|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Теоретической и прикладной математики | | |
|  | | |
| Лабораторная работа № 2 | | |
| по дисциплине «ОСНОВЫ ТЕОРИИ ИНФОРМАЦИИИ КРИПТОГРАФИИ» | | |
|  | | |
| **ОСНОВНЫЕ МЕТОДЫ ПОБУКВЕННОГО КОДИРОВАНИЯ** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-02 |
| Вариант: | 7 |
| Студент: | Сидоров Даниил, |
|  | Дюков Богдан |
| Преподаватель: | Авдеенко Татьяна Владимировна, |
|  | Сивак Мария Алексеевна. |
|
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |

1. **Цель р****а****боты**

Освоить основные алгоритмы побуквенного кодирования.

1. **Задача**
2. Реализовать приложение для кодирования с помощью заданного в варианте алгоритма:
   1. вероятности появления символов алфавита должны храниться в одном файле, а последовательность, подлежащая кодированию, – в другом;
   2. закодированный текст должен сохраняться в файл;
   3. приложение должно:

* выводить полученные кодовые слова для всех символов алфавита;
* вычислять среднюю длину кодового слова;
* вычислять избыточность;
* проверять неравенство Крафта.

1. Реализовать приложение для декодирования с помощью заданного в варианте алгоритма:
2. вероятности появления символов алфавита должны храниться в одном файле, а закодированная последовательность – в другом;
3. раскодированная последовательность должна сохраняться в файл.
4. Рассмотреть 3 распределения вероятностей символов алфавита: равномерное, *P*1(*A*) и *P*2(*A*). Для каждого распределения получить кодовые слова, вычислить среднюю длину кодового слова, избыточность и проверить неравенство Крафта.
5. С помощью реализованных приложений исследовать зависимость получаемых кодовых слов от распределения вероятностей символов алфавита:
   1. смоделировать последовательность, символы которой подчиняются равномерному распределению. Закодировать эту последовательность при равномерном, *P*1(*A*) и *P*2(*A*) распределениях и вычислить длину каждой из трёх закодированных последовательностей;
   2. смоделировать последовательность, символы которой подчиняются распределению *P*1(*A*). Закодировать эту последовательность при равномерном, *P*1(*A*) и *P*2(*A*) распределениях и вычислить длину каждой из трёх закодированных последовательностей;
   3. смоделировать последовательность, символы которой подчиняются распределению *P*2(*A*). Закодировать эту последовательность при равномерном, *P*1(*A*) и *P*2(*A*) распределениях и вычислить длину каждой из трёх закодированных последовательностей;
   4. сделать выводы о связи распределения символов последовательности, вероятностях символов, используемых при кодировании, и длины получившегося кода.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Алфавит источника** | **Вероятности появления букв** | | | **Алгоритм** |
| **Равномерное распределение** | **P1(A)** | **P2(A)** |
| А, Б, В, Г, Д, Е, Ж, З | 0.125, 0.125, 0.125, 0.125,  0.125, 0.125, 0.125, 0.125 | 0.1,0.01,0.09,  0.4,0.4,0,0,0 | 2-2,2-5,2-1,,2-3,0,0,0 | Хаффмана |

1. **Разработанное программное средство**

**Программа алгоритм кодирования и декодирования Хаффмана:**

1. **Окно кодирование:**

На вход программе подаются: имя файла с символами алфавита и вероятностями их появления, имя файла с текстовой последовательностью, подлежащей кодированию, и имя файла, куда будет записан закодированный текст.

В случае успеха программа выводит в заданные поля данные: закодированные слова, среднюю длину кодового слова, избыточность, результат неравенства Крафта (выполнение или не выполнение) и энтропию. Также в поле “Кодовые слова” выводится каждый символ алфавита и его кодовое слово.

В случае неудачи (ошибка при открытии файла) программа выводит соответствующее сообщение.

**Программа декодирования:**

На вход программе подаются: имя файла с символами алфавита и вероятностями их появления, имя файла с текстовой последовательностью, подлежащей декодированию, и имя файла, куда будет записан декодированный текст.

