой итерации при сопоставлении символу кода изменяется ход вычислений так, что на следующих итерациях этому же символу может быть сопоставлен другой код. При декодировании происходит аналогичный процесс.

Цель работы – реализовать динамический метод Хаффмана, визуализировать работу алгоритма.

1. ОПИСАНИЕ ДИНАМИЧЕСКОГО МЕТОДА ХАФФМАНА

1.1. Алгоритм построения и упорядочивания дерева кодирования

В динамическом алгоритме Хаффмана сжатия информации используется упорядоченное бинарное дерево. Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке неубывания весов. Перечисление происходит по уровням снизу вверх и слева направо на каждом уровне. Пример упорядоченного дерева Хаффмана приведён на рисунке 1.

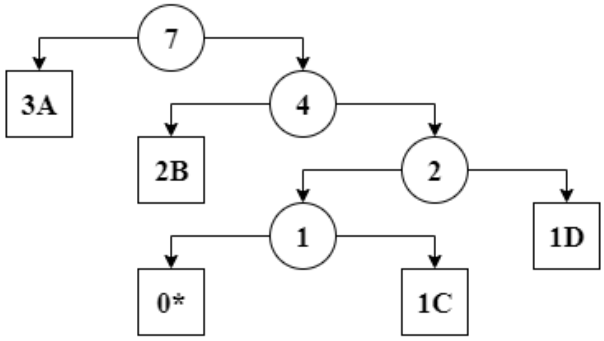

Рисунок 1 – Упорядоченное дерево Хаффмана

В начале работы алгоритма дерево кодирования состоит из единственного узла, содержащего специальный символ и имеющий вес, равный 0. Он необходим для добавления в дерево новых узлов. При этом вместо данного узла вставляется поддерево, левым сыном корня которого становится этот самый узел, а правым – узел, вставляемый в дерево.

Чтобы упорядочить дерево, необходимо поменять местами два узла: узел, нарушивший упорядоченность, и последний из следующих за ним узлов меньшего веса. После перемены мест узлов необходимо пересчитать веса всех узлов-предков.

1.2. Алгоритм динамического кодирования

На каждой итерации алгоритма из входной строки считывается обрабатываемый символ.

Если входной символ присутствует в дереве, в результат добавляется код, соответствующий расположению листа, который вычисляется следующим

образом: при переходе от корня поддерева к левому сыну добавляется 0, при переходе к правому – 1. Веса данного узла и узлов-предков увеличиваются на 1. Если дерево становится неупорядоченным, то оно упорядочивается.

Если входной символ отсутствует в дереве, в результат добавляется код, соответствующий расположению листа с нулевым весом, и 8 бит ASCII-кода нового символа. В дерево вместо листа с нулевым весом добавляется поддерево, левый сын которого становится нулевым, а правый – новым, добавленным в дерево символом. Веса узлов-предков корректируются, а дерево, при необходимости, упорядочивается.

1.3. Алгоритм динамического декодирования

На каждой итерации алгоритма элементы входного сообщения считываются побитно. Каждый раз при считывании 0 или 1 происходит перемещение от корня вниз по соответствующей ветке бинарного дерева Хаффмана, до тех пор, пока не будет достигнут какой-либо лист дерева.

Если достигнут лист, соответствующий символу, в результат добавляется данный символ. Веса данного листа и узлов-предков увеличиваются на 1. При необходимости дерево упорядочивается.

Если же достигнут нулевой символ, из входного сообщения считываются следующие 8 бит, в результат записывается символ, ASCII-код которого соответствует этим 8 бит. В дерево добавляется новый символ. Веса узлов-предков корректируются, а дерево, при необходимости, упорядочивается.

2. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

2.1. Функция main()

В функции *main()* обрабатываются введёные ключи: ключ --encode включает режим кодирования, ключ --decode – режим декодирования, ключи --file и -f отвечают за имя входного файла, -o – за имя выходного файла, --debug включает режим отладочного вывода, а ключи -?, -h и --help вызывают справку. В том случае, если был введён ключ --file или -f, производится попытка считать строку из указанного входного файла. В случае неудачи, выводится строка ошибки и программа заканчивается. Если ключ указан не был, строка считывается из стандартного потока ввода. В том случае, если был введён ключ --encode, то вызывается статический метод *encode()* класса *Tree*. Если был введён ключ --decode, то вызываются статические методы *removeSpaces()* для удаления пробельных символов из строки и *decode()* класса *Tree*. В случае, если был указан ключ -o, результат программы записывается в указанный файл. В противном случае – в стандартный поток вывода. Ключ --debug включает режим отладочного вывода, который демонстрирует работу алгоритма на каждом его шаге.

