**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по лабораторной работе № 2**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант №2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8301 | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Пискунович К.В. |
| Преподаватель | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Тутуева А.В. |

Санкт-Петербург

2020

**Цель работы**

Реализовать кодирование и декодирование по алгоритму входной строки, вводимой через консоль.

**Описание реализованного алгоритма и использованных структур данных**

**Map** – класс реализующий ассоциативный массив на основе красно-чёрного дерева

**List**(1 аргумент) - класс списка, который хранит один элемент

**List**(2 аргумента)- класс списка, который хранит два элемента

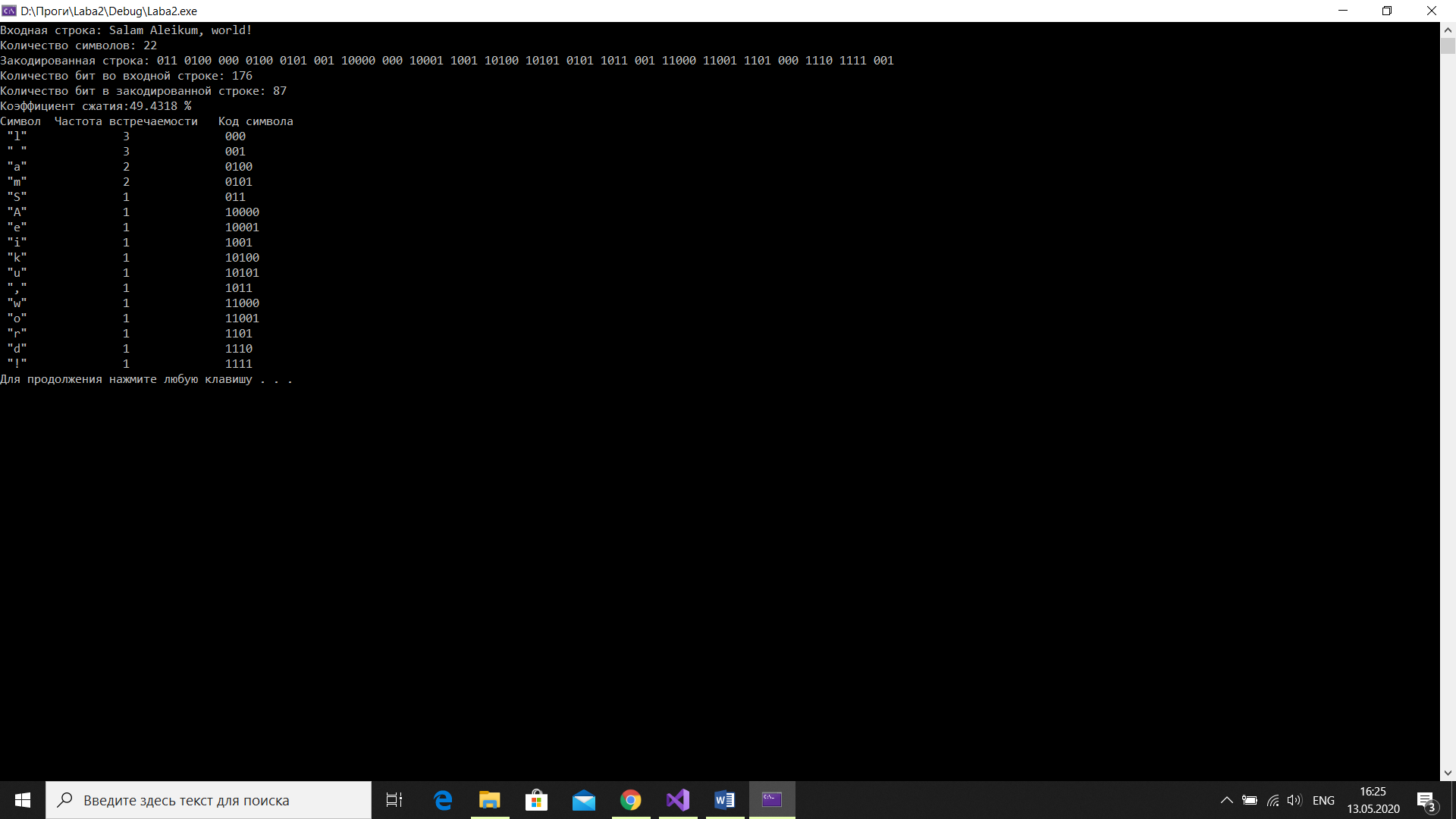
**Shannon\_Phano –** основной класс, в котором и реализован алгоритм кодирования и декодирования

* *List<bool> getEncodedString() – метод, который возвращает список нулей и единиц, получившихся в результате кодирования.*
* *List<char> decode(string) – метод, осуществляющий декодирование нулей и единиц, переданных как аргумент на основании строки, переданной методом void Convertion\_input\_string(string) и возвращающий декодированную строку;*
* *void Convertion\_input\_string(string) – метод, принимающий строку, кодирование и осуществляет её кодирование;*
* *void show\_on\_display() – метод, который будет выводить всю информацию о кодировании строки, занесенную функцией void Convertion\_input\_string(string);*

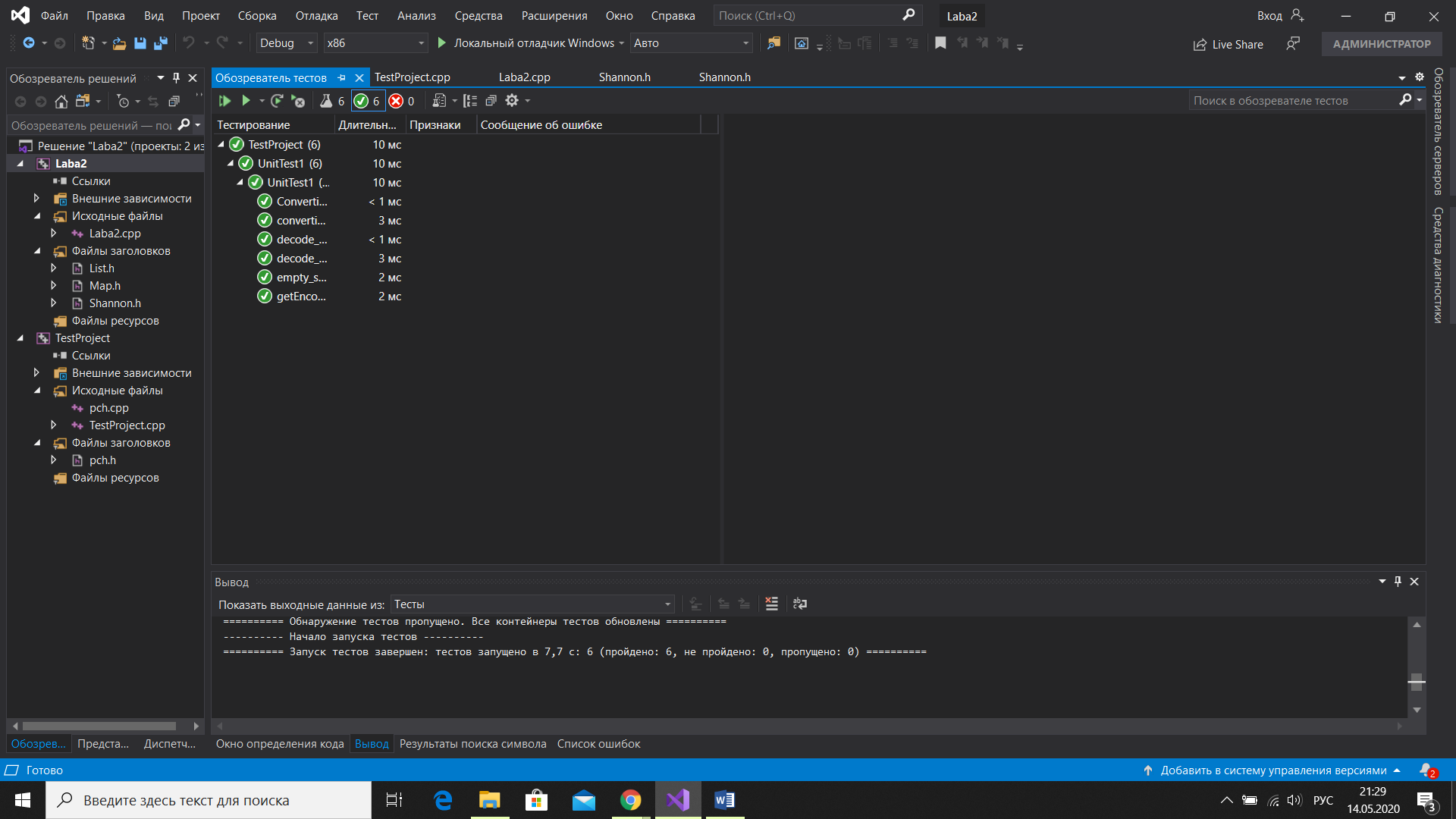
**Оценка временной сложности каждого метода.**

* List<char> decode(std::string) – O(n);
* List<bool> getEncodedString() – O(1);
* void *Convertion\_input\_string(string)*– O(n);
* void *show\_on\_display()*  – O(n);

**Примеры работы программы**



**Подтверждение юнит тестов**



**Листинг**

*Shannon\_Phano.cpp*

#include <iostream>

#include <string>

#include "List.h"

#include "Map.h"

using namespace std;

class Shannon\_Fano

{

private:

List<bool> encoded\_string; // Хранит закодированную в 0 и 1 строку

map<char, float, List<bool>> map; //Хранит ключ, его встречаемость (значение 1), код символа (значеие 2)

List<bool> code\_symbols; // Список, который хранит код, символа

List<pair<char, float>> symbols\_and\_their\_count; // Список, который хранит сами символы, а также количество этих символов в строке

void set\_codes(size\_t indexL, size\_t indexR);

public:

string input\_string;

Shannon\_Fano() = default;

void display();

List<bool> getEncodedString();

void convert(string str);

List<char> decode(string str);

};

List<char> Shannon\_Fano::decode(string str)

{

if (input\_string.empty())

throw runtime\_error("Input string is empty");

string encoded\_string\_to\_decode = str;

if (encoded\_string\_to\_decode.empty())

throw runtime\_error("The string is empty");

List<char> decoded\_string;

List<bool> temp\_code;

for (int i = 0; i < encoded\_string\_to\_decode.length(); i++)

{

temp\_code.push\_back(encoded\_string\_to\_decode.at(i) - 48);

if (map.code(temp\_code))

{

decoded\_string.push\_back(map.find\_key(temp\_code));

temp\_code.clear();

}

}

if (!temp\_code.isEmpty())

{

throw runtime\_error("The string is wrong");

}

return decoded\_string;

}

void Shannon\_Fano::convert(string str)

{

if (str.empty())

throw runtime\_error("Input string is empty");

input\_string = str;

for (int i = 0; i < input\_string.length(); i++)

{

map.insert(input\_string.at(i), float(1), float(1));

}

for (int i = 0; i < input\_string.length(); i++)

{

symbols\_and\_their\_count.insert\_with\_sorting(input\_string.at(i), map.find\_value1(input\_string[i]));

}

if (symbols\_and\_their\_count.get\_size() == 1)

{

code\_symbols.push\_back(0);

map.set\_value2(input\_string.at(0), code\_symbols);

}

else

set\_codes(0, symbols\_and\_their\_count.get\_size());

for (size\_t i = 0; i < input\_string.length(); i++)

{

encoded\_string.push\_back(map.find\_value2(input\_string.at(i)));

}

}

List<bool> Shannon\_Fano::getEncodedString()

{

return encoded\_string;

}

void Shannon\_Fano::display()

{

if (input\_string.empty())

throw runtime\_error("There is no string");

cout << "Входная строка: ";

cout << input\_string << endl;

cout << "Количество символов: ";

cout << input\_string.length() << endl;

cout << "Закодированная строка: ";

for (size\_t i = 0; i < input\_string.length(); i++)

{

map.find\_value2(input\_string.at(i)).print\_to\_console();

cout << " ";

