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: Динамическое кодирование и декодирование Хаффмана

текущий контроль

Студент гр. 9303 _____ Емельянов С.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Емельянов С.А.

Группа 9303

Тема работы Динамическое кодирование и декодирование Хаффмана текущий контроль.

Исходные данные: вариант 6, требуется реализовать текущий контроль по динамическому кодированию и декодированию Хаффмана.

Содержание пояснительной записки: аннотация, содержание, введение, основные теоретические положения, описание программы, тестирование, заключение, список используемой литературы и интернет ресурсов, приложение.

Предполагаемый объём пояснительной записки: Не менее 15 страниц.

Дата выдачи задания: 6.11.2020

Дата сдачи курсовой работы: 25.12.2020

Дата защиты курсовой работы: 25.12.2020

Студент гр. 9303 _____ Емельянов С.А.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В данной работе была разработана программа текущего контроля по динамическому кодированию и декодированию Хаффмана. Программа была написана в среде Visual Studio Code на языке C++. Данные считываются из файла, также данные прописываются пользователем, потом эти данные сравниваются и выдаётся насколько правильный ответ дал студент.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....5
1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ.....6
2. ОПИСАНИЕ ПРОГРАММЫ.....8
2.1 Описание основных функций.....8
2.2 Описание вспомогательных функций.....9
2.3 Описание библиотек.....9
3. ТЕСТИРОВАНИЕ.....10
ЗАКЛЮЧЕНИЕ.....12
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ И ИНТЕРНЕТ РЕСУРСОВ.....13
ПРИЛОЖЕНИЕ А.....14

ВВЕДЕНИЕ

Целью данной работы является реализация программы по динамическому кодированию и декодированию Хаффмана и текущего контроля студентов.

Для достижения цели требуется решить следующие задачи:

1. Изучение теоретических сведений по динамическому кодированию и декодированию Хаффмана.
2. Написание программы, которая позволяет строить бинарные деревья и перестраивать их для динамического кодирования и декодирования.
3. Внести функции в программу для сравнения результата программы и ответа студента.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Основная идея адаптивного кодирования заключается в том, что компрессор и декомпрессор начинают работать с «пустого» дерева Хаффмана, а потом модифицируют его по мере чтения и обработки символов. Соответственно, и кодер и декодер должны модифицировать дерево одинаково, чтобы все время использовать один и тот же код, который может меняться по ходу процесса. Итак, в начале кодер строит пустое дерево Хаффмана, т.е. никакому символу коды еще не присвоены. Поэтому первый символ просто записывается в выходной поток в незакодированной форме, что обычно соответствует 8 битному коду ASCII. Затем, этот символ помещается в дерево и ему присваивается код, например, 0. После этого кодируется следующий символ во входном потоке, и если этот символ встретился впервые, то он также записывается в выходной поток в виде 8 битного ASCII символа, и помещается в дерево Хаффмана, где ему присваивается определенный код в соответствии с его текущей частотой появления, равной 1. По мере того как поступают символы на вход кодера, происходит подсчет числа их появления и их количества и в соответствии с этой информацией выполняется перестройка дерева Хаффмана. Но здесь есть один нюанс: как отличить 8 битный ASCII символ от кода переменной длины в момент декодирования последовательности? Чтобы разрешить эту коллизию используют специальный esc (escape) символ, который показывает, что за ним следует незакодированный символ. Соответственно код самого esc символа должен находиться

в дереве Хаффмана и будет меняться каждый раз по мере кодирования информации (перестройки дерева).

Декодер при восстановлении исходной последовательности работает подобно кодеру, т.е. он также последовательно строит дерево Хаффмана по мере декодирования последовательности, что позволяет корректно определять исходные символы.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1 Описание основных функций

Программа была реализована в среде разработки Visual Studio Code на языке c++.

Считывание данных происходит в цикле while в main с помощью функции genline() из файла «test.txt», результат декодирования записывается в файл «Result.txt», бинарные деревья выводятся в файл «resultGR.txt», также происходит считывание данных пользователя и вариант, который указал преподаватель.

Для декодирования файла была реализована функция DecodingFile, в которой происходит инициализация дерева, списка .состоящего из узлов дерева, также в цикле происходит декодирования файла и записывание результата декодирования в файл. Функция возвращает строку, которую декодировала.

