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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Колованов Р.А.

Группа: 9381

Тема работы:
Вариант 1. Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона - демонстрация.

Исходные данные:
Текст, который требуется закодировать, или закодированный текст, который требуется раскодировать.

Содержание пояснительной записки:
«Содержание», «Введение», «Формальная постановка задачи», «Описание алгоритма», «Описание структур данных и функций», «Описание интерфейса пользователя», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.12.2020

Дата защиты реферата: 16.12.2020

Студент _____ Колованов Р.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

Задача курсовой работы состоит в разработке программы для кодирования и декодирования текстовых файлов алгоритмами Шеннона-Фано и Хаффмана. В качестве интерфейса для пользователя было решено реализовать консольный интерфейс.

Курсовая работа состоит из пояснительной записки и исходного кода разработанной программы.

В ходе работы была разработана программа с консольным интерфейсом для кодирования и декодирования текстовых файлов алгоритмами Шеннона-Фано и Хаффмана. Для написания программы использовался язык программирования C++.

SUMMARY

The task of the course work is to develop a program for encoding and decoding text files using Shannon-Fano and Huffman algorithms. It was decided to implement a console interface as an interface for the user.

Course work consists of an explanatory note and the source code of the developed program.

In the course of work, a program with a console interface was developed for encoding and decoding text files using the Shannon-Fano and Huffman algorithms. To write the program, the programming language C ++ was used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы.

Разработка программы для кодирования и декодирования текстовых файлов алгоритмами Шеннона-Фано и Хаффмана.

Задачи.

- Изучение языка программирования C++;
- Изучение алгоритмов кодирования и декодирования Шеннона-Фано и Хаффмана;
- Изучение структур данных, которые используются при реализации алгоритмов кодирования и декодирования;
- Написание исходного кода программы;
- Сборка программы;
- Тестирование программы.

Основные теоретические положения.

Подобно алгоритму Хаффмана, алгоритм Шеннона-Фано использует избыточность сообщения, заключённую в неоднородном распределении частот символов его алфавита, то есть заменяет коды более частых символов короткими двоичными последовательностями, а коды более редких символов – более длинными двоичными последовательностями. Коды Шеннона-Фано – беспрефиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов. Кодовые слова сопоставляются символам алфавита при помощи бинарного дерева: путь от корня дерева до листа, содержащего некоторый символ алфавита, является его кодом – если переходим в левое поддерево, то в кодовое слово добавляется 0, если же в правое – то 1.

ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

На вход подаётся файл с закодированным или незакодированным содержимым. Требуется написать программу, которая способна раскодировать или закодировать содержимое файла алгоритмом Шеннона-Фано или Хаффмана. По мимо этого, требуется продемонстрировать работу алгоритмов и структур данных в программе.

1. ОПИСАНИЕ АЛГОРИТМА

1.1. Кодирование алгоритмом Шеннона-Фано

Для этого был реализован класс *ShannonFanoEncoder*. Для начала происходит подсчет количества вхождений каждой буквы алфавита в кодируемый текст. Для этого создается вектор пар (*std::vector<std::pair<char, size_t>>*), который будет хранить буквы и количество их вхождений. Далее алгоритм проходит по каждой букве текста: если буква еще не была встречена (еще не занесена в вектор), то происходит добавление пары с этой буквой и 1 в вектор. Иначе у элемента вектора, который соответствует этой букве, инкрементируется количество вхождений. В конце вектор сортируется по убыванию значения вхождения с учетом лексикографического порядка символов алфавита. После подсчета вхождений происходит построение дерева кодирования Шеннона-Фано и сопоставление каждому символу алфавита своего кода. Для начала в векторе частот вхождений символов алфавита происходит поиск такого индекса k, для которого абсолютное значение разности сумм вхождений символов, стоящих слева от индекса k (включая этот индекс) и справа от индекса k, минимально. Исходный вектор вхождений делится индексом k на два вектора, которые после рекурсивно передаются в этот же метод, из которого будут возвращены созданные для этих вхождений бинарные деревья, и присваивает их значение левому и правому поддеревьям с учетом того факта, что сумма вхождений символов у правого поддерева должна быть больше или равна сумме вхождений у левого поддерева. В процессе деления векторов вхождений символов алфавита алгоритм вскоре дойдет до ситуации, когда в векторе останется один элемент. В этом случае узлу дерева присваивается значение оставшегося символа, в словарь кодов символов добавляется оставшийся символ и соответствующий ему код, и происходит выход из функции.