Если после ключа --encode идёт число 2, 8 или 16, то результат кодирования будет записан в двоичной, восьмеричной или шестнадцатеричной системе счисления. По умолчанию, результат записывается в двоичной системе.

2.2. Класс Node

Класс *Node* реализует узел дерева. Он содержит в себе такие поля, как указатели на левое и правое поддеревья *left* и *right*, слабый указатель на родителя parent для более удобной реализации некоторых алгоритмов, поле *data*, хранящее в себе символ, если узел является листом, и вес узла *weight*. В классе реализовано два конструктора: один для создания листа, другой – для создания узла, не являющегося листом дерева. Метод *add()* увеличивает вес данного узла и всех его родителей. Метод *recalculate()* пересчитывает вес данного узла и всех его

родителей. Метод *print()* записывает в поток информацию об узле, если она имеется, и вес узла. Метод *colorPrint()* помимо вывода дерева в стандартный поток вывода так же выделяет переданные ему узлы.

2.3. Класс Tree

Класс *Tree* реализует структуру бинарного дерева. Его полями являются указатель на корневой узел дерева и вектор всех узлов в порядке их перечисления. В конструкторе данного класса создаётся нулевой узел, который является корнем дерева на данном шаге.

Перегруженный метод *find()* принимает на вход искомый символ или строку, состоящую из нулей и единиц и вызывает рекурсивную функцию *recursionFind()*. Функции *recursionFind()* возвращают пару значений: указатель на элемент и строку или символ. Функция, принимающая символ, рекурсивно обходит дерево, ищет символ в нём и, в случае успеха, возвращает строку из нулей и единиц, соответствующую расположению символа в дереве, а в противном случае – пустую строку. Функция, принимающая на вход строку, переходит по заданному расположению в поисках листа. Если был найден ненулевой узел, то возвращается символ, соответствующий полю *data* узла. Если был найден нулевой узел, тогда из строки считывается 8 бит, являющиеся ASCII-кодом следующего символа, который возвращается функцией.

Функция вставки нового листа в дерево *insert()* ищет нулевой узел в дереве и заменяет его на поддерево, левым сыном корня которого является нулевой узел, а правым – новый лист, содержащий вставляемый символ. С помощью метода *add()* нового узла выставляется вес, равный 1, и у всех предков корректируются веса. В конце новые узлы вставляются в вектор узлов так, чтобы их расположение соответствовало порядку обхода узлов.

Перегруженный оператор вывода в поток вызывает метод *print()* у корня дерева и возвращает новый поток.

Статический метод *removeSpaces()* принимает на вход строку, проходится по ней и удаляет встречающиеся в ней пробельные символы.

Статический метод *changeRadix()* принимает на вход строку, являющуюся числом в двоичной записи, и основание степени, в которую необходимо перевести число, и переводит двоичное число в указанную степень.

Метод *encode()* принимает ссылку на строку и булевую переменную *isDebug*, равную true, если был введён ключ --debug. В методе создаётся экземпляр класса *Tree*, строка *result*, в которую будет записываться результат, и строка *table*, в которую будет записываться таблица кодов каждого символа. Для каждого символа в строке производится попытка поиска его в дереве. Если в дереве данный элемент присутствует, то к строке *result* прибавляется его код в дереве, а вес соответствующего узла увеличивается с помощью метода *add()*. Если в дереве данный элемент отсутствует, тогда к строке *result* прибавляется его код в дереве и его код ASCII, после чего данный элемент добавляется в дерево. После обработки символа, дерево упорядочивается с помощью метода *normalize()*. Если переменная *isDebug* равна true, то параллельно процессу выполнения алгоритма выводится информация о ходе алгоритма в стандартный поток вывода.

Метод *decode()* принимает ссылку на строку и булевую переменную *isDebug*. В методе создаётся экземпляр класса *Tree*, строка *result*, в которую будет записываться результат, и строка *table*, в которую будет записываться таблица кодов каждого символа. Пока длина входной строки больше нуля, производится поиск элемента в дереве по коду. Если элемент не найден, то функция возвращает пустую строку. В противном случае, к строке *result* прибавляется соответствующий символ, а в дерево либо вставляется новый узел, если данного символа нет, либо увеличивается вес соответствующего узла. После обработки дерева, оно упорядочивается с помощью метода *normalize()*. Если переменная *isDebug* равна true, то параллельно процессу выполнения алгоритма выводится информация о ходе алгоритма в стандартный поток вывода.

