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

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы.

Изучить алгоритм Хаффмана статического кодирования текста. Реализовать данный алгоритм на языке C++ с помощью бинарного дерева поиска.

Постановка задачи.

Закодировать строку статическим методом Хаффмана.

Уточнение задачи.

На вход программе подается строка (с файла и консоли, на выбор пользователя). Итогом завершения программы должна быть строка из нулей и единиц, которая является кодом Хаффмана для заданной строки.

Описание алгоритма.

На вход программе с файла или консоли (на выбор пользователя) подается строка. Программа проходится по строке и считает количество вхождений каждого символа в строку и выводит эту информацию на консоль. Количество вхождений каждого символа хранятся в ассоциативном массиве `map<char, int> count_of_entry`. Далее создается список указателей на узлы дерева. Проходом по ассоциативному массиву, заполняем список указателями на узлы дерева (в узле хранится символ и количество его вхождений). Затем, в цикле, условием выхода из которого является хранение 1 элемента в списке, происходит на каждой итерации: сортировка списка по количеству вхождений (от малого к большему), удаление двух элементов с наименьшим показателем вхождения, и запись в список указателя на родителя этих двух элементов. После этого, из списка (в котором после цикла остался 1 элемент — корень дерева Хаффмана) извлекается корень дерева Хаффмана. Корень передается в функцию `void print(Node *root, unsigned k = 0)` для вывода дерева Хаффмана на экран. Так же корень передается в функцию `void BuildTable (Node *root)`, которая обходом дерева записывает в ассоциативный массив `map<char, vector<bool> > table` код Хаффмана для каждого символа. И, в завершении, исходное сообщение кодируется и код Хаффмана выводится на экран.

Описание функций

Описание функций, использованных в программе представлено в табл. 1.

Таблица 1 — Описание функций

Название	Тип функции	Входные данные	Описание
print	void	**Node* root** — указатель на корень дерева **unsigned k** — счетчик глубины рекурсии	Функция, выводящая дерево на экран
BuildTable	void	**Node* root** — указатель на корень дерева	Функция нахождения детей по индексу имени.

Описание класса Node

В данном дереве реализовано бинарное дерево. Объекты класса описаны в табл. 2. Методы класса описаны в табл. 3.

Таблица 2 — Описание объектов класса `Node`

Объект	Тип	Описание
int	public	Хранит количество вхождений.
char	public	Хранит символ.
Node	public	Указатель на левое/правое поддерево

Таблица 3 — Методы класса `Queue`

Метод	Тип	Выходные данные	Входные данные	Описание
Node()	public	-	-	Конструктор дерева. Строит родителя левого и правого сына.

Тестирование

Было создано несколько тестов для проверки работы программы. Тестовые случаи представлены в приложении А.

Исходный код программы представлен в приложении Б.

Вывод

В процессе выполнения лабораторной работы было реализовано бинарное дерево. Так же был реализован статический алгоритм Хаффмана для кодирования входной строки (с построением дерева Хаффмана).

ПРИЛОЖЕНИЕ А

Тестирование

Входные данные	Код Хаффмана	Верно?
Happy New Year!!!	10000010110111101000100101011000001010010001101111111111111	Да
Happy Happy	10100111101100101001111101	Да
Happy happy	01110111110001010010111110000	Да
12 yt 11 y	1101010000111011111000	Да
[]as]]][	10011011100010	Да

