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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «АиСД»

Тема: Алгоритмы кодирования и декодирования

Студент гр. 9303 _____ Емельянов С.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Реализация и экспериментальное машинное исследование алгоритмов кодирования (Фано-Шеннона, Хаффмана), быстрого поиска на основе БДП или хеш-таблиц, сортировок.

Задание.

Вариант 6.

Реализовать алгоритм динамического декодирования Хафмена.

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.

Выполнение работы.

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».

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 и вывод в консоль.

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.

Подробнее код программы см. в приложении А.

Тестирование.

Результаты тестирования программы см. таблицу 1.

Таблица 1 – Результаты тестирования программы

Номер теста	Входные данные	Выходные данные	Коментарий
1	01110011001101000 01000110111100111 10001110111100111 11100011001010001 01000110110011001	Original data: 01110011001101000 01000110111100111 10001110111100111 11100011001010001 01000110110011001 symbol = 01110011 (s) symbol = 001101000 (h) symbol = 01 (h) symbol = 0001101111 (o) symbol = 001 (o) symbol = 11 (o) symbol = 10001110111 (w) symbol = 1001 (w)	Программа работает исправно!

		symbol = 111 (w) symbol = 110001100101 (e) symbol = 0001 (e) symbol = 010001101100 (l) symbol = 11001 (l) Result: shhooowwweell	
2	0011001111000110 0100100001100011 00100111	Original data: 0011001111000110 0100100001100011 00100111 symbol = 00110011 (3) symbol = 1 (3) symbol = 1 (3) symbol = 000110010 (2) symbol = 01 (2) symbol = 0000110001 (1) symbol = 1 (3) symbol = 001 (1) symbol = 001 (1) symbol = 11 (1) Result: 3332213111	Программа работает исправно!
3	0110100000110010	Original data:	Программа

	1000110110010111 0011011111101	0110100000110010 1000110110010111 0011011111101 symbol = 01101000 (h) symbol = 001100101 (e) symbol = 0001101100 (l) symbol = 101 (l) symbol = 11001101111 (o) symbol = 1101 (o) Result: helloo	работает исправно!

Выводы

Был реализован динамический алгоритм декодирования Хафмена, 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.

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;
    }
};

int InputBits(string& str, int& n,  ofstream& fout) {
```

```
    int result = 0;
    char symbol;
    //cout << n<< " ";
    for (int i = n; i < n + 8; i++) {
        fout << str[i];
        symbol = str[i];
        result = (result << 1) + atoi(&symbol);
    }
    n += 8;
    //cout << n << endl;
    //cout << char(result)<<" "<<result<<endl;
    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 = current;
            current->right = prev->right;
            if (current->right) current->right->parent = current;
            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<Tre
e*>& 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) << ")"<< endl;
        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, nullptr, tree_test, true);
            tree_test->right->code = InputBits(str, n, fout);
            addweight(tree_test->right);
            tree_test->left = new Tree(nullptr, nullptr, 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;
  }
}

void DecodingFile(string& str, ofstream& fout) {
  Tree tree(nullptr, nullptr, nullptr, false);
  list<Tree*> sort;
  sort.push_back(&tree);
  string result = "";
  int symbol;
  int n = 0;
  int f = 1;
  while ((symbol = DecodeSymbol(&tree, str, n, f, sort, fout)) != 256 && sy
mbol != 258) {
    // cout << "res: " << symbol << "\n";
    result += (char)symbol;
    //cout << result << endl;
    restore(sort);
  }
  fout << "Result: " << result;
}
```

```
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 str;
    string result;
    switch (flag) {
    case '1':
        fin.open("test.txt");
        fout.open("result.txt");
        while (!fin.eof()) {
            getline(fin, str);
            fout << "\nOriginal data: " << str << endl;
            DecodingFile(str, fout);
        }
        fin.close();
        fout.close();
        break;
    case '2':
        cout << "Thanks for your attention!";
        break;
    default:
        cout << "\nInvalid data entered!";
        break;
    }
```

```
    return 0;
}
```