Для декодирования битов была реализована функция DecodeSymbol, в которой происходит проверка вхождения символа в дерева, если встречается ESCAPE символ, то считываются последующие 8 символов, которые переводятся в битовую последовательность, а затем преобразуются в тип char.

Для перестройки узлов дерева по весу была реализована функция restore, перестройка дерева осуществляется с помощью списка. Функции addweight и remweight отмечают за изменение веса в дереве.

Для кодирования файла была реализована функция CodingFile, в которой происходит инициализация дерева, списка, состоящего из узлов дерева, в цикле происходит кодировка и записывание результата в файл. Функция возвращает строку, которую закодировала.

Подробнее код программы см. в приложении А.

2.2 Описание вспомогательных функций

Для вывода задания для студента в консоль была написана функция students_task.

Для проверки ответа студента и данных, считанных с файла в соответствии с вариантом, была написана функция check_answer. Функция ничего не возвращает.

Подробнее код программы см. в приложении А.

2.3 Описание библиотек

Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.

Для использования библиотеки iostream в программе необходимо включить заголовочный файл #include <iostream>.

Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream,

является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

- cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
- cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
- cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (»).

Хедер fstream предоставляет функционал для считывания данных из файла и для записи в файл. В целом он очень похож на хедер iostream, который работает с консолью, поскольку консоль это тоже файл. Поэтому все основные операции такие же, за мелкими отличиями.

Класс string предназначен для работы со строками типа char*, которые представляют собой строку с завершающим нулем. Класс string был введенн как альтернативный вариант для работы со строками типа char*. Строки, которые завершаются символом ‘\0’ еще называются С-строками. Поскольку, string есть классом, то можно объявлять объекты этого класса.

В C++ контейнер list задаёт стандартные двунаправленные списки. В эти списки можно быстро вставлять, а также удалять элементы. Однако операция обращения к элементу по номеру долгая. Использование list требует подключения #include <list>.

- begin() - указатель на начало списка
- end() - указатель на конец списка
- begin() - реверсивный указатель на конец списка

3. ТЕСТИРОВАНИЕ

На рисунке 1 представлены результаты запуска программы. Пользователь указал неверный вариант, программа завершилась успешно.

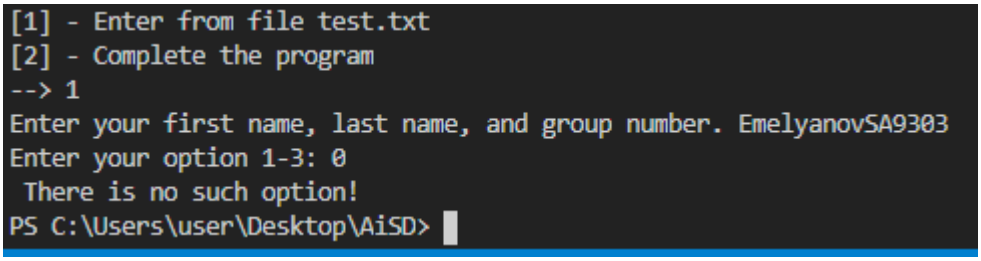

Рисунок 1 – Запуск программы с неверным вариантом

На рисунке 2 представлены результаты запуска программы. Пользователь неверно указал ответы к заданиям. Программа завершилась успешно, причём для удобной проверки, бинарное дерево по кодировке и декодировке записывается в отдельный файл.

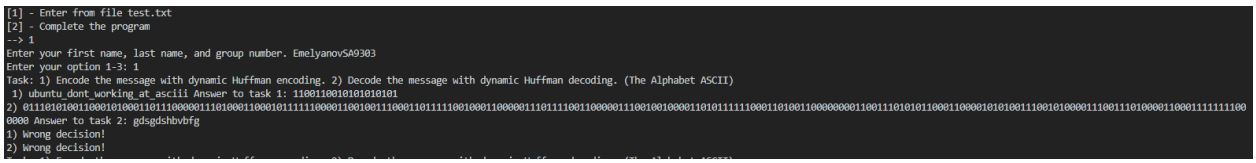

Рисунок 2 – Запуск программы с неверным ответом

ЗАКЛЮЧЕНИЕ

Был реализован динамический алгоритм декодирования и кодирования Хаффмана, для реализации алгоритма понадобились знания работы с бинарным деревом, также пришлось реализовать структуру дерева, функции для перестройки узлов дерева. Затруднение встретилось в перестроении дерева при дополнении его новыми символами с учётом веса. Были разработаны функции для вывода в консоль задания для студентов, также была реализована функция по проверки ответа студента.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ И ИНТЕРНЕТ РЕСУРСОВ