ПРИЛОЖЕНИЕ Б

Код программы

Файл main.cpp

```
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <string>
#include <fstream>
#include <locale>

using namespace std;

class Node
{
public:
    int entries;
    char symbol;
    Node *left, *right;

    Node () //перегрузка конструктора
    {left=right=NULL;}

    Node(Node *L, Node *R)  //конструктор, принимает 2 ссылки (на сыновей)
                              //создает отца
    {
        left = L;
        right = R;
        entries = L->entries + R->entries;
    }
};

struct Compare
{
    bool operator() (const Node* l, const Node* r) const
    { return l->entries < r->entries; }
};

void print(Node *root, unsigned k = 0)
{
    if (root != NULL)
    {
        print(root->left, k+3);

        for (unsigned i = 0; i<k; i++)
            cout << "   ";

        /*if (root->c)
            cout << root->a << " (" << root->c << ")" << endl;*/
        if (root->left == NULL && root->right == NULL)
            cout << root->entries << " (" << root->symbol << ")" << endl;
        else
            cout << root->entries << endl;
        print(root->right, k+3);
    }
}

vector<bool> code;
```

```
map<char, vector<bool> > table;  //ассоциация символа с кодом

void BuildTable (Node *root)
{
    if (root->left!=NULL)
    {
        code.push_back(0);   //если слева не 0, ставим в вектор 0
        BuildTable(root->left);
    }

    if (root->right!=NULL)   //если справа не 0, ставим в вектор 1
    {
        code.push_back(1);
        BuildTable(root->right);
    }

    if (root->left == NULL && root->right == NULL) //если нашли букву
        table[root->symbol] = code;  //ассоциируем эту букву с кодом

    code.pop_back();  //сокращаем код на 1 цифру
}

int main()
{
    unsigned r;
    cout << "Выберите способ ввода строки:" << endl <<
            "1. Из файла" << endl << "2. С консоли"<<endl;
    cin >> r;
    string s;
    cin.ignore();

    map<char, int> count_of_entry; //для подсчета количества вхождений символов

    if (r==1)
    {
        ifstream f("input.txt");
        if(!f.is_open())
        {
            cerr << "Ошибка. Файл не найден.";
            exit(1);
        }
        else
            getline(f, s);
        f.close();
    }
    if (r==2)
    {
        cout << endl << "Введите строку: ";
        getline(cin, s);
    }

    for (int i = 0; i < s.length(); i++) //считаем вхождения
    {
        char c = s[i];
        count_of_entry[c]++;     //m от символа увеличиваем на 1
    }

    cout << "Количество вхождений символов в строку:" << endl;
```

```
    map<char, int>::iterator iter;
    for (iter=count_of_entry.begin(); iter!=count_of_entry.end(); iter++)
        cout << iter->first << ": " << iter->second << endl;

    //"засеиваем" список этими вершинами с кол-вом вхождений

    list<Node*> list;   //список указателей на узлы

    for (iter=count_of_entry.begin(); iter!=count_of_entry.end(); ++iter)
    {
        Node *p = new Node;  //создаем новый узел и инициализируем его
        p->symbol = iter->first;
        p->entries = iter->second;
        list.push_back(p);       //указатель на узел скидываем в лист ( в списке
хранятся указатели)
    }

    while (list.size()!=1) //на каждом шаге убирается 2 элемента и добавляется 1
    {                   //останется 1 переменная - вершина дерева
        list.sort(Compare());  //сортировка списка по кол-ву вхождений
        //SonL, SonR - левый, правый сын, вспомогательные переменные
        Node *SonL = list.front(); //присваиваем переменную
        list.pop_front();          //удаляем её из списка
        Node *SonR = list.front();
        list.pop_front();

        Node *parent = new Node(SonL, SonR); //создается отец двух сыновей
        list.push_back(parent);                 //кладется в список
    }

    Node *root = list.front(); //достали корень дерева

    cout << "Вид дерева Хаффмана: " << endl;
    print(root);

    BuildTable(root);//создадим таблицу значений (символ-его код)

    cout << "Код Хаффмана: " << endl;

    for (int i=0; i<s.length(); i++)
    {
        char c=s[i];  //вспомогательная переменная с
        vector<bool> x =  table[c]; //всп. перем. x, содержит код символа с

        for (int n=0; n < x.size(); n++)  //выводим код символа
            cout << x[n];
    }

    cout << endl;
    return 0;
}
```