Метод *normalize()* упорядочивает дерево. Сперва производится обход по вектору узлов. Если он упорядочен, то метод ничего не делает, лишь возвращает

пустой вектор. В противном случае, создаётся пустой вектор и производится поиск элементов, которые необходимо поменять местами: узел, нарушивший упорядоченность, и следующий за ним последний узел меньшего веса. После этого веса узлов и их родителей пересчитываются. В вектор добавляются указатели на эти элементы, чтобы при демонстрации алгоритма выделить данные узлы. Функция вызывается ещё раз для проверки упорядоченности.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

3.1. Пример корректной работы программы

Запуск программы начинается с ввода команды “make”, приводящей к компиляции и линковки программы, после чего будет создан исполняемый файл cw. Запуск программы осуществляется с помощью ввода “./cw” и последующих ключей:

- --encode для кодирования входного сообщения;
- --decode для декодирования входного сообщения;
- -f / --file для указания входного файла;
- -o для указания выходного файла;
- --debug для включения режима отладки, демонстрирующего работу алгоритма;
- -? / -h / --help для вызова справки.

Тестирование программы осуществляется с помощью скрипта test.py, написанного на языке программирования Python3. Запуск скрипта осуществляется с помощью команды “python3 test.py”.

Примеры корректной работы программы в режиме кодирования во все допустимые программой системы счисления представлены в таблице 1.

Таблица 1 – Результаты работы программы в режиме кодирования

№ п/п	Входные данные	Выходные данные
1.	Hello, World!	0100100000110010100011011001011000110111111000010110010000010000001000101011110100000111001010110000011001000011000010000
		220312154543374131010042572034530310604 1
		48328D9637E1641022BD072B0C8C21

Продолжение таблицы 1

2.	Live not on evil	0100110000110100100011101101000110010100000100000110001101110100001101111010001110100101001011111111111111100011100011011000460644355062404061564157216451377761615419869 1DA32820C6E86F474A5FFF1C6C
3.	Beware of bugs	0100001000110010100011101111000110000100001110010001100001000000100011011110000011001100011110001100010101000111010110000011001110110001110011041062435706041621410043360314361425072603166163408CA3BC610E46102378331E3151D60CEC73

Примеры корректной работы программы в режиме декодирования представлены в таблице 2.

Таблица 2 – Результаты работы программы в режиме декодирования

№ п/п	Входные данные	Выходные данные
1.	0100100000110010100011011001011000110111111000010110010000010000001000010101111010000011100101011000001100100011000010000	Hello, World!
2.	0100110000110100100011101101000110010100000100000110001101110100001101111010001110100101001011111111111111100011100011011000	Live not on evil
3.	0100001000110010100011101111000110000100001110010001100001000000100011011110000011001100011110001100010101000111010110000011001110110001110011	Beware of bugs

Ниже на рисунках 2-6 представлены примеры работы программы в режиме отладки.

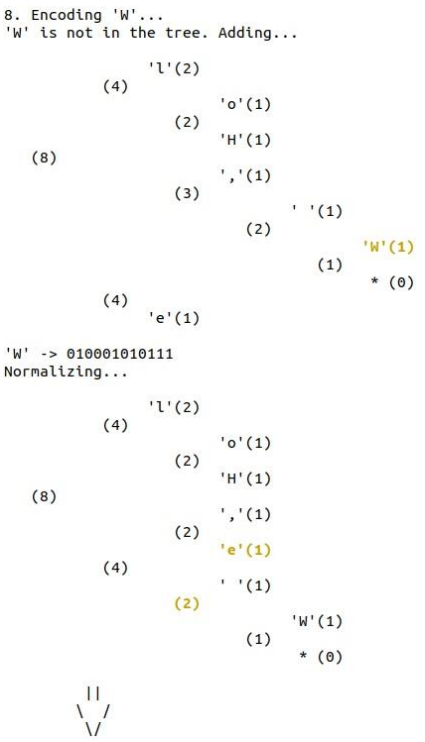

Рисунок 2 – Кодирование ‘W’

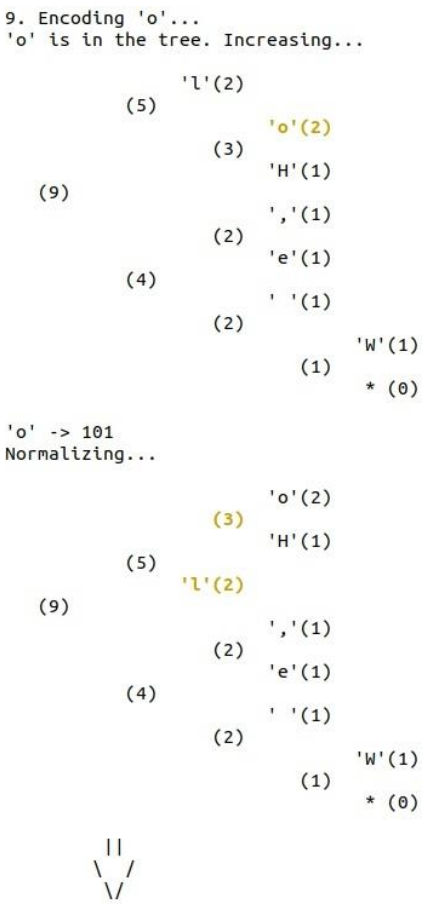

Рисунок 3 – Кодирование ‘o’