В случае успеха программа выводит в заданное поле декодированный текст. Также в поле “Кодовые слова” выводится каждый символ алфавита и его кодовое слово

В случае неудачи (ошибка при открытии файла) программа выводит соответствующее сообщение.

1. **Исследования**

**Распределение вероятностей символов алфавита**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тип распределения** | **Кодовая таблица** | **Средняя длина кодового слова** | **Избыточность** | **Выполнение неравенства Крафта** |
| Равномерное | А 000  Б 011  В 001  Г 111  Д 110  Е 101  Ж 100  З 010 | 3.0 | 0.0 | Выполняется |
| *P1(A)* | А 101  Б 10001  В 1001  Г 0  Д 11  Е 1000010  Ж 1000011  З 100000 | 1.91 | 0.141172 | Выполняется |
| *P2(A)* | А 01  Б 00001  В 1  Г 0001  Д 001  Е 0000010  Ж 0000011  З 000000 | 1.875 | 0.0394824 | Выполняется |

**Длины закодированных последовательностей**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Равномерное** | ***P1(A)*** | ***P2(A)*** |
| **Равномерное** | 276 | 403 | 403 |
| ***P1(A)*** | 300 | 191 | 314 |
| ***P2(A)*** | 298 | 324 | 188 |

В заголовках столбцов указаны типы распределения, которым подчиняются символы кодируемой последовательности, в заголовках строк – типы распределений, используемых при кодировании.

**Вывод:**

1. Более вероятным буквам источника поставлены в соответствие более короткие кодовые слова.
2. Средняя длина кодового слова минимальна при распределении P2(A).
3. Избыточность минимальна при равномерном распределении.
4. Минимальная длина закодированной последовательности достигается, когда распределение, которой подчиняется последовательность, совпадает с распределением при кодировании.
5. Наилучший результат происходит при распределениях P2(A). Это связано с тем, что средняя длина кодового слова минимальна именно при этом распределении.
6. Наихудший результат происходит при равномерных распределениях. Это связано с тем, что средняя длина кодового слова максимальна именно при этом распределении.
7. Наилучший результат при разных распределениях происходит при последовательности, которая подчиняется распределению P2(A), и при кодировании с равномерным распределением.
8. Наихудший результат при разных распределениях происходит при последовательности, которая подчиняется равномерному распределению, и при кодировании с распределением P1(A) и P2(A).
9. **Код программы**

Заголовочный файл EncodingAndDecodingFunctions.h:

#pragma once

#ifndef \_EncodingAndDecodingFunctions\_H

#define \_EncodingAndDecodingFunctions\_H

#include <msclr\marshal\_cppstd.h>

#include <iostream>

#include <string>

#include <queue>

#include <unordered\_map>

#include <fstream>

#include <sstream>

// Узел дерева

struct Node

{

std::string symbol;

double chance;

Node\* left;

Node\* right;

};

// Компаратор, который будет использоваться для упорядочивания узлов в очереди

struct comp

{

bool operator()(Node\* l, Node\* r)

{

return l->chance > r->chance;

}

};

// Создание нового узла дерева

Node\* GetNewNode(std::string ch, double freq, Node\* left, Node\* right);

// Определение кодовых слов из дерева и запись их в контейнер huffmanCode по ключу соответствующего символа

void EncodingSymbolsFromTree(Node\* root, std::string encodedText, std::unordered\_map<std::string, std::string>& huffmanCode);

// Получение закодированного текста

std::string GetEncodedText(std::unordered\_map<std::string, std::string> huffmanCode, std::string originalText);

// Расшифровка кодовых слов из дерева и последовательная запись символов в decodingText

void DecodingCodesFromTree(Node\* root, int& index, std::string decodedText, std::string& decodingText);

// Построение Хаффмановского дерева (функция возвращает адрес корня дерева)

Node\* BuildHuffmanTree(std::unordered\_map <std::string, double > symbolAndChance);

// Получение текста (последовательность алфавитных букв или кодов) из файла

std::string GetTextFromFile(std::string fileName);