1.2. Кодирование алгоритмом Хаффмана

Для этого был реализован класс *HuffmanEncoder*. Для начала происходит подсчет количества вхождений каждой буквы алфавита в кодируемый текст. Для этого создается вектор пар (*std::vector<std::pair<char, size_t>>*), который будет хранить буквы и количество их вхождений. Далее алгоритм проходит по каждой букве текста: если буква еще не была встречена (еще не занесена в вектор), то происходит добавление пары с этой буквой и 1 в вектор. Иначе у элемента вектора, который соответствует этой букве, инкрементируется количество вхождений. В конце вектор сортируется по убыванию значения вхождения с учетом лексикографического порядка символов алфавита. После подсчета вхождений происходит построение дерева кодирования Хаффмана. Для начала для каждого элемента в векторе частот вхождений символов алфавита создается узел дерева со значением, равным символу, и весу, равному частоте символа. Все эти узлы заносятся в отдельный список. Далее в цикле выполняются следующие действия: для начала выбираем два узла дерева с наименьшими весами, после чего создается их родитель с весом, равным их суммарному весу. Родитель добавляется в список узлов дерева, а два его потомка удаляются из этого списка. Действия повторяются до тех пор, пока в списке не останется один элемент – итоговое дерево кодирования Хаффмана. После построения дерева происходит сопоставление каждому символу алфавита текста своего кода. Для этого происходит обход бинарного дерева кодирования: при переходе в левое поддерево узла к пути добавляется 0, а в правое – 1. Когда алгоритм дойдет до листа дерева (узла, содержащего закодированный символ), то в словарь кодов символов алфавита будет добавлен найденный закодированный символ и его код – путь к данному узлу дерева.

1.2. Декодирование

Для этого был реализован класс *Decoder*. Для начала происходит получение бинарного дерева кодирования и последовательности бит

закодированного текста. В начале работы алгоритм находится в корне бинарного дерева кодирования. Далее алгоритм проходит последовательность бит и при получении очередного бита выполняет следующие действия: если бит равен 0, то происходит переход в левое поддерево текущего дерева, иначе – в правое. Далее проверяется, достигнут ли лист дерева: если да – то в текущем листе записан очередной символ текста, который добавляется в раскодированный текст, после чего происходит переход обратно в корень дерева кодирования, если же нет – то спуск по бинарному дереву происходит дальше, пока не будет встречен лист дерева.

2. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

Для решения поставленной задачи были написаны классы *ShannonFanoEncoder, HuffmanEncoder* и *Decoder*, которые позволяют кодировать и декодировать текст алгоритмами Шеннона-Фано и Хаффмана. Для хранения бинарного дерева кодирования был реализован класс *BinaryTree*. Для вывода основной и промежуточной информации на экран и в файл был использован класс *Logger*. Для реализации вспомогательных функций был написан класс *Utils*, где вспомогательные Помимо этого, был реализован консольный интерфейс для удобной работы с программой.

В программе используются следующие синонимы для типов данных из C++ и библиотеки STL:

- *Character – char*, символ текста.
- *BitSequence – std::vector<bool>*, последовательность бит, где 0 – это false, а 1 – это true.
- *CharacterCodes – std::map<Character, BitSequence>*, словарь, в котором ключи – это символы, а значения – это коды, которые соответствуют ключу.
- *CharacterFrequency – std::pair<Character, size_t>*, пара значений: символ и количество его вхождений в текст.
- *CharacterFrequencies – std::vector<CharacterFrequency>*, вектор количества вхождений символов в текст.

Результаты тестирования см. в приложении А.

Разработанный программный код см. в приложении Б.

2.1. Перечисление MessageType

Хранит тип сообщения для логгера. В зависимости от типа сообщения меняется поток вывода, а некоторых случаях вывод может не производится. Существуют следующие значения перечисления:

- *MessageType::Common* – обычное сообщение, выводится в поток *stdout*.
- *MessageType::Error* – сообщение об ошибке, выводится в поток *stderr*.
- *MessageType::Debug* – отладочное сообщение (промежуточные данные), выводятся в поток *stdout* только в том случае, если у логгера включен режим *DebugMode*.