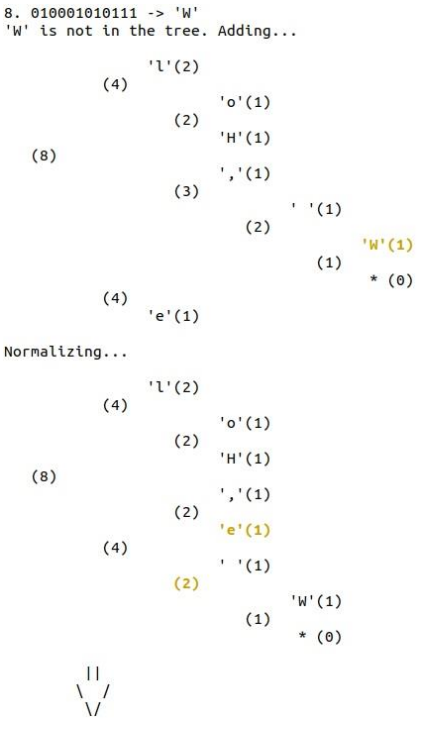

Рисунок 4 – Декодирование ‘W’

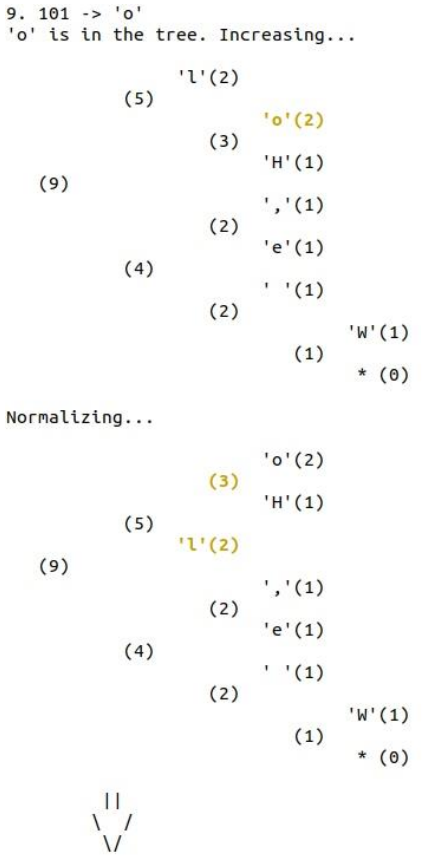

Рисунок 5 – Декодирование ‘o’

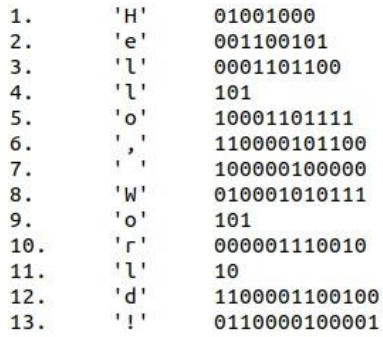

Рисунок 6 – Таблица кодов символов

3.2. Пример обработки ошибок

Примеры обработки ошибок программой представлены в таблице 3.

Таблица 3 – Результаты обработки ошибок

№ п/п	Входные данные	Выходные данные
1.	0100100000110010100011011001011000110111111000010110010000010000000100001010111010000001110010101100001100100011000010000	Error: Wrong input string
2.	02002000002200202000220220020220002202 22	Error: Wrong input string
3.	./cw --decode --file tests/testError3.txt	Error: Wrong input file

ЗАКЛЮЧЕНИЕ

В ходе работы был рассмотрен динамический метод Хаффмана сжатия информации. Была разработана программа на языке программирования C++ с консольным интерфейсом CLI, позволяющая кодировать и декодировать сообщения с помощью адаптивного алгоритма Хаффмана и демонстрирующая алгоритм на каждом шаге. Использование данного алгоритма, на мой взгляд, оправдано ввиду хорошего сжатия информации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лидовский В.В. Теория информации. М.:Наука, 2003. 112 с.
2. Адаптивный алгоритм Хаффмана сжатия информации / М.А. Кудрина, К.А. Кудрин, О.А. Дегтярева и Е.В. Сопченко // Труды Международного симпозиума «Надежность и качество». 2015, том 1.
3. en.cppreference.com/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <getopt.h>
#include "tree.h"