// Получение символов алфавита и вероятности их появления

std::unordered\_map<std::string, double> GetAlphabet(std::string fileName);

// Запись текста (закодированного или декодированного) в файл

void WriteTextToFile(std::string fileName, std::string text);

// Проверка неравенства Крафта

bool CheckingCraftInequality(std::unordered\_map<std::string, std::string> huffmanCode);

// Получение энтропии

double GetEntrophy(std::unordered\_map<std::string, double> alphabet);

// Получение средней длины кодового слова

double GetAverageLength(std::unordered\_map<std::string, std::string> huffmanCode, std::unordered\_map<std::string, double> alphabet);

// Получение избыточности

double GetRedundancy(double entrophy, double averageLenth);

#endif

Файл EncodingAndDecodingFunctions.h:

#include "EncodingAndDecodingFunctions.h"

// Создание нового узла дерева

Node\* GetNewNode(std::string ch, double freq, Node\* left, Node\* right)

{

Node\* node = new Node();

node->symbol = ch;

node->chance = freq;

node->left = left;

node->right = right;

return node;

}

// Определение кодовых слов из дерева (запись кодовых слов в контейнер huffmanCode по ключу соответствующего символа)

void EncodingSymbolsFromTree(Node\* root, std::string encodedText, std::unordered\_map<std::string, std::string>& huffmanCode)

{

if (root == nullptr)

{

return;

}

// Дошли до листа, записали кодовое слово

if (!root->left && !root->right)

{

huffmanCode[root->symbol] = encodedText;

}

EncodingSymbolsFromTree(root->left, encodedText + "0", huffmanCode);

EncodingSymbolsFromTree(root->right, encodedText + "1", huffmanCode);

}

// Получение закодированного текста (генерация последовательности из значений, взятых из контейнера по ключам,

// равным символам исходного текста)

std::string GetEncodedText(std::unordered\_map<std::string, std::string> huffmanCode, std::string originalText)

{

std::string encodingText;

std::string firstSymbol;

while(originalText != "")

{

firstSymbol = originalText.substr(0, 1);

encodingText += huffmanCode[firstSymbol];

originalText.erase(0, 1);

}

return encodingText;

}

// Расшифровка кодовых слов из дерева и последовательная запись символов в decodingText

void DecodingCodesFromTree(Node\* root, int& index, std::string encodingText, std::string& decodingText)

{

if (root == nullptr)

{

return;

}

// Дошли до листа, записали символ

if (!root->left && !root->right)

{

decodingText += root->symbol;

return;

}

index++;

if (encodingText[index] == '0')

DecodingCodesFromTree(root->left, index, encodingText, decodingText);

else

DecodingCodesFromTree(root->right, index, encodingText, decodingText);

}

// Построение Хаффмановского дерева (функция возвращает адрес корня дерева)

Node\* BuildHuffmanTree(std::unordered\_map<std::string, double> symbolAndChance)

{

// Приоритетная очередь для хранения активных узлов

std::priority\_queue<Node\*, std::vector<Node\*>, comp> activeNodes;

// Добавление в приоритетную очередь созданных узлов для каждого символа последовательности

for (auto elem : symbolAndChance)

{

activeNodes.push(GetNewNode(elem.first, elem.second, nullptr, nullptr));

}

// Пока в очереди более 1 узла:

// 1) Убираем из очереди пару узлов, содержащих символы с минимальными вероятностями

// 2) Помещаем в очередь новый узел с этими двумя узлами в качестве дочерних и вероятностью, равной сумме вероятностей обоих узлов

while (activeNodes.size() != 1)

{

Node\* left = activeNodes.top();

activeNodes.pop();

Node\* right = activeNodes.top();

activeNodes.pop();

double sum = left->chance + right->chance;

activeNodes.push(GetNewNode("\0", sum, left, right));

}

return activeNodes.top();

}

// Получение закодированного текста из файла

std::string GetTextFromFile(std::string fileName)

{

std::ifstream file;

file.open(fileName);

// Eсли файл не открыт, генерируем исключение

if (!file.is\_open())

{

throw 0;

