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: Алгоритм Хаффмана. Динамический

Студент гр. 9304 Ламбин А.В.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с алгоритмом динамического кодирования Хаффмана, реализовать его с помощью языка программирования C++.

Задание.

Вариант 5.

Реализовать алгоритмы динамического кодирования и декодирования Хаффмана.

Выполнение работы.

При запуске программы 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* за вывод справки. 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.

Метод *encode()* принимает ссылку на строку и булевую переменную *isDebug*, равную *true*, если был введён ключ *--debug*. В методе создаётся экземпляр класса *Tree* и строка *result*, в которую будет записываться результат. 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. Если переменная *isDebug* равна *true*, то параллельно процессу выполнения алгоритма выводится отладочная информация в стандартный поток вывода.

Метод *decode()* принимает ссылку на строку и булевую переменную *isDebug*. В методе создаётся экземпляр класса *Tree* и строка *result*, в которую будет записываться результат. 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. Если переменная *isDebug* равна *true*, то параллельно процессу выполнения алгоритма выводится отладочная информация в стандартный поток вывода.

Для реализации алгоритмов динамического кодирования и декодирования, были реализованы классы бинарного дерева *Tree* и узла *Node*.

Класс *Tree* включает в себя два приватных поля: *root* – указатель на корень дерева – и *nodeArray* – вектор всех узлов дерева, упорядоченных по убыванию их весов. Конструктор класса *Tree* создаёт корень дерева, помещая его в вектор. Для поиска узлов по данным и по коду, был реализован перегруженный метод *find()*, возвращающий результат приватного метода *recursionFind()*, который реализует КЛП-обход дерева в поисках необходимого узла и возвращает пару значений указателя на узел и соответствующий символ. Для вставки элемента в дерево реализован метод *insert()*, вставляющий элемент в возможное место в дереве и добавляющий новые узлы в вектор *nodeArray*. Метод возвращает указатель на новый элемент. Для вывода дерева перегружен оператор <<. Для упорядочивания дерева используется метод *normalize()*, проходящий по вектору *nodeArray* с конца и меняющий два элемента местами в случае нарушения упорядоченности. Метод возвращает вектор указателей на изменённые узлы.

Класс *Node* включает в себя приватные указатели на левое и правое поддеревья, а также на родителя, данные, типа *char* или *std::monostate*, и вес элемента. В перегруженных конструкторах все поля заполняются. Метод *add()* увеличивает вес узла и вызывает сам себя для всех родителей для корректировки их весов. Метод *recalculate()* пересчитывает для данного узла и всех его родителей веса. Методы *print()* и *colorPrint()* печатают деревья, однако второй выделяет жёлтым цветом указанные узлы.

Разработанный программный код см. в приложении А.

Тестирование.

Запуск программы начинается с ввода команды *“make”*, что приведёт к компиляции и линковки программы, после чего будет создан исполняемый файл lab5. Запуск программы осуществляется с помощью ввода *“./lab5”*.

Тестирование программы производится с помощью скрипта script.py, написанного на языке программирования Python3. Запуск скрипта осуществляется командой *“python3 script.py”*.

Результаты тестирования см. в приложении Б.

Выводы.

Было проведено ознакомление с алгоритмами динамического кодрования и декодирования Хаффмана, которые были реализованы с помощью языка программирования C++.

Была разработана программа, выполняющая динамическое кодирование и декодирование. Использование данного алгоритма, на мой взгляд, оправдано ввиду хорошего сжатия информации.

