

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

**ОТЧЕТ
По лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Кодирование**

Студент гр. 9381

Фоминенко А.Н.

Преподаватель

Фирсов М.А.

Санкт-Петербург
2020

1. Цель работы.

Познакомиться со статическим кодированием Хаффмана

2. Задание.

Вариант 3

3	Кодирование: статическое Хаффмана
---	--------------------------------------

3. Теоретические положения

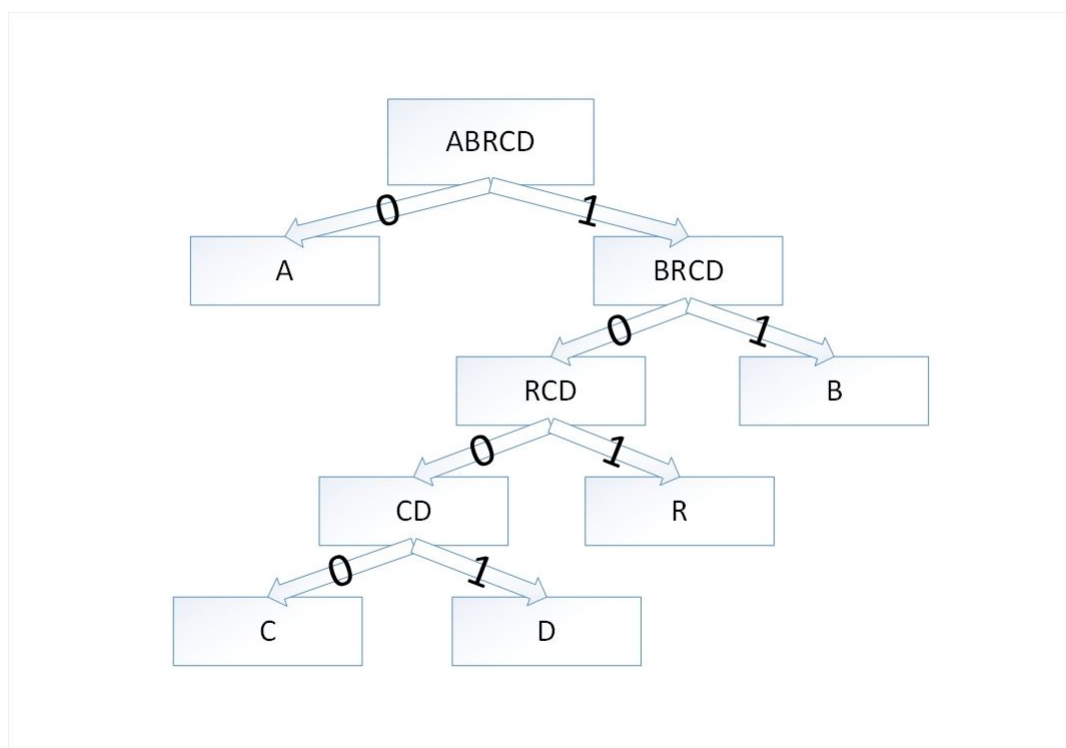
Алгоритм Хаффмана (англ. **Huffman's algorithm**) — алгоритм оптимального префиксного кодирования алфавита. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. Используется во многих программах сжатия данных, например, PKZIP 2, LZH и др.

4. Описание работы алгоритма.

Построение кода Хаффмана сводится к построению соответствующего бинарного дерева по следующему алгоритму:

1. Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента с весом, равным частоте появления символа в строке.
2. Из списка выберем два узла с наименьшим весом.
3. Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.
4. Добавим к списку только что сформированный узел вместо двух объединенных узлов.
5. Если в списке больше одного узла, то повторим пункты со второго по пятый.

пример кодирования слова "abracadabra"



5. Пример работы программы:

Основной тест №1:

Входные данные : qewqweqwwe jhdaskjhdkjash kjahsdkjhaskjhd123aksjdhasdh

Выходные данные :

Printing the order of joining nodes ::

3 (1) 2 (1)

1 (1) 32 (2)

q (3) e (3)

132 (3) w (4)

s (6) k (6)

d (6) a (6)

qe (6) j (7)

132w (7) h (8)

sk (12) da (12)

qej (13) 132wh (15)

skda (24) qej132wh (28)

skdaqej132wh (52)

Code of symbols in alphabet order ::

1 (11000) 2 (110011) 3 (110010) a (011) d (010) e (1001) h (111) j (101) k (001)
q (1000) s (000) w (1101)

Code of symbols in numeric order ::

s(000) k(001) d(010) a(011) j(101) h(111) q(1000) e(1001) w(1101) l(11000)
3(110010) 2(110011)

Encrypted message ::

10001001110110001101100110001101110110011011110100110000011011110100011010110
00111001101011111000010001101111011000001101111010110001100111100100110010001
01010111011000010111

Length :: 174

Дополнительное тестирование :

Номер теста	Входные данные	Результат
2	mama	Code of symbols in alphabet order :: a(1) m(0) Code of symbols in numeric order :: m(0) a(1) Encrypted message :: 0101
3	pizza	Code of symbols in alphabet order :: a(10) i(01) p(00) z(11)

		<p>Code of symbols in numeric order ::</p> <p>p(00) i(01) a(10) z(11)</p> <p>Encrypted message ::</p> <p>0001111110</p>
4	Mama mila ramy	<p>Code of symbols in alphabet order ::</p> <p>M(100) a(11) i(001) l(000) m(01) r(1011) y(1010)</p> <p>Code of symbols in numeric order ::</p> <p>m(01) a(11) l(000) i(001) M(100) y(1010) r(1011)</p> <p>Encrypted message ::</p> <p>1001101110100100011101111011010</p>
5	123+345 = 468	<p>Code of symbols in alphabet order ::</p> <p>+(100) 1(011) 2(010) 3(110) 4(101) 5(001) 6(000) 8(1111) =(1110)</p> <p>Code of symbols in numeric order ::</p> <p>6(000) 5(001) 2(010) 1(011) +(100) 4(101) 3(110) =(1110) 8(1111)</p> <p>Encrypted message ::</p> <p>01101011010011010100111101010001111</p>
6	message	<p>Code of symbols in alphabet order ::</p> <p>a(00) e(10) g(111) m(110) s(01)</p> <p>Code of symbols in numeric order ::</p> <p>a(00) s(01) e(10) m(110) g(111)</p> <p>Encrypted message ::</p> <p>1101001010011110</p>
7	1	<p>Code of symbols in alphabet order ::</p> <p>1(0)</p> <p>Code of symbols in numeric order ::</p> <p>1(0)</p> <p>Encrypted message ::</p>

		0
--	--	---

6. Выполнение программы:

Пользователю требуется ввести строку в файл “input.txt” и сохранить.

Запустить программу.

Посмотреть результат в файле “output.txt”

7. Описание функций и структур:

```
/**
 * структура вершины(узла)
 *
 * string str - строка в узле
 *
 * int num - номер узла(чтобы можно было пройтись dfs и отметить те
узлы где мы уже были)
 *
 * int len - длина строки узла
 *
 * int k - вес узла
 *
 * Node* right, left соответственно ссылки на правого и левого
ребенка
 */
struct Node
=====

/**
```

```

* функция обхода графа в глубину

* @param x - текущая вершина

* @param c - путь до вершины (бинарная строка)

*/

void dfs(Node *x, stack<char> &c)

=====

/**

* структура для сортировки строк

* char key - символ

* string str - строка которая ему соответствует

*/

struct MyStruct

=====

/**

* структура для сортировки Node*

* сравнивает Node* a и Node* b , что позволяет поддерживать
priority_queue

*/

struct compare

=====

/**

```



```
* основная функция, кодирующая данную строку

*/

void solve()
```

8. Вывод:

В ходе выполнения лабораторной работы была изучено и реализовано на языке C++ статическое кодирование хаффмана.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <utility>

#include <vector>

#include <string>

#include <map>

#include <algorithm>

#include <stack>

#include <iostream>

#include <fstream>

#include <queue>

#define all(a) begin(a),end(a)

using namespace std;

/**

* структура вершины(узла)
```

```

* string str - строка в узле
* int num - номер узла(чтобы можно было пройти dfs и отметить те узлы где мы уже были)
* int len - длина строки узла
* int k - вес узла
* Node* right, left соответственно ссылки на правого и левого ребенка
*/

struct Node {
    string str;

    int num{};

    int len{};

    int k{};

    struct Node *right{};

    struct Node *left{};
};

map<char, string> mp;

vector<bool> used;

/**
* функция обхода графа в глубину
* @param x - текущая вершина
* @param c - путь до вершины (бинарная строка)
*/

void dfs(Node *x, stack<char> &c) {
    used[x->num] = true;

    if (x->left != NULL && !used[x->left->num]) {
        c.push('0');

        dfs(x->left, c);

        c.pop();
    }

    if (x->right != NULL && !used[x->right->num]) {
        c.push('1');
    }
}

```

```

        dfs(x->right, c);

        c.pop();
    }

    if (x->left == NULL && x->right == NULL) {

        stack<char> t = c;

        string s;

        while (!t.empty()) s.push_back(t.top()), t.pop();

        reverse(all(s));

        mp[x->str[0]] = s;

    }

}

/**
 * структура для сортировки строк
 * char key - символ
 * string str - строка которая ему соответствует
 */

struct MyStruct {

    char key;

    std::string str;

    bool operator<(const MyStruct &struct1) const {

        if ((struct1.str.size() != str.size()))

            return struct1.str.size() > str.size();

        else {

            for (int i = 0; i < struct1.str.size(); i++) {

                if (struct1.str[i] != str[i]) {

                    return struct1.str[i] > str[i];

                }

            }

        }

    }

}

```

```

        return struct1.key > key;

    }

}

MyStruct(char k, string s) : key(k), str(move(s)) {}

};

/**
 * структура для сортировки Node*
 * сравнивает Node* a и Node* b , что позволяет поддерживать priority_queue
 */
struct compare {

    inline bool operator()(const Node* a, const Node* b) {

        if (a->k != b->k)

            return a->k > b->k;

        if(a->str.size() != b->str.size())

            return a->str.size() > b->str.size();

        for (int i = 0; i < a->str.size(); i++) {

            if (a->str[i] != b->str[i]) {

                return a->str[i] < b->str[i];

            }

        }

        return false;

    }

};

/**
 * основная функция, кодирующая данную строку
 */
void solve() {

    string s;

    getline(cin, s);

```

```

map<char, int> m;

for (char c : s) { //put chars into the map and calculate number of occurrences
    if (c != ' ')
        m[c]++;
}

int number = 0;

priority_queue<Node *, vector<Node *>, compare> q;

for (auto[i, j] : m) { //take char and create a node with it,put it in queue
    Node *x = new Node();

    x->num = number++;

    x->str.clear();

    x->str.push_back(i);

    x->len = 1;

    x->k = j;

    q.push(x);
}

cout << "Printing the order of joining nodes :: \n";

while (q.size() > 1) { //take last 2 elements of queue and join it
    Node *left, *right, *parent = new Node();

    left = q.top();

    q.pop();

    right = q.top();

    q.pop();

    cout << left->str << '(' << left->k << ") " << right->str << '(' << right->k <<
    ")\n";

    parent->left = left;

    parent->right = right;

    parent->str = left->str + right->str;

    parent->len = left->len + right->len;

    parent->num = number++;

    parent->k = left->k + right->k;

    q.push(parent);
}

```

```

}

cout << q.top()->str << '(' << q.top()->k << ")\n";

used.resize(number);

Node *parent = q.top();

stack<char> c;

dfs(parent, c);

vector<MyStruct> v;

if(m.size() == 1){

    mp[q.top()->str[0]] = '0';

}

cout << "Code of symbols in alphabet order :: \n";

for (auto[i, j] : mp) {

    v.push_back(MyStruct(i, j));

    cout << i << '(' << j << " ) ";

}

sort(all(v));


cout << '\n' << "\nCode of symbols in numeric order :: \n";

for (auto[i, j] : v) {

    cout << i << '(' << j << " ) ";

}

cout << "\n\nEncrypted message :: \n";


int len = 0;

for (char p : s) {

    len += mp[p].size();

    cout << mp[p];

}


cout << "\n\nLength ::"<< len;

```

```
}
```

```
int32_t main() {  
    freopen("../input.txt", "r", stdin);  
    freopen("../output.txt", "w", stdout);  
    solve();  
    fclose(stdin);  
    fclose(stdout);  
}
```