

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: Кодирование и декодирование

Студент гр. 7382

\_\_\_\_\_

Глазунов С.А.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2018

## 1. Задание 3

Кодирование: статическое Хаффмана

## 2. Пояснение задания

Алгоритм Хаффмана — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

## 3. Описание алгоритма

Считывается строка из входного потока. После подсчитывается количество повторения каждого символа в строке. После строится бинарное дерево, так, что символ который повторялся как можно чаще оказался выше(ближе) к корню. В конце мы получим дерево Хаффмана. После мы заново проходим по строке уже сопоставляя символу его код и выводим код.

## 4. Описание функций и структур данных

```
class Node
```

Это класс, который является элементом списка. Он содержит указатели на правый и левый элемент, а также содержит поля, описывающий элемент.

```
create_code_of_each_symbol(Node*root)
```

Функция создает выделяет код для каждого символа из бинарного дерева Хаффмана.

```
void Code_Haffmana::create_tree_of_haffmana()
```

Функция создает дерево Хаффмана, используя список, который содержит сам символ и количество сколько он встречается в строке.

```
void Code_Haffmana::create_list()
```

Функция создает список, который заполняется на основе ассоциативного массива.

```
void Code_Haffmana::create_list()
```

Функция создает(заполняет) ассоциативный массив, исходя из строки.

## 5. Тестирование

№	Исходное выражение	Результат
1	fdfdfhgjdfhgd fjkghdfjkgdf gdfgdfg	100110011011100011110110111000 011011111101001110011011111101 000110110000011000011000110
2	<пустая строка>	input cant be empty!!!
3	fast push aaaaa oh noooooo plsss	100000011110111111010101011011 110011100000000000110011001110 101110010101010101110101010001 111111111
4	it's very not interesting prigram	01011010010001110111101111110 11111001010001000111010101000 01101111011111100111100100000 01010111101001101000100111001 110000
5	hello it's small string	111001111000000111110011010101 100010110010111101110010000100 101010011011010111011111

```

light5551@light5551-ThinkPad-E470: ~/AISD/7382/glazunov_sergey/lab5
fdfdhghjdfhghdfjkghdfjkgdf gdfgdfg
enter:
      7 (g)
15      8 (d)
34      9 (f)
      19
          4      2 ( )
          10      2 (k)
              6      3 (h)
                  3 (j)
Code haffmana:1001100110111000111101101110000110111110100111001101111101000110110000011000011000110
0

test4:
fast push aaaaa oh noooooo plsss
enter:
      6 (a)
13      7 (o)
32
          2      1 (f)
          4      1 (l)
          9      2 (h)
              5      2 (p)
                  3      1 (u)
                      2      1 (n)
                          1 (t)
19
      10      5 ( )
          5 (s)
Code haffmana:100000011110111110101010110111100111000000000001100110011101011100101010101110101010
001111111111

test5:
it's very not interesting prigram
enter:
      3 (n)
7      2 (g)
      4      2 (s)
15      4 (i)
      8      4 (r)
33

```

**Вывод:** Разработана программа, которая для каждой пары соответствующих открывающей и закрывающей скобок выводит номера их позиций в тексте, упорядочив пары в порядке возрастания номеров позиций: а) закрывающих и б) открывающих скобок. В результате выполнения работы были получены знания о структуре программ языка C++, стеках и очередях, потоках, а также изучены синтаксис и типы данных.

## Приложение

### Код программы

```
#include "Code_Haffmana.h"

Code_Haffmana::Code_Haffmana()
{
}

Code_Haffmana::~Code_Haffmana()
{
}

void Code_Haffmana::get_input()
{
    cout << input << endl;
}

void Code_Haffmana::set_input()
{
    getline(cin,input);
    len_of_input = input.length();
    if(!len_of_input)
    {
        cout<<"string cant be empty\n";
        exit(0);
    }
}

void Code_Haffmana::create_code_haffmana()
{
}

void Code_Haffmana::count_symbol_frequency()
{
    for (int i = 0; i < len_of_input; i++)
    {
        symbol_frequency[input[i]]++;
    }
}

void Code_Haffmana::create_list()
{
    for (iter=symbol_frequency.begin(); iter!=symbol_frequency.end(); iter++)
    {
        Node*cur = new Node(iter->first,iter->second);
        myList.push_back(cur);
    }
}

void Code_Haffmana::create_tree_of_haffmana()
```

```

{
while (myList.size()!=1)
{
myList.sort(My_Comparator_for_list());
Node*left = myList.front();
myList.pop_front();
Node*right = myList.front();
myList.pop_front();
Node*new_node = new Node(left,right);
myList.push_front(new_node);
}
root_of_tree_haffmana = myList.front();
}

void Code_Haffmana::print(Node * root, int pad)
{
if (root != nullptr)
{
print(root->left, pad + 3);
for (int i = 0;i < pad;i++)
{
cout << " ";
}
if (root->symbol) cout <<root->number<<" ("<< root->symbol<<")"<< endl;
else cout << root->number << endl;
print(root->right,pad+3);
}
}

void Code_Haffmana::create_code_of_each_symbol(Node*root)
{
if (root->left!=NULL)
{
code.push_back(0);
create_code_of_each_symbol(root->left);
}
if (root->right!=NULL)
{
code.push_back(1);
create_code_of_each_symbol(root->right);
}
if (root->left == NULL && root->right == NULL)
{
table[root->symbol] = code;
}
if(!code.empty())
code.pop_back();
}

void Code_Haffmana::print_code_haffmana()

```

```

{
    for (int i = 0; i < len_of_input; i++)
    {
        vector<bool> bin_number = table[input[i]];
        for (int j = 0; j < bin_number.size(); j++)
        {
            cout << bin_number[j];
        }
    }
}

Node* Code_Haffmana::get_root()
{
    return root_of_tree_haffmana;
}

```

#pragma once

```

#include "Node.h"
#include <iostream>
#include <string>
#include <list>
#include <map>
#include <vector>
#include <fstream>
// #define __INFILE__
using namespace std;
class Code_Haffmana
{
public:
    Code_Haffmana();
    ~Code_Haffmana();
    void get_input();
    void set_input();
    void create_code_haffmana();
    void count_symbol_frequency();
    void create_list();
    void create_tree_of_haffmana();
    void print(Node*root, int pad);
    void create_code_of_each_symbol(Node*root);
    void print_code_haffmana();
    Node* get_root();
private:
    string input;
    list<Node*>myList;
    map<char, int>symbol_frequency;
    map<char, int>::iterator iter;
    int len_of_input;
    Node*root_of_tree_haffmana;
}

```

```
vector<bool>code;
map<char, vector<bool> >table;
};
```

```
#pragma once
```

```
class Node
{
public:
Node();
Node(char _symbol,int _number) { number = _number;symbol = _symbol; }
Node(Node*_left, Node*_right) { number = _left->number + _right->number;left = _left;right =
_right; }
~Node(){delete left;delete right;}
int number;
char symbol;
Node*left, *right;
};
struct My_Comparator_for_list
{
bool operator()(Node*fst,Node*snd)
{
return fst->number < snd->number;
}
};
```

```
#include <iostream>
```

```
#include "Code_Haffmana.h"
int main()
{
Code_Haffmana code;
cout<<"enter:"<<endl;
code.set_input();
code.count_symbol_frequency();
code.create_list();
code.create_tree_of_haffmana();
code.print(code.get_root(), 0);
code.create_code_of_each_symbol(code.get_root());
cout<<"Code haffmana:";
code.print_code_haffmana();
cout<<"\n";
return 0;
}
```