}

cout << '\n';

cout << "Количество бит во входной строке: " << input\_string.length() \* 8 << endl;

cout << "Количество бит в закодированной строке: " << encoded\_string.get\_size() << endl;

cout << "Коэффициент сжатия:" << (float)encoded\_string.get\_size() \* 100 / (input\_string.length() \* 8) << " %" << endl;

cout << "Символ" << "\t" << "Частота встречаемости" << "\t" << "Код символа" << endl;

for (int i = 0; i < symbols\_and\_their\_count.get\_size(); i++)

{

cout << " \"" << symbols\_and\_their\_count.at(i).first << "\"\t " << symbols\_and\_their\_count.at(i).second << " ";

map.find\_value2(symbols\_and\_their\_count.at(i).first).print\_to\_console();

cout << endl;

}

}

void Shannon\_Fano::set\_codes(size\_t indexL, size\_t indexR)

{

if (indexR - indexL == 2)

{

/\* Осталось два символа, которые не получили свой код \*/

code\_symbols.push\_back(0);

map.set\_value2(symbols\_and\_their\_count.at(indexL).first, code\_symbols);

code\_symbols.pop\_back();

code\_symbols.push\_back(1);

map.set\_value2(symbols\_and\_their\_count.at(indexR - 1).first, code\_symbols);

code\_symbols.pop\_back();

return;

}

else if (indexR - indexL == 1)

{

/\* Остался один символ, который не получил свой код\*/

map.set\_value2(symbols\_and\_their\_count.at(indexL).first, code\_symbols);

return;

}

else

{

float dS = 0, sum = 0;

for (int i = indexL; i < indexR; i++)

{

dS += symbols\_and\_their\_count.at(i).second;

}

dS /= 2;

size\_t newIndex = indexL;

for (int i = indexL; i < symbols\_and\_their\_count.get\_size(); i++)

{

sum += symbols\_and\_their\_count.at(i).second;

newIndex++;

if (sum >= dS)

{

break;

}

}

code\_symbols.push\_back(0);

set\_codes(indexL, newIndex);

code\_symbols.pop\_back();

code\_symbols.push\_back(1);

set\_codes(newIndex, indexR);

code\_symbols.pop\_back();

}

}

**unittest1.cpp**

#pragma once

#include "CppUnitTest.h"

#include <string>

#include <stdexcept>

#include "../Laba2/Shannon.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(Convertion\_input\_string\_test)

{

Shannon\_Fano example;

string exampleStr = "Иллюзия";

example.convert(exampleStr);

Assert::AreEqual(example.input\_string, exampleStr);

}

TEST\_METHOD(getEncodedString\_test)

{

Shannon\_Fano example;

string exampleStr = "Иллюзия", outString;

example.convert(exampleStr);

List<bool> list = example.getEncodedString();

string TestString = "010000001110010111"; // "Иллюзия"

for (size\_t i = 0; i < list.get\_size(); i++)

{

if (list.at(i))

outString = outString + "1";

else

outString = outString + "0";

}

Assert::AreEqual(TestString, outString);

}

TEST\_METHOD(decode\_test)

{

Shannon\_Fano example;

string exampleStr = "Иллюзия", outString;

example.convert(exampleStr);

List<char> list = example.decode("00011100"); // "люз"

string TestString = "люз";

for (size\_t i = 0; i < list.get\_size(); ++i)

outString = outString + list.at(i);

Assert::AreEqual(TestString, outString);

}

TEST\_METHOD(convertion\_with\_empty\_string)

{

Shannon\_Fano example;

string exampleStr = "";

try

{

example.convert(exampleStr);

}

catch (const exception &ex)

{

Assert::AreEqual("Input string is empty", ex.what());

}

}

TEST\_METHOD(decode\_with\_empty\_string)

{

Shannon\_Fano example;

try

{

example.display();

}

catch (const exception& ex)

{

Assert::AreEqual("There is no string", ex.what());

}

}

TEST\_METHOD(empty\_string\_in\_decode)

{

string exampleStr = "";

Shannon\_Fano example;

try

{

example.decode(exampleStr);

}

catch (const runtime\_error error)

{

Assert::AreEqual("Input string is empty", error.what());

}

}

};

}