}

std::string encodedText;

file >> encodedText;

file.close();

return encodedText;

}

// Получение символов алфавита и вероятности их появления

std::unordered\_map<std::string, double> GetAlphabet(std::string fileName)

{

std::ifstream alphabetFile;

alphabetFile.open(fileName);

// Eсли файл не открыт, генерируем исключение

if (!alphabetFile.is\_open())

{

throw 0;

}

std::string symbol;

double chance;

std::unordered\_map<std::string, double> alphabet;

while (!alphabetFile.eof())

{

alphabetFile >> symbol;

alphabetFile >> chance;

alphabet[symbol] = chance;

}

alphabetFile.close();

return alphabet;

}

// Запись декодированного текста в файл

void WriteTextToFile(std::string fileName, std::string text)

{

std::ofstream file;

file.open(fileName);

// Eсли файл не открыт, генерируем исключение

if (!file.is\_open())

{

throw 0;

}

file << text;

file.close();

}

// Проврка неравенства Крафта

bool CheckingCraftInequality(std::unordered\_map<std::string, std::string> huffmanCode)

{

auto k = huffmanCode.size();

double sum = 0.0;

for (auto elem : huffmanCode)

sum += pow(2.0, -1.0 \* elem.second.length());

return sum <= 1;

}

// Получение энтропии

double GetEntrophy(std::unordered\_map<std::string, double> alphabet)

{

double entrophy = 0.0;

for (auto elem : alphabet)

if (elem.second > 0.0)

entrophy += elem.second \* log2(elem.second);

return -1.0 \* entrophy;

}

// Получение средней длины

double GetAverageLength(std::unordered\_map<std::string, std::string> huffmanCode, std::unordered\_map<std::string, double> alphabet)

{

double averageLength = 0.0;

for (auto elem : alphabet)

averageLength += elem.second \* huffmanCode[elem.first].length();

return averageLength;

}

// Получение избыточности

double GetRedundancy(double entrophy, double averageLenth)

{

return averageLenth - entrophy;

}

Файл DecodingForm.h (только обработчики событий):

#pragma endregion

// Событие при нажатии на кнопку "Вставить текст из файла"

// Происходит запись в текстбокс с закодированной последовательностью содержимого файла SavingEncoding.txt

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

textBox2->Text = gcnew System::String(GetTextFromFile("SavingEncodingText.txt").c\_str());

}

catch(int item)

{

if(item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

}

// Событие при нажатии на кнопку "Декодировать"

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::unordered\_map<std::string, double> alphabet;

try

{

// Берем из файла символы алфавита и вероятности их появления, в зависимости от выбранного распределения

if (comboBox1->Text == L"Равномерное")

alphabet = GetAlphabet("Alphabet.txt");

else if (comboBox1->Text == L"P1(A)")

alphabet = GetAlphabet("AlphabetP1(A).txt");

else

alphabet = GetAlphabet("AlphabetP2(A).txt");

}

catch (int item)

{

if (item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

// Строим Хаффмановское дерево, а после определяем кодовые слова из дерева

auto huffmanTree = BuildHuffmanTree(alphabet);

std::unordered\_map<std::string, std::string> huffmanCode;

EncodingSymbolsFromTree(huffmanTree, "", huffmanCode);

// Записываем в переменную содержимое текстбокса с закодированной последовательностью

auto encodingText = msclr::interop::marshal\_as<std::string>(textBox2->Text);

std::string decodingText;

int index = -1;

// Декодируем закодированную последовательность, записываем декодированную последовательность в decodingText

while (index < (int)encodingText.size() - 2)

DecodingCodesFromTree(huffmanTree, index, encodingText, decodingText);

// Записываем в текстбокс декодированную последовательность

textBox3->Text = gcnew System::String(decodingText.c\_str());

// Заполняем текстбокс символ - кодовое слово

for (auto elem : huffmanCode)

textBox4->AppendText("\r\n" + gcnew System::String(elem.first.c\_str()) + "\t" + gcnew System::String(elem.second.c\_str()));

}

// Событие при нажатии на кнопку "Сохранить текст в файл"

