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 Хаффмана
Вариант 3

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с кодированием Хаффмана и реализовать на языке программирования C++ статическое кодирование Хаффмана.

Постановка задачи.

Кодирование: статическое Хаффмана.

Описание алгоритма.

Для начала программа считывает строку и запоминает частоту встречаемости каждого символа, занося данные в массив *count*. Затем сортирует данный массив в порядке убывания частоты встречаемости символа. Далее создается вектор, в котором складываются символы (то есть если символ 'a' встречается 1 раз и символ 'b' встречается 1 раз, то в вектор заносится сочетание символов "ab", а так же складывается их частота). В ходе цикла добавления этих элементов в вектор, он сортируется. Затем вызывается функция `huffmanEncoding` , в которой на основе дерева и начального массива происходит кодирование.

Описание структуры ltcode.

Структура состоит из трех полей: `unsigned long long frequency` - частота встречаемости символа, `unsigned char letter` - сам символ, `string code` - код для символа.

Описание структуры tree.

Структура состоит из пяти полей: `string list` - строка, хранящая в себе символ или сочетание символов, `int freq` - частота встречаемости символа или сочетания символов, `int left` - индекс левого поддерева, `int right` - индекс правого поддерева, `string code` - код для символа или сочетания символов.

Описание функции huffmanEncoding.

Рекурсивная функция `void huffmanEncoding(int num, string code, vector <tree>& treee, ltcode* letterCodes)` , на вход которой подается: `num` - индекс элемента вектора tree (дерево реализовано на базе массива), строка `code` - код для символа или сочетания символов, вектор `tree` типа B, элемент `letterCodes` структуры `ltcode`. Функция начинается с того, что записывает код символа в поле `code` вектора `treee`. Затем, если в данном векторе хранится сочетание символов, то вызывается эта же функция для левого поддерева, при этом добавляя к коду "0", а потом для правого поддерева, добавляя к коду "1". Если же в векторе хранится не сочетание символов, значит в нем хранится одиночный символ. В таком случае в элементе `letterCodes` структуры `ltcode` производится поиск данного элемента. Когда элемент находится, в поле `code` этой структуры заносится код для символа.

Спецификация программы.

Программа написана на языке C++. Входные данные подаются либо из терминала, либо из файла. Результат программы выводится на экран.

Тестирование.

```
Как вы хотите ввести строку? 1 - из файла, 2 - из терминала
1
Введите путь до txt файла
test.txt
Содержимое файла:
abracadabra
Частота встречаемости символа 'a' - 5
Частота встречаемости символа 'b' - 2
Частота встречаемости символа 'r' - 2
Частота встречаемости символа 'c' - 1
Частота встречаемости символа 'd' - 1

Частота: 11 , сочетание символов: arcdb , код:
Рекурсивный вызов функции для левого поддерева:
Частота: 5 , сочетание символов: a , код: 0
Это единичный символ
Рекурсивный вызов функции для правого поддерева:
Частота: 6 , сочетание символов: rcdb , код: 1
Рекурсивный вызов функции для левого поддерева:
Частота: 2 , сочетание символов: r , код: 10
Это единичный символ
Рекурсивный вызов функции для правого поддерева:
Частота: 4 , сочетание символов: cdb , код: 11
Рекурсивный вызов функции для левого поддерева:
Частота: 2 , сочетание символов: cd , код: 110
Рекурсивный вызов функции для левого поддерева:
Частота: 1 , сочетание символов: c , код: 1100
Это единичный символ
Рекурсивный вызов функции для правого поддерева:
Частота: 1 , сочетание символов: d , код: 1101
Это единичный символ

Рекурсивный вызов функции для правого поддерева:
Частота: 2 , сочетание символов: b , код: 111
Это единичный символ


Дерево:
            b
           /
         cdb
          \
              d
             /
           cd
             \
              c
       /
     /
   rcdb
     \
      r
  /
 /
arcdb
  \
   a


Коды для букв:
'a' - 0  'b' - 111  'r' - 10  'c' - 1100  'd' - 1101

Закодированная строка:
0 111 10 0 1100 0 1101 0 111 10 0
Количество бит: 23
```

Input	Output
HeLLo WORLD!	1101001101001001100001110111111011000111 39 бит L - 10, ' ' - 0110, '!' - 0111, D - 1100, H - 1101, O - 1110, R - 1111, W - 000, e - 001, o - 010
123$$$$123	11011110000011011110 20 бит '$' - 0, 1 - 110, 2 - 111, 3 - 10
rooj rnna rccc	01111111110011010111011010001010100000038 бит C - 00, r - 01, ' ' - 101, n - 110, o - 111, a - 1000, j - 1001
Prog_C++	0011011000110100001111 22 бита '+' - 11, C - 000, P - 001, '_' - 010, g - 011, o - 100, r - 101

Вывод.

В ходе выполнения лабораторной работы были приобретены навыки в кодировании Хаффмана, а так же написана программа для статического кодирования Хаффмана.

ПРИЛОЖЕНИЕ.

КОД ПРОГРАММЫ.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <fstream>

using namespace std;

struct ltcode {
	unsigned long long frequency = 0;	//частота встречаемости символа
	unsigned char letter = NULL;		//буква
	string code = "";			//код для буквы
};

struct tree {
	string list;	//сочетание символов
	int freq;		//частота встречаемости символа/сочетания символов

	int left;	//индекс левого поддерева
	int right;	//индекс правого поддерева

	string code = "";	//код для сочетания символов

	tree() {		//конструктор
		left = 0;
		right = 0;
		list = "";
		freq = 0;
	}
};

void zeroset(ltcode* c);
bool cmpForStr(ltcode x, ltcode y);
bool cmpForTree(tree x, tree y);
void huffmanEncoding(int num, string code, vector <tree>& treee, ltcode* letter_codes);
char reading(ifstream &in);
void printTree(int num, vector <tree>& treee, int level = 0);

int main() {
	setlocale(LC_ALL, "Russian");
	char c = NULL;					//переменная для посимвольного считывания
	string S;					//строка, которую вводит пользователь
	ltcode* count = new ltcode[256];		//выделяем память под массив типа ltcode, который хранит в себе частоту встречаемости введенных символов
	ltcode* stringLinks = new ltcode[256];
	ifstream in;
	char how = NULL;
	string filePath;

	cout << "Как вы хотите ввести строку? 1 - из файла, 2 - из терминала\n";	//выбор ввода строки
	how = getchar();
	if (how > '2' || how < '1') {
		cout << "Неверно выбран вариант ввода\n";
		return 0;
	}
	if (how == '1') {
		cout << "Введите путь до txt файла\n";
		cin >> filePath;
```

```
		in.open(filePath);
		if (!in.is_open()) {
			cout << "Неверно введен путь до файла\n";
			return 0;
		}
		string file;
		getline(in, file);
		cout << "Содержимое файла:\n" << file << endl;
		in.close();
		in.open(filePath);
	}
	getchar();
	zeroset(count);			//обнуление всех переменных массива структур count

	while (c = reading(in)) {					//производится посимвольное
считывание строки с подсчетом частоты встречаемости
		if (c == '\n' || in.eof()) break;
		S += c;
		++count[c].frequency;
	}
	sort(count, count + 256, cmpForStr);			//сортировка элементов массива
count в порядке убывания частоты встречаемости символа

	//вывод элементов и частоты
	for (int i = 0; count[i].frequency; i++) {
		cout << "Частота встречаемости символа '" << count[i].letter << "' - " <<
count[i].frequency << endl;
	}

	vector <tree> treee;
	tree tmp;
	tmp.list = "0";
	int j = 0;
	for (j = 0; count[j].frequency; j++) {
		tmp.freq = count[j].frequency;
		tmp.list[0] = count[j].letter;
		treee.push_back(tmp);				//добавление в конец вектора
	}
	int maxSize = j;
	int size = 0;

	if (maxSize == 1) {
		count[0].code = "0";
		tmp.list = count[0].letter;
		tmp.freq = count[0].frequency;
	}
	else {
		sort(treee.begin(), treee.end(), cmpForTree);	//сортировка дерева в порядке
возрастания частоты встречаемости

		for (j = 0; size < maxSize; ) {
			tmp.list = treee[j].list + treee[j + 1].list;		//складываем
символы или сочетания символов
			tmp.freq = treee[j].freq + treee[j + 1].freq;
//складываем частоты встречаемости
			tmp.left = j;
			tmp.right = j + 1;

			size = tmp.list.size();

			treee.push_back(tmp);
			j += 2;
			sort(treee.begin() + j, treee.end(), cmpForTree);	//сортируем в
порядке возрастания частоты все элементы, кроме тех, которые уже прошли проверку
```

```
			}

			//!!!функция кодирования
			cout << endl;
			huffmanEncoding(treee.size() - 1, "", treee, count);
		}

		cout << "Дерево:\n";
		printTree(treee.size() - 1, treee);
		cout << endl;

		cout << "\nКоды для букв:\n";
		for (j = 0; (count[j].frequency); ++j) {
			cout << '\'' << count[j].letter << "' - " << count[j].code << "  ";
			stringLinks[count[j].letter].code = count[j].code;
		}

		cout << "\n\nЗакодированная строка:\n";
		size = 0;
		for (unsigned int i = 0; i < S.size(); ++i) {
			cout << stringLinks[S[i]].code << " ";
			size += stringLinks[S[i]].code.size();
		}
		cout << "\nКоличество бит: " << size << endl;

		delete[] count;
		delete[] stringLinks;
		return 0;
}

void zeroset(ltcode* c) {				//функция, обнуляющая массив структур с
		for (int i = 0; i < 256; i++) {
			c[i].frequency = 0;
			c[i].letter = i;
		}
}

bool cmpForStr(ltcode x, ltcode y) {			//компаратор для сортировки строки
		if (x.frequency != y.frequency) return x.frequency > y.frequency;
		else return x.letter < y.letter;
}

bool cmpForTree(tree x, tree y) {			//компаратор для сортировки дерева
		if (x.freq != y.freq) return x.freq < y.freq;
		else return x.list.size() > y.list.size();
}

void huffmanEncoding(int num, string code, vector <tree>& treee, ltcode* letterCodes) {
		cout << "Частота: " << treee[num].freq << " , сочетание символов: " <<
treee[num].list << " , код: " << code << endl;
		treee[num].code = code;
		if (treee[num].list.size() > 1) {
			cout << "Рекурсивный вызов функции для левого поддерева:\n";
			huffmanEncoding(treee[num].left, code + "0", treee, letterCodes);
			cout << "\nРекурсивный вызов функции для правого поддерева:\n";
			huffmanEncoding(treee[num].right, code + "1", treee, letterCodes);
			cout << endl;
		}
		else {
			cout << "Это единичный символ";
			for (int j = 0; ; j++) {				//поиск в элементе
letterCodes структуры ltcode нужного символа и запись туда кода
				if (letterCodes[j].letter == treee[num].list[0]) {
					letterCodes[j].code = code;
```

```
                        break;
                    }
                }
            }
        }

char reading(ifstream &in) {
        char c;
        if (in.is_open()) c = in.get();
        else c = getchar();
        return c;
}

void printTree(int num, vector <tree>& treee, int level) {
        if (treee[num].list.size() > 1) {
                if (treee[treee[num].right].list.size() >= 1) printTree(treee[num].right,
treee, level + 4);
                if (level) {
                        for (int i = 0; i < level; i++) cout << " ";
                        cout << " ";
                }
                if (treee[treee[num].right].list.size() >= 1) {
                        cout << " /\n";
                        for (int i = 0; i < level; i++) cout << " ";
                        cout << " ";
                }
                cout << treee[num].list << "\n ";
                if (treee[treee[num].left].list.size() >= 1) {
                        for (int i = 0; i < level; i++) cout << " ";
                        cout << " " << " \\\n";
                        printTree(treee[num].left, treee, level + 4);
                }
        }
        else if (treee[num].list.size() == 1) {
                if (level) {
                        for (int i = 0; i < level; i++) cout << " ";
                }
                cout << treee[num].list << "\n ";
        }
}
```