Был раз интерфейс командной строки для более простого взаимодействия пользователя с программой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: 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;
    std::string nameInput = "", nameOutput = "";
    bool isDebug = false;
    const char *shortOptions = "f:o:h?";
    option longOptions[] = {
        {"encode", no_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 'h':
            case '?':
                std::cout << "\t--encode\t\tEnable encoding mode\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;
            case 0:
                if (longOptions[longIndex].name == "encode") {
                    isEncode = true;
                    break;
                }
                if (longOptions[longIndex].name == "decode") {
                    isDecode = true;
                    break;
                }
                if (longOptions[longIndex].name == "debug") {
```

```
                        isDebug = true;
                        break;
                    }
            }
            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 (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;
    }
```

Название файла: tree.h

```
#ifndef TREE_H
#define TREE_H

#include <iostream>
#include <string>
#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 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
```

Название файла: 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--;
            }
    }

    std::string Tree::encode (std::string &str, bool isDebug) {
        Tree tree;
        std::string result = "";
        int num = 0;
        for (char symbol : str) {
            if (isDebug)
                std::cout << "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);
                    std::cout  <<  "\n\'"  <<  symbol  <<  "\'  ->  "  <<
std::get<1>(searchResult) << '\n';
                }
            } else {
                result += 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;
                }
                result += binSymbolCode;
                auto cur = tree.insert(symbol);
                if (isDebug) {
                    std::cout << "\'" << symbol << "\' is not in the tree.
Adding...\n\n";
                    tree.root->colorPrint({cur}, 0);
                    std::cout  <<  "\n\'"  <<  symbol  <<  "\'  ->  "  <<
std::get<1>(tree.find('\0')) + binSymbolCode << '\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";
            }
        }
        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';
        }
        while (code.length() > 0) {
            auto searchResult = tree.find(code);
            if (std::get<0>(searchResult)) {
                if (isDebug) {
                    std::cout    <<    str.substr(0,    str.length()    -
code.length()) << " -> " << std::get<1>(searchResult) << '\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) {
                    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 "";
            }
        }
        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;
    }
```

Название файла: 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
```

Название файла: tree.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;
     }
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 – Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Hello, World!	01001000001100101000110110010 11000110111111000010110010000 01000000100010101111010000011 10010101100001100100011000010 0001	
2.	Cozy sphinx waves quart jug of bad milk.	01000011001101111000111101010 00111100100000100000110001110 01110000111000001000110100000 00011010011110001101110110000 11110001100101000111011101100 01100001010000111011000100011 00101101010000000011100011110 00011101010011110100011100101 00100011101001001010000110101 01011101110001100111100101110 10100011001100011111000110001 01100001000011001001111101100 01101101110000000000110110011 0100001101011101110000101110	
3.	Deutschland! Mein Herz in Flammen, will dich lieben und verdammen.	01000100001100101000111010110 00111010000001110011110001100 01110000110100001000110110000 00011000011110001101110110000 110010010100001000011000000010 00000110001001101011001000011 010011011010100000001001000001 11010001110010110000011110101 110110111111011101100010001101 100000000111000110110101000010	

		01000010000001011001111000010 10111101001101100100100101100 01100011101001110110000111011 00011000100010001001010000011 01010010101000111011001110101 00100110011100100000100000111 10000101110	
4.	01001000001100101000110110010 11000110111111000010110010000 01000000100010101111010000011 10010101100001100100011000010 0001	Hello, World!	
5.	01000011001101111000111101010 00111100100000100000110001110 01110000111000001000110100000 00011010011110001101110110000 11110001100101000111011101100 01100001010000111011000100011 00101101010000000011100011110 00011101010011111010001110010 10 00100011101001001010000110101 01011101110001100111100101110 10100011001100011111000110001 01100001000011001001111101100 01101101110000000000110110011 01000011010111011100000101110	Cozy sphinx waves quart jug of bad milk.	
6.	01000100001100101000111010110 00111010000001110011110001100 01110000110100001000110110000 00011000011110001101110110000 11001001010000100001100000010 00000110001001101011001000011 01001101101010000001001000001	Deutschland! Mein Herz in Flammen, will dich lieben und verdammen.	

	11010001110010110000011110101 11011011111101110110001000110 10000000011100011011010100010 01000010000001011001111000010 10111101001101100100100101100 01100011101001110110000111011 00011000100010001001010000011 01010010101000111011001110101 00100110011100100000100000111 10000101110		