// Происходит запись содержимого текстбокса с декодированной последовательностью в файл "SavingDecoding.txt"

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

WriteTextToFile("SavingDecodingText.txt", msclr::interop::marshal\_as<std::string>(textBox3->Text));

}

catch(int item)

{

if(item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

}

Файл EncodingForm.h (только обработчики событий):

#pragma endregion

private: System::Void EncodingForm\_Load(System::Object^ sender, System::EventArgs^ e) {}

// Событие при нажатии на кнопку "Вставить текст из файла"

// Происходит запись в текстбокс с исходной последовательностью содержимого файла Original.txt

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

textBox2->Text = gcnew System::String(GetTextFromFile("OriginalText.txt").c\_str());

}

catch (int item)

{

if (item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

}

// Событие при нажатии на кнопку "Кодировать"

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

// Записываем в переменную содержимое текстбокса с исходной последовательностью

auto originalText = msclr::interop::marshal\_as<std::string>(textBox2->Text);

std::unordered\_map<std::string, double> alphabet;

try

{

// Берем из файла символы алфавита и вероятности их появления, в зависимости от выбранного распределения

if (comboBox1->Text == L"Равномерное")

alphabet = GetAlphabet("Alphabet.txt");

else if (comboBox1->Text == L"P1(A)")

alphabet = GetAlphabet("AlphabetP1(A).txt");

else

alphabet = GetAlphabet("AlphabetP2(A).txt");

}

catch (int item)

{

if (item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

// Строим Хаффмановское дерево, а после определяем кодовые слова из дерева

auto huffmanTree = BuildHuffmanTree(alphabet);

std::unordered\_map<std::string, std::string> huffmanCode;

EncodingSymbolsFromTree(huffmanTree, "", huffmanCode);

// Заполняем текстбоксы: полученная закодированная последовательность, средняя длина кодового слова,

// избыточность и энтропия

auto averageLength = GetAverageLength(huffmanCode, alphabet);

auto entrophy = GetEntrophy(alphabet);

textBox3->Text = gcnew System::String(GetEncodedText(huffmanCode, originalText).c\_str());

textBox5->Text = gcnew System::String((std::to\_string(averageLength).c\_str()));

textBox6->Text = gcnew System::String((std::to\_string(GetRedundancy(entrophy, averageLength))).c\_str());

textBox7->Text = gcnew System::String((std::to\_string(entrophy).c\_str()));

// Заполняем текстбокс с неравенством Крафта

if (CheckingCraftInequality(huffmanCode))

textBox8->Text = L"Выполняется";

else

textBox8->Text = L"Не выполняется";

textBox4->Text = L"";

// Заполняем текстбокс символ - кодовое слово

for(auto elem : huffmanCode)

textBox4->AppendText("\r\n" + gcnew System::String(elem.first.c\_str()) + "\t" + gcnew System::String(elem.second.c\_str()));

}

// Событие при нажатии на кнопку "Сохранить текст в файл"

// Происходит запись содержимого текстбокса с закодированной последовательностью в файл "SavingEncoding.txt"

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

WriteTextToFile("SavingEncodingText.txt", msclr::interop::marshal\_as<std::string>(textBox3->Text));

}

catch (int item)

{

if (item == 0)

System::Windows::Forms::MessageBox::Show("Файл не может быть открыт!", "Ошибка");

}

}

Файл MainForm.h (только обработчики событий):

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

EncodingForm^ encodingForm = gcnew EncodingForm();

encodingForm->ShowDialog();

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

DecodingForm^ decodingForm = gcnew DecodingForm();

decodingForm->ShowDialog();

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e)