int main (int argc, char *argv[]) {
    bool isEncode = false, isDecode = false;
    int radix = 2;
    std::string nameInput = "", nameOutput = "";
    bool isDebug = false;
    const char *shortOptions = "f:o:h?";
    option longOptions[] = {
        {"encode", optional_argument, nullptr, 0},
        {"decode", no_argument, nullptr, 0},
        {"file", required_argument, nullptr, 'f'},
        {"debug", no_argument, nullptr, 0},
        {"help", no_argument, nullptr, 'h'},
        {0, 0, nullptr, 0}
    };
    int longIndex;
    int option = getopt_long(argc, argv, shortOptions, longOptions,
&longIndex);
    while (option != -1) {
        switch (option) {
            case 'f':
                nameInput = optarg;
                break;
            case 'o':
                nameOutput = optarg;
                break;
            case 0:
                if (longOptions[longIndex].name == "encode") {
                    isEncode = true;
                    if (optarg != nullptr)
                        try {
                            radix = std::stoi(optarg);
                            if (radix != 8 && radix != 16 && radix
!= 2)
                                throw 1;
                            break;
                        } catch (...) {}
                    else
                        break;
                }
                if (longOptions[longIndex].name == "decode") {
                    isDecode = true;
                    break;
                }
                if (longOptions[longIndex].name == "debug") {
                    isDebug = true;
                    break;
```

```
                    }
                case 'h':
                case '?':
                    std::cout << "\t--encode[=<radix>]\tEnable encoding
mode [and set radix of the code (may be 8 or 16)]\n";
                    std::cout << "\t--decode\t\tEnable decoding
mode\n";
                    std::cout << "\t-f --file <file>\tSpecify the name
of the input file\n";
                    std::cout << "\t-o <file>\t\tSpecify the name of
the output file\n";
                    std::cout << "\t--debug\t\t\tEnable debugging
mode\n";
                    std::cout << "\t-h -? --help\t\tCall help\n";
                    return 0;
            }
            option = getopt_long(argc, argv, shortOptions, longOptions,
&longIndex);
        }

        std::string str = "";

        if (nameInput.size()) {
            std::ifstream file(nameInput);
            if (!file.is_open()) {
                std::cerr << "Error: Wrong input file\n";
                return 0;
            }
            std::string temp;
            while (std::getline(file, temp))
                str += temp;
            file.close();
        } else {
            std::getline(std::cin, str);
        }

        if (isEncode) {
            str = Tree::encode(str, isDebug);
            if (radix != 2) {
                Tree::changeRadix(str, radix);
            }
        }
        if (isDecode) {
            Tree::removeSpaces(str);
            str = Tree::decode(str, isDebug);
        }

        if (nameOutput.size()) {
            std::ofstream file(nameOutput);
            file << str;
            file.close();
        } else {
            std::cout << str << '\n';
        }
        return 0;
    }
```

Файл node.h

```
#ifndef NODE_H
#define NODE_H

#include <iostream>
#include <vector>
#include <variant>
#include <memory>

class Node {
public:
    Node (std::weak_ptr<Node>, char);
    Node (std::shared_ptr<Node>, std::shared_ptr<Node>,
std::weak_ptr<Node>);
    void add ();
    void recalculate ();
    void print (std::ostream &, int);
    int colorPrint (std::vector<std::shared_ptr<Node>>, int);

private:
    std::shared_ptr<Node> left, right;
    std::weak_ptr<Node> parent;
    std::variant<char, std::monostate> data;
    unsigned int weight;
    friend class Tree;

};

#endif //NODE_H
```

Файл node.cpp

```
#include "node.h"

Node::Node (std::weak_ptr<Node> parent, char data) : left(nullptr),
right(nullptr), parent(parent) {
    this->data = data;
    this->weight = 0;
}

Node::Node (std::shared_ptr<Node> left, std::shared_ptr<Node>
right, std::weak_ptr<Node> parent) : left(left), right(right),
parent(parent) {
    this->data = std::monostate();
    this->weight = 0;
}

void Node::add () {
    this->weight++;
    if (this->parent.lock())
        this->parent.lock()->add();
}

void Node::recalculate () {
    if (std::holds_alternative<std::monostate>(this->data)) {
        int leftWeight = 0, rightWeight = 0;
```

```
            if (this->left)
                leftWeight = this->left->weight;
            if (this->right)
                rightWeight = this->right->weight;
            this->weight = leftWeight + rightWeight;
        }
        if (this->parent.lock())
            this->parent.lock()->recalculate();
    }

    void Node::print (std::ostream &out, int level) {
        if (this->right)
            this->right->print(out, level + 1);

        for (int i = 0; i < level; i++)
            out << '\t';
        if (std::holds_alternative<char>(this->data))
            if (std::get<char>(this->data) == '\0')
                out << " * (" << this->weight << ")\n";
            else
                out << '\'' << std::get<char>(this->data) << "\'(" <<
this->weight << ")\n";
        else
            out << "   (" << this->weight << ")\n";

        if (this->left)
            this->left->print(out, level + 1);
    }

    int Node::colorPrint (std::vector<std::shared_ptr<Node>> arr, int
level) {
        int strCount = 0;

        if (this->right)
            strCount += this->right->colorPrint(arr, level + 1);

        for (int i = 0; i < level; i++)
            std::cout << '\t';
        bool isColor = false;
        std::vector<std::shared_ptr<Node>>::iterator iter;
        for (iter = arr.begin(); iter != arr.end(); iter++)
            if (this == iter->get()) {
                isColor = true;
                break;
            }

        if (isColor)
            std::cout << "\033[1;33m";
        if (std::holds_alternative<char>(this->data))
            if (std::get<char>(this->data) == '\0')
                std::cout << " * (" << this->weight << ')';
            else
                std::cout << '\'' << std::get<char>(this->data) <<
"\'(" << this->weight << ')';
        else
            std::cout << "   (" << this->weight << ')';
        if (isColor)
```

```
            std::cout << "\033[0m";
        std::cout << '\n';
        strCount++;

        if (this->left)
            strCount += this->left->colorPrint(arr, level + 1);
        return strCount;
    }
```

Файл tree.h

```
    #ifndef TREE_H
    #define TREE_H

    #include <iostream>
    #include <string>
    #include <algorithm>
    #include <vector>
    #include <utility>
    #include <memory>
    #include <cmath>
    #include "node.h"

    class Tree {
    public:
        Tree ();
        std::pair<std::shared_ptr<Node>, std::string> find (char);
        std::pair<std::shared_ptr<Node>, char> find (std::string &);
        std::shared_ptr<Node> insert (char);
        friend std::ostream &operator<< (std::ostream &, const Tree &);
        static void removeSpaces (std::string &);
        static void changeRadix (std::string &, int);
        static std::string encode (std::string &, bool);
        static std::string decode (std::string &, bool);

    private:
        std::pair<std::shared_ptr<Node>,   std::string>   recursionFind
(std::shared_ptr<Node>, char, std::string);
        std::pair<std::shared_ptr<Node>,       char>       recursionFind
(std::shared_ptr<Node>, std::string &);
        std::vector<std::shared_ptr<Node>> normalize ();

        std::shared_ptr<Node> root;
        std::vector<std::shared_ptr<Node>> nodeArray;

    };

    #endif //TREE_H
```

Файл tree.cpp

```
    #include "tree.h"

    Tree::Tree () {
        std::shared_ptr<Node> null = nullptr;
        root = std::make_shared<Node>(null, '\0');
```

```
        nodeArray.push_back(root);
    }

    std::pair<std::shared_ptr<Node>, std::string> Tree::find (char
data) {
        return recursionFind(this->root, data, "");
    }

    std::pair<std::shared_ptr<Node>, char> Tree::find (std::string
&str) {
        return recursionFind(this->root, str);
    }

    std::shared_ptr<Node> Tree::insert (char data) {
        std::shared_ptr<Node> null = nullptr;
        std::shared_ptr<Node> cur = std::make_shared<Node>(null, data);
        auto searchResult = recursionFind(this->root, '\0', "");
        std::shared_ptr<Node> zero = std::get<0>(searchResult);
        if (!zero->parent.lock()) {
            root = std::make_shared<Node>(zero, cur, null);
            cur->parent = root;
            zero->parent = root;
            cur->add();
        } else {
            std::shared_ptr<Node> parent = std::make_shared<Node>(zero,
cur, zero->parent);
            cur->parent = parent;
            if (zero->parent.lock()->left == zero)
                zero->parent.lock()->left = parent;
            else
                zero->parent.lock()->right = parent;
            zero->parent = parent;
            cur->add();
        }

        std::vector<std::shared_ptr<Node>>::reverse_iterator iter;
        for (iter = nodeArray.rbegin(); iter != nodeArray.rend();
iter++)
            if (std::holds_alternative<char>((*iter)->data) && (*iter)-
>weight == 0) {
                *iter = zero->parent.lock();
                break;
            }
        nodeArray.push_back(cur);
        nodeArray.push_back(zero);
        return cur;
    }

    std::ostream &operator<< (std::ostream &out, const Tree &cur) {
        cur.root->print(out, 0);
        return out;
    }

    void Tree::removeSpaces (std::string &str) {
        for (int i = 0; i < str.length(); i++)
            if (isspace(str[i])) {
                str.erase(i, 1);
```

```
                i--;
            }
        }

        void Tree::changeRadix (std::string &str, int radix) {
            std::string res = "";
            int count = (radix == 8) ? 3 : 4;
            std::string num = "";
            do {
                try {
                    num = str.substr(str.length() - count);
                    str.erase(str.length() - count);
                } catch (...) {
                    num = str;
                    str = "";
                }
                int digit = 0;
                for (int i = 0; i < count; i++)
                    if (num[i] == '1')
                        digit += pow(2, count - i - 1);
                if (digit == 10)
                    res += 'A';
                else if (digit == 11)
                    res += 'B';
                else if (digit == 12)
                    res += 'C';
                else if (digit == 13)
                    res += 'D';
                else if (digit == 14)
                    res += 'E';
                else if (digit == 15)
                    res += 'F';
                else
                    res += std::to_string(digit);
            } while (str.length() > 0);
            std::reverse(res.begin(), res.end());
            str = res;
        }

        std::string Tree::encode (std::string &str, bool isDebug) {
            Tree tree;
            std::string result = "";
            int num = 0;
            std::string table = "";
            for (char symbol : str) {
                if (isDebug)
                    std::cout << num + 1 << ". Encoding \'" << symbol <<
"\'...\n";
                auto searchResult = tree.find(symbol);
                if (std::get<0>(searchResult)) {
                    result += std::get<1>(searchResult);
                    std::get<0>(searchResult)->add();
                    if (isDebug) {
                        std::cout << "\'" << symbol << "\' is in the tree.
Increasing...\n\n";
                        tree.root->colorPrint({std::get<0>(searchResult)},
0);
```

```
                    table += std::to_string(num + 1) + ".\t\'" + symbol
+ "\'\t" + std::get<1>(searchResult) + '\n';
                    std::cout << "\n\'" << symbol << "\' -> " <<
std::get<1>(searchResult) << '\n';
                }
            } else {
                auto code = std::get<1>(tree.find('\0'));
                int symbolCode = static_cast<int>(symbol);
                std::string binSymbolCode = "00000000";
                for (int i = 7; i >= 0; i--) {
                    if (symbolCode % 2) {
                        binSymbolCode[i] = '1';
                    }
                    symbolCode /= 2;
                }
                code += binSymbolCode;
                result += code;
                auto cur = tree.insert(symbol);
                if (isDebug) {
                    std::cout << "\'" << symbol << "\' is not in the
tree. Adding...\n\n";
                    tree.root->colorPrint({cur}, 0);
                    table += std::to_string(num + 1) + ".\t\'" + symbol
+ "\'\t" + code + '\n';
                    std::cout << "\n\'" << symbol << "\' -> " << code
<< '\n';
                }
            }
            auto normalized = tree.normalize();
            if (isDebug) {
                num++;
                std::cout << "Normalizing...\n\n";
                tree.root->colorPrint(normalized, 0);
                std::string enter;
                std::getline(std::cin, enter);
                if (num < str.size())
                    std::cout << "\t ||\n\t\\  /\n\t \\/\n\n";
            }
        }
        if (isDebug)
            std::cout << table << '\n';
        return result;
    }

    std::string Tree::decode (std::string &code, bool isDebug) {
        Tree tree;
        std::string result = "", str = code;
        if (isDebug) {
            std::cout << "Decoding...\n\n" << tree << '\n';
        }
        int num = 0;
        std::string table = "";
        while (code.length() > 0) {
            auto searchResult = tree.find(code);
            if (std::get<0>(searchResult)) {
                if (isDebug) {
```

```
                    std::cout << num + 1 << ". " << str.substr(0,
str.length() - code.length()) << " -> \'" << std::get<1>(searchResult) <<
"\'\n";
                    table += std::to_string(num + 1) + ".\t\'" +
std::get<1>(searchResult) + "\'\t" + str.substr(0, str.length() -
code.length()) + '\n';
                    str.erase(0, str.length() - code.length());
                }
                result += std::get<1>(searchResult);
                if (std::get<char>(std::get<0>(searchResult)->data) ==
'\0') {
                    auto cur = tree.insert(std::get<1>(searchResult));
                    if (isDebug) {
                        std::cout << '\'' << std::get<1>(searchResult)
<< "\' is not in the tree. Adding...\n\n";
                        tree.root->colorPrint({cur}, 0);
                        std::cout << '\n';
                    }
                } else {
                    std::get<0>(searchResult)->add();
                    if (isDebug) {
                        std::cout << '\'' << std::get<1>(searchResult)
<< "\' is in the tree. Increasing...\n\n";
                        tree.root-
>colorPrint({std::get<0>(searchResult)}, 0);
                        std::cout << '\n';
                    }
                }
                auto normalized = tree.normalize();
                if (isDebug) {
                    num++;
                    std::cout << "Normalizing...\n\n";
                    tree.root->colorPrint(normalized, 0);
                    std::string enter;
                    std::getline(std::cin, enter);
                    if (code.length())
                        std::cout << "\t ||\n\t\\  /\n\t \\/\n\n";
                }
            } else {
                return "";
            }
        }
        if (isDebug)
            std::cout << table << '\n';
        return result;
    }