2.2. Перечисление Color

Хранит тип цвета для текста или заднего фона консоли. Существуют следующие значения перечисления:

- *Color::Black* – черный цвет;
- *Color::Blue* – синий цвет;
- *Color::Green* – зеленый цвет;
- *Color::Cyan* – сине-зеленый цвет;
- *Color::Red* – красный цвет;
- *Color::Magenta* – пурпурный цвет;
- *Color::Brown* – коричневый цвет;
- *Color::LightGray* – светло-серый цвет;
- *Color::DarkGray* – темно-серый цвет;
- *Color::LightBlue* – светло-синий цвет;
- *Color::LightGreen* – светло-зеленый цвет;
- *Color::LightCyan* – светло-сине-зеленый цвет;
- *Color::LightRed* – светло-красный цвет;
- *Color::LightMagenta* – светло-пурпурный цвет;
- *Color::Yellow* – желтый цвет;
- *Color::White* – белый цвет;

2.3. Класс Logger

Класс предоставляет функционал для вывода сообщений в консоль и файл из любой точки программы. Реализован с использованием паттерна *Singleton*. Поля и методы класса приведены в таблицах 1 и 2.

Таблица 1 - Поля класса *Logger*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*int indent_size_*	Хранит размер отступа в пробелах.	*4*
private	*bool debug_mode_*	Хранит информацию о том, включен ли режим отладки. При выключенном режиме отладки сообщения типа MessageType::Debug будут игнорироваться.	*false*
private	*bool file_output_*	Хранит информацию о том, нужно ли выводить сообщения в файл.	*false*
private	*std::ofstream file_*	Поток вывода данных в файл.	-

Таблица 2 - Методы класса *Logger*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*Logger&*	*getInstance()*
public	*void*	*log(const std::string& message, MessageType type = MessageType::Common, int indents = 0)*
public	*void*	*setConsoleColor(Color text_color, Color background_color)*
public	*void*	*setOutputFile(const std::string& filePath)*
public	*void*	*setDebugMode(bool value)*
public	*bool*	*getDebugMode()*
public	*std::string*	*getCurrentDataTime()*

Метод Logger::getInstance.

Ничего не принимает. Создает статическую переменную объекта класса *Logger* (создается только один раз — при первом вызове данного метода). Возвращает ссылку на единственный объект класса *Logger*.

Метод Logger::log.

Принимает на вход три аргумента: *message* — сообщение, *type* — тип сообщения и *indents* — количество отступов. Печатает сообщение с отступом в консоль и, если установлен флаг *file_output_*, в файл. В зависимости от типа сообщения выбирается поток вывода. Если включен режим отладки и тип сообщения — MessageType::*Debug*, то сообщение выведено не будет. Ничего не возвращает.

Метод Logger::setConsoleColor.

Принимает на вход два аргумента: *text_color* — цвет текста консоли и *background_color* — цвет заднего фона текста. Меняет цвета текста и заднего фона текста консоли. Ничего не возвращает.

Метод Logger::setOutputFile.

Принимает на вход *filepath* — путь к файлу для записи сообщений. Открывает поток вывода сообщений в файл и присваивает полю *file_output_* значение *true*. Ничего не возвращает.

Метод Logger::setDebugMode.

Принимает на вход *value* — новое значение флага режима отладки. Устанавливает полю *debug_mode_* значение *value*. Ничего не возвращает.

Метод Logger::getDebugMode.

Ничего не принимает. Возвращает значение поля *debug_mode_*.

Метод Logger::getCurrentDataTime.

Ничего не принимает. Возвращает текущие дату и время в виде следующей строки: <*день*>-<*месяц*>-<*год*>_<*часы*>-<*минуты*>-<*секунды*>. Используется для генерации имени файла с логами.

2.4. Класс Exception

Используется в качестве класса-исключения. Объект данного класса выбрасывается в качестве исключения оператором throw, после чего отлавливается блоком try-catch. Поля и методы класса приведены в таблице 3 и 4.

Таблица 3 - Поля класса *Exception*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*const std::string message_*	Хранит сообщение об ошибке.	-

Таблица 4 - Методы класса *Exception*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*Exception(const std::string& message)*
public	*const std::string&*	*getMessage() const*

Метод Exception::Exception.

Принимает на вход *message* – сообщение об ошибке. Устанавливает полю *message_* значение *message*. Ничего не возвращает.

Метод Exception::getMessage.