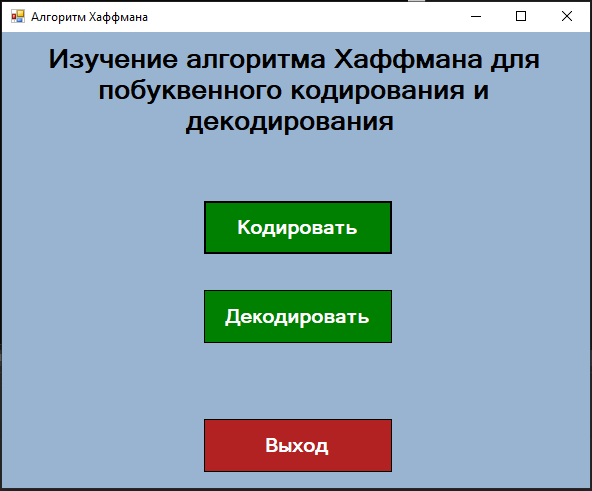
{

System::Windows::Forms::Application::Exit();

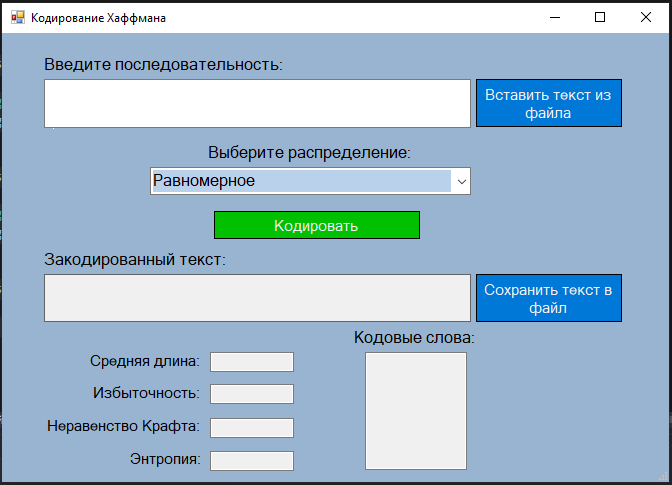
}

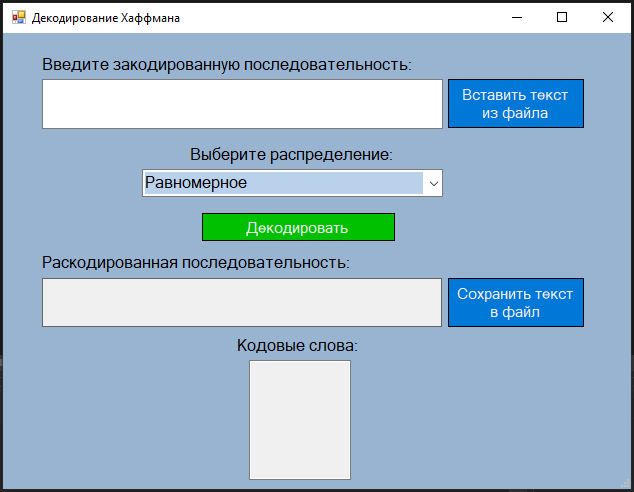
1. **Тесты**

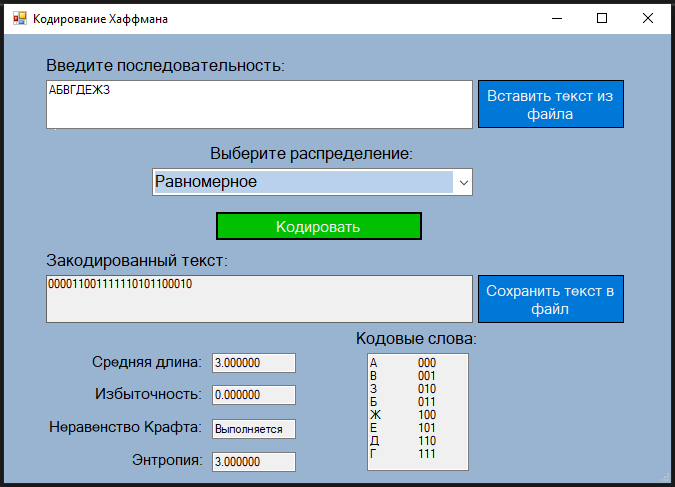
Экран “Главное меню”:



Экран “Кодирование Хаффмана”:



Экран “Декодирование Хаффмана”:

Пример кодирования:

Пример декодирования:

