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: Алгоритмы кодирования и декодирования

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

Цель работы

Цель работы: познакомиться с алгоритмами кодирвоания и декодирования, создать программу для декодирования файла на языке программирования C++.

Формулировка задачи: Вариант 4. Написать программу для декодирования файла с помошью статического алгоритма Хаффмана.

Реализация задачи

Для реализации декодирования было принято создать класс HuffmanTree, который содержит дерево с алфавитом.

```
template <class T>
class HuffmanTree{
private:
   HuffmanTree* left;   // 0
   HuffmanTree* right;  // 1
   bool flag = false; //true if leaf
   T value;
public:
   HuffmanTree(ifstream& s);
   bool isLeaf() {return flag;}
   HuffmanTree *get_left() {return left;}
   HuffmanTree *get_right() {return right;}
   T val() {return value;}
   ~HuffmanTree();
};
```

Конструктор класса инициализирует дерево строкой, содержащейся в начале декодируемого файла.

Метод `bool isLeaf()` возвращает true, если элемент является листом дерева. Методы `HuffmanTree *get_left()` и `HuffmanTree *get_right()` возвращают указатель на левый и правый элемент. В код символа добавляется 0 при спуске по левой ветви и 1 при спуске по правой. Метод `T val()` возвращает кодируемый элемент.

Исходный код программы представлен в приложении Б.

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе Ubuntu 18.04.1 LTS bionic компилятором g++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0. В других ОС и компиляторах тестирование не проводилось.

2. Результаты тестирования

В результате тестирования было обнаружено падение программы при отсутствии в исходном файле строки, содержащей дерево для декодирования. Также были обнаружены лишние символы - программа декодировала маркеры конца файла. Проблемы былы решены добавлением соответствующих проверок. Тестовые случаи представлены в приложении А.

Вывод

В ходе выполнения данной работы были изучены алгоритмы кодирования и декодирования. Была написана программа, декодирующая файл с помощью алгоритма Хаффмана.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Содержимое файла	Декодированный текст
((a)(((b)(c))(d)))Xqfbdvsabgfdhgmfjhg	acbadbaadaadadaaacb
((a)((b)(c)))X	abcaa
Xqfbdvsabgfdhgmfjhg	file.txt is broken.
(a)Xqfbdvsabgfdhgmfjhg	file.txt is broken.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include <string>

using namespace std;

template <class T>
class HuffmanTree{
private:
    HuffmanTree* left;   // 0
    HuffmanTree* right;  // 1
    bool flag = false; //true if leaf
    T value;

public:
    HuffmanTree(ifstream& s);
    bool isLeaf() {return flag;}
    HuffmanTree *get_left() {return left;}
    HuffmanTree *get_right() {return right;}
    T val() {return value;}
    ~HuffmanTree();
};

template <class T>
HuffmanTree<T>::HuffmanTree(ifstream& s)
{
    flag = false;
    left = NULL;
    right = NULL;
    char ch;
    if (s.peek() == '(')
        s >> ch; // remove '('
    else
        return;
    if (s.peek() == '(')
        {
            left = new HuffmanTree(s);
            if (s.peek() == ')')
                s >> ch; // remove ')'
        }
    else
```

```
        {
            flag = true;
            s >> value;
            return;
        }
    if (s.peek() == '(')
    {
        right = new HuffmanTree(s);
        s >> ch; // remove ')'
    }
    if (s.peek() == ')')
        s >> ch; // remove ')'
}

template <class T>
HuffmanTree<T>::~HuffmanTree()
{
    if (left)
        delete left;
    if (right)
        delete right;
}

int main()
{
    int n, c;
    int *el;
    string str_i, str_o;
    while(true)
    {
        cout << "Press 1 to decode a file\n" <<
                "Press 2 to exit." << endl;
        cin >> str_i;
        if (!isdigit(str_i[0]))
            continue;
        c = stoi(str_i);
        str_i.clear();
        switch (c)
        {
            case 1:
                break;
            case 2:
                return 0;
            default:
                cout << "Something went wrong. Try again!" << endl;
```

```
        continue;
    }
    cout << "Enter input file name: ";
    cin >> str_i;
    cout << "Enter output file name: ";
    cin >> str_o;
    if (str_i == str_o)
    {
        cout << "Input and output files can't be the same." << endl;
        continue;
    }
    ifstream f;
    ofstream o;
    char b;
    f.open(str_i);
    o.open(str_o);
    if (!f)
    {
        cout << "Unable to open the input file!" << endl;
        continue;
    }
    if (!o)
    {
        cout << "Unable to open the output file!" << endl;
        continue;
    }
    HuffmanTree<char> *Dictionary = new HuffmanTree<char>(f);
    HuffmanTree<char> *tmp = Dictionary;
    while (!f.eof())
    {
        b = f.get();
        if (b == '\n')
            if (f.peek() == EOF)
                break;
        for (char i = 7; i > 0; i--)
        {
            if (tmp)
                tmp = ((b & (1 << i)) == 0 ? tmp->get_left() : tmp->get_right());
            if (tmp && tmp->isLeaf())
            {
                o << tmp->val();
                tmp = Dictionary;
            }

        }
```

```
        }
        f.close();
        o.close();
        if (tmp)
            cout << str_i << " was decoded to " << str_o << " succesfully." << endl;
        else
            cout << str_i <<" is broken." << endl;
        delete Dictionary;
    }
}
```