Ничего не принимает. Возвращает значение поля *message_*.

2.5. Класс BinaryTree

Класс бинарного дерева. Для реализации класса используется шаблон, который определяет тип элементов дерева. Предоставляет интерфейс для создания бинарного дерева по скобочной записи и работы с бинарным деревом. Связь элементов бинарного дерева реализована при помощи указателей. В данной работе используется для хранения дерева кодирования. Поля и методы класса приведены в таблице 5 и 6.

Таблица 5 - Поля класса *BinaryTree*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*T element*	Хранит значение корня дерева.	-
private	*size_t weight_*	Хранит вес корня дерева.	*0*
private	*BinaryTree* right_*	Хранит адрес правого поддерева.	*nullptr*
private	*BinaryTree* left_*	Хранит адрес левого поддерева.	*nullptr*

Таблица 6 - Методы класса *BinaryTree*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*BinaryTree()*
public	-	*BinaryTree(const T& element, size_t weight = 0)*
public	-	*BinaryTree(const std::string& expression)*
public	-	*~BinaryTree()*
public	*bool*	*createFromString(const char*& character)*
public	*void*	*setElement(const T& value)*
public	*T*	*getElement() const*
public	*size_t*	*getWeight() const*
public	*void*	*setWeight(size_t value)*
public	*BinaryTree**	*getRightSubtree()*
public	*BinaryTree**	*getLeftSubtree()*
public	*const BinaryTree**	*getRightSubtree()const*
public	*const BinaryTree**	*getLeftSubtree() const*
public	*void*	*setRightSubtree(BinaryTree* subtree)*
public	*void*	*setLeftSubtree(BinaryTree* subtree)*
public	*bool*	*isLeaf() const*
public	*std::string*	*getString() const*

public	*std::string*	*getElementString() const*

Метод BinaryTree::BinaryTree.

Конструктор. Ничего не принимает. Создает пустое бинарное дерево.

Метод BinaryTree::BinaryTree.

Конструктор. Принимает на вход два аргумента: *element* – значение элемента корня дерева и *weight* – вес корня дерева. Создает пустое бинарное дерево, состоящее только из корня. Полю *element_* и *weight_* присваиваются значения *element* и *weight* соответственно.

Метод BinaryTree::BinaryTree.

Конструктор. Принимает на вход *expression* – строку, содержащую скобочную запись бинарного дерева. Создает бинарное дерево по скобочной записи при помощи метода *createFromString*.

Метод BinaryTree::~BinaryTree.

Деструктор. Является рекурсивным методом. Очищает выделенную под элементы бинарного дерева динамическую память.

Метод BinaryTree::createFromString.

Является рекурсивным методом. Принимает на вход *character* – ссылку на указатель начала строки, содержащую скобочную запись бинарного дерева. Создает бинарное дерево по заданной скобочной записи. Если бинарное дерево было успешно создано по скобочной записи, то возвращает *true*. Если скобочная запись оказалась некорректной, то возвращает *false*.

Метод BinaryTree::setElement.

Устанавливает элементу узла дерева новое значение. Принимает на вход *value* – новое значение. Ничего не возвращает.

Метод BinaryTree::getElement.

Ничего не принимает. Возвращает значение элемента узла дерева.

Метод BinaryTree::getWeight.

Ничего не принимает. Возвращает значение веса узла дерева.

Метод BinaryTree::setWeight.

Устанавливает весу узла дерева новое значение. Принимает на вход *value* – новое значение. Ничего не возвращает.

Метод BinaryTree::getRightSubtree.

Ничего не принимает. Возвращает правое поддерево узла дерева.

Метод BinaryTree::getLeftSubtree.

Ничего не принимает. Возвращает левое поддерево узла дерева.

Метод BinaryTree::setRightSubtree.

Устанавливает узлу дерева правое поддерево. Принимает на вход *subtree* – новое правое поддерево. Ничего не возвращает.

Метод BinaryTree::setLeftSubtree.

Устанавливает узлу дерева левое поддерево. Принимает на вход *subtree* – новое левое поддерево. Ничего не возвращает.

Метод BinaryTree::isLeaf.

Ничего не принимает. Если бинарное дерево является листом (*left = nullptr, right == nullptr*), то возвращает *true*. Иначе возвращает *false*.

Метод BinaryTree::getString.

Является рекурсивным методом. Ничего не принимает. Возвращает строку