    std::pair<std::shared_ptr<Node>, std::string> Tree::recursionFind
(std::shared_ptr<Node> node, char data, std::string res) {
        if (std::holds_alternative<char>(node->data)) {
            if (std::get<char>(node->data) == data)
                return std::pair<std::shared_ptr<Node>,
std::string>(node, res);
        } else {
            std::pair<std::shared_ptr<Node>, std::string> result
{nullptr, ""};
            if (node->left) {
```

```
                result = recursionFind(node->left, data, res + "0");
                if (std::get<0>(result))
                    return result;
            }
            if (node->right) {
                result = recursionFind(node->right, data, res + "1");
                if (std::get<0>(result))
                    return result;
            }
        }
        return std::pair<std::shared_ptr<Node>, std::string>(nullptr,
"");
    }

    std::pair<std::shared_ptr<Node>, char> Tree::recursionFind
(std::shared_ptr<Node> node, std::string &str) {
        if (!node) {
            std::cerr << "Error: Wrong input string\n";
            return std::pair<std::shared_ptr<Node>, char>(nullptr,
'\0');
        }
        if (std::holds_alternative<char>(node->data)) {
            if (std::get<char>(node->data) == '\0') {
                std::string asciiCode = str.substr(0, 8);
                str.erase(0, 8);
                if (asciiCode.length() != 8) {
                    std::cerr << "Error: Wrong input string\n";
                    return std::pair<std::shared_ptr<Node>,
char>(nullptr, '\0');
                }
                char symbol = 0;
                for (int i = 0; i < 8; i++)
                    if (asciiCode[i] == '1')
                        symbol += pow(2, 7 - i);
                return std::pair<std::shared_ptr<Node>, char>(node,
symbol);
            } else {
                return std::pair<std::shared_ptr<Node>, char>(node,
std::get<char>(node->data));
            }
        } else {
            if (str.length() == 0) {
                std::cerr << "Error: Wrong input string\n";
                return std::pair<std::shared_ptr<Node>, char>(nullptr,
'\0');
            }
            if (str[0] == '0') {
                str.erase(0, 1);
                return recursionFind(node->left, str);
            } else if (str[0] == '1') {
                str.erase(0, 1);
                return recursionFind(node->right, str);
            } else {
                std::cerr << "Error: Wrong input string\n";
                return std::pair<std::shared_ptr<Node>, char>(nullptr,
'\0');
            }
```

```
        }
    }

    std::vector<std::shared_ptr<Node>> Tree::normalize () {
        bool isNorm = true;
        unsigned int curWeight = 0;
        std::vector<std::shared_ptr<Node>>::reverse_iterator iter;
        for (iter = nodeArray.rbegin(); iter != nodeArray.rend();
iter++) {
            if ((*iter)->weight > curWeight)
                curWeight = (*iter)->weight;
            if ((*iter)->weight < curWeight) {
                isNorm = false;
                break;
            }
        }

        std::vector<std::shared_ptr<Node>> used;
        if (!isNorm) {
            std::shared_ptr<Node> temp;
            int pos;
            for (int i = nodeArray.size() - 1; i >= 0; i--)
                if (nodeArray[i]->weight == curWeight) {
                    temp = nodeArray[i];
                    pos = i;
                    break;
                }
            for (int i = 0; i < nodeArray.size(); i++) {
                if (nodeArray[i]->weight < curWeight) {
                    std::shared_ptr<Node> cur = nodeArray[i];
                    if (cur->parent.lock()->left == cur)
                        cur->parent.lock()->left = temp;
                    else
                        cur->parent.lock()->right = temp;
                    if (temp->parent.lock()->left == temp)
                        temp->parent.lock()->left = cur;
                    else
                        temp->parent.lock()->right = cur;
                    std::swap(temp->parent, cur->parent);
                    cur->recalculate();
                    temp->recalculate();
                    nodeArray[i] = temp;
                    nodeArray[pos] = cur;
                    used.push_back(cur);
                    used.push_back(temp);
                    break;
                }
            }
            auto elseUsed = this->normalize();
            std::vector<std::shared_ptr<Node>>::iterator it;
            for (it = elseUsed.begin(); it != elseUsed.end(); it++)
                used.push_back(*it);
        }
        return used;
    }
```