1. https://ru.wikipedia.org/wiki/Адаптивный_алгоритм_Хаффмана
2. https://habr.com/ru/post/144200/
3. https://ru.qaz.wiki/wiki/Adaptive_Huffman_coding

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#define ESCAPE 257
using namespace std;

class Tree {
public:
    Tree* left;
    Tree* right;
    Tree* parent;
    int code;
    bool symbol_code;
    int weight;
    Tree(Tree* left, Tree* right, Tree* parent, bool symbol_
code) {
        this->left = left;
        this->right = right;
        this->parent = parent;
        this->symbol_code = symbol_code;
        this->weight = 0;
        this->code = 255;
    }
};




void PrintTree(Tree* tree, int r, ofstream& fout){
    r++;
```

```
        if(tree->right!=nullptr)
            PrintTree(tree->right,r+1, fout);
        for (int i=0;i<(4*r);i++)
            fout << " ";
        if(tree->symbol_code){
            if(tree->code == ESCAPE){
                fout<<tree->weight<<"("<<"0*"<<")"<<endl;
            }
            else{
                fout<<tree->weight<<"("<<(char)tree-
>code<<")"<<endl;
            }
        }
        else{
            fout<<tree->weight<<endl;
        }
        if(tree->left!=nullptr)
        PrintTree(tree->left,++r, fout);
        --r;
    }
```

```
    ////////////////////////////////////////// Декодирование
строки
    int InputBits(string& str, int& n,  ofstream& fout) {
        int result = 0;
        char symbol;
        int bit;
```

```
            fout << str[n];
            for (int i = n; i < n + 8; i++) {
                fout << str[i];
                symbol = str[i];
                bit = atoi(&symbol);
                result = (result << 1) + bit;
            }
            n += 8;
            return result;
        }

        void addweight(Tree* tree) {
            tree->weight++;
            if (tree->parent) addweight(tree->parent);
        }

        void remweight(Tree* tree) {
            tree->weight--;
            if (tree->parent) remweight(tree->parent);
        }

        void restore(list<Tree*>& sort) {
            Tree* prev = *(--sort.end());
            Tree* current = nullptr;
            for (std::list<Tree*>::iterator i = --sort.end(); true;
i--) {
                if (prev->weight > (*i)->weight) current = prev;
                if (current && (current->weight <= (*i)->weight)) {
                    Tree test = *current;
                    current->left = prev->left;
                    if (current->left) current->left->parent = curre
nt;
                    current->right = prev->right;
```

```
                if (current->right) current->right->parent = cur
rent;
                current->code = prev->code;
                current->symbol_code = prev->symbol_code;
                prev->left = test.left;
                if (prev->left) prev->left->parent = prev;
                prev->right = test.right;
                if (prev->right) prev->right->parent = prev;
                prev->code = test.code;
                prev->symbol_code = test.symbol_code;
                int diff = current->weight - prev->weight;
                for (int i = 0; i < diff; i++) {
                    remweight(current);
                    addweight(prev);
                }
                restore(sort);
                break;
            }
            prev = *i;
            if (i == sort.begin()){
                break;
            }
        }
    }

    int DecodeSymbol(Tree* tree, string& str, int& n, int& flag_
parent, list<Tree*>& sort, ofstream& fout) {

        if (flag_parent) {

            if (n + 8 > str.length()) return 258;
            fout << "symbol = ";
            tree->right = new Tree(nullptr, nullptr, tree, true)
;
```

```
            tree->right->code = InputBits(str, n, fout);
            addweight(tree->right);
            tree->left = new Tree(nullptr, nullptr, tree, true);
            tree->left->code = ESCAPE;
            flag_parent = 0;
            sort.push_back(tree->right);
            sort.push_back(tree->left);
            fout<<" ("<<char(tree->right->code) << ")"<< endl;
            return tree->right->code;
        }
        Tree* tree_test = tree;
        if (n >= str.length() )
            return 258;
        fout << "symbol = ";
        while (1) {
            if (n >= str.length()) return 258;
            if (str[n] == '1') {
                fout << "1";
                tree_test = tree_test->right;
                if (!tree_test) return 256;
                if (tree_test->symbol_code) {
                    n += 1;
                    addweight(tree_test);
                    //fout << endl;
                    fout<<" ("<<char(tree_test->code) << ")"<< e
ndl;
                    return tree_test->code;
                }
            }
            else if (str[n] == '0') {
                fout << "0";
                tree_test = tree_test->left;
                if (!tree_test) return 256;
                if (tree_test->symbol_code) {
```

```
                    if (tree_test->code == ESCAPE) {
                        n += 1;
                        tree_test->right = new Tree(nullptr, nul
lptr, tree_test, true);
                        tree_test->right->code = InputBits(str,
n, fout);
                        addweight(tree_test->right);
                        tree_test->left = new Tree(nullptr, null
ptr, tree_test, true);
                        tree_test->left->code = ESCAPE;
                        tree_test->symbol_code = false;
                        sort.push_back(tree_test->right);
                        sort.push_back(tree_test->left);
                        fout<<" ("<<char(tree_test->right->code)
<< ")"<< endl;
                        return tree_test->right->code;
                    }
                    else {
                        n += 1;
                        addweight(tree_test);
                        fout<<" ("<<char(tree_test->code) << ")"
<< endl;
                        return tree_test->code;
                    }
                }
            }
            n += 1;
        }
    }

    string DecodingFile(string& str, ofstream& fout) {
        Tree tree(nullptr, nullptr, nullptr, false);
        list<Tree*> sort;
        ofstream file_graph;
```

```
        sort.push_back(&tree);
        string result = "";
        int symbol;
        int n = 0;
        int f = 1;
        file_graph.open("resultGR.txt", std::fstream::in | std::
fstream::out | std::fstream::app);
        while ((symbol = DecodeSymbol(&tree, str, n, f, sort, fo
ut)) != 256 && symbol != 258) {
            result += (char)symbol;
            restore(sort);
            PrintTree(&tree, 0, file_graph);
            file_graph<<"\
n-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------\n";
        }
        fout << "Result: " << result;
        file_graph.close();
        return result;
    }

    ////////////////////////////////////////////////////////////
////////////// Кодирвоание строки
    string BinSymbol( int dec){
        int mod;
        std::string str = "";
        int k = 128;
        char sym = (char) dec;
        for (int i = 7; i > -1; i--) {

            str += ((char)((sym & k) >> i) + 48);
            k = k >> 1;
        }
```

```
        return str;
    }


    string CodeSymbol(Tree* tree, string& str, int& n, int& flag
_parent, list<Tree*>& sort, ofstream& fout, bool flag_str){
        if (flag_parent) {
            if (n + 1 > str.length()) flag_str = false;
            tree->right = new Tree(nullptr, nullptr, tree, true)
;
            tree->right->code = (int)str[n];
            n+=1;
            addweight(tree->right);
            tree->left = new Tree(nullptr, nullptr, tree, true);
            tree->left->code = ESCAPE;
            flag_parent = 0;
            sort.push_back(tree->right);
            sort.push_back(tree->left);
            fout<< BinSymbol (tree->right->code);
            fout << " =";
            fout<<" ("<<char(tree->right->code) << ")"<< endl;
            return BinSymbol (tree->right->code);
        }
        std::string code = "";
        Tree* end = nullptr;
        for (std::list<Tree*>::iterator i = sort.begin(); i != s
ort.end(); i++) {
            if ((char)(*i)->code == str[n]) {
                end = (*i);
                break;
            }
        }
        //// здесь проверка в дереве
        if (end) {
```

```
                //if (first) end = end->parent;
                int result = end->code;
                addweight(end);
                Tree* current = end;
                end = end->parent;
                for (end; end; current = end, end = end->parent)
{
                    if (end->right == current ){
                        code += '1';
                    }
                    if (end->left == current ){
                        code += '0';
                    }
                }
                for (int i = 0; i < code.length() / 2; i++) {
                    char reserve = code[i];
                    code[i] = code[code.length() - i - 1];
                    code[code.length() - i - 1] = reserve;
                }
                n+=1;
                fout <<  code << ' ';
                fout << " =";
                fout<<" ("<<(char)result << ")"<< endl;
                return code;
        }
        /// добавление элемента в дерево
        else{
                Tree* esc = *(--sort.end());
                esc->right = new Tree(nullptr, nullptr, esc, tru
e);
                esc->right->code =(int) str[n];
                addweight(esc->right);
                int result = esc->right->code;
```

```
                esc->left = new Tree(nullptr, nullptr, esc, true
);
                esc->left->code = ESCAPE;
                esc->code = 258;
                esc->symbol_code = false;
                sort.push_back(esc->right);
                sort.push_back(esc->left);
                Tree* current;
                esc = esc->parent;
                for (esc; esc; current = esc, esc = esc->parent)
{
                    if (esc->right == current ){
                        code += '1';
                    }
                    else {
                        code += '0';
                    }
                }
                for (int i = 0; i < code.length() / 2; i++) {

                    char reserve = code[i];
                    code[i] = code[code.length() - i - 1];
                    code[code.length() - i - 1] = reserve;
                }
                code+= BinSymbol(result);
                n+=1;
                fout << code << ' ';
                fout << " =";
                fout<<" ("<<char(result) << ")"<< endl;
                //restore (sort);
                return code;
            }

        }
```

```
    string CodingFile(string& str, ofstream& fout){
        Tree tree(nullptr, nullptr, nullptr, false);
        list<Tree*> sort;
        sort.push_back(&tree);
        ofstream file_graph;
        string result = "";
        string symbol = "";
        int n = 0;
        int f = 1;
        bool flag = true;
        int count = 0;
        file_graph.open("resultGR.txt", std::fstream::in | std::
fstream::out | std::fstream::app);
        while (flag){
            result += symbol;
            if (n >= str.length()) break;
            symbol = CodeSymbol(&tree, str, n, f, sort, fout,fla
g);
            restore(sort);
            PrintTree(&tree, 0, file_graph);
            file_graph<<"\
n------------------------------------------------------------------
------------------------------------------------------------------
-----------------\n";
        }
        fout << "Result: " << result;
        file_graph.close();
        return result;
    }

    void check_answer(int n, string& str1, string& str2){
        cout<<n<<") ";
        if (str1 == str2){
```

```
            cout<<"Solved correctly!\n";
        }
        else{
            cout<<"Wrong decision!\n";
        }
    }

    void students_task(){
        cout<<"Task: 1) Encode the message with dynamic Huffman
encoding. 2) Decode the message with dynamic Huffman decoding. (
The Alphabet ASCII)\n ";
    }

    int main() {
        cout << "[1] - Enter from file test.txt \n[2] - Complete
the program\n";
        char flag;
        cout << "--> ";
        cin >> flag;
        ofstream fout;
        ifstream fin;
        string str1;
        string str2;
        string result1;
        string result2;
        string student_answer;
        string student_name;
        int variant;
        switch (flag) {
        case '1':
            fin.open("test.txt");
            fout.open("Result.txt", std::fstream::in | std::fstr
eam::out | std::fstream::app);
```

```
            cout<<"Enter your first name, last name, and group n
umber. ";
            cin >> student_name;
            cout<<"Enter your option 1-3: ";
            cin>>variant;
            if (variant-1<0 || variant-1>2){
                cout<<" There is no such option!";
                return 0;
            }
            while (!fin.eof()) {
                for(int i = 0; i < variant;i++ ){
                    getline(fin, str1);
                    getline(fin, str2);
                }
                students_task();
                cout<<"1) "<< str1 <<" Answer to task 1: "; cin
>> student_answer;
                fout<<student_name<<"\nTask 1. "<< "Original dat
a: " << str1 << endl;
                result1 = CodingFile(str1, fout);
                fout<<"Student's response: "<< student_answer<<"
\
n----------------------------------------------------------------
-----------------------------------------\n";
                cout<<"2) "<< str2 <<" Answer to task 2: "; cin
>> student_answer;
                fout<<student_name<<"\nTask 2. "<< "Original dat
a: " << str2 << endl;
                result2 = DecodingFile(str2, fout);
                fout<<"Student's response: "<< student_answer<<"
\
n----------------------------------------------------------------
-----------------------------------------\n";
                check_answer(1,result1, str1);
```

```
            check_answer(2,result2, str2);
            break;
        }
        fin.close();
        fout.close();
        break;
    case '2':
        cout << "Thanks for your attention!";
        break;
    default:
        cout << "\nInvalid data entered!";
        break;
    }
    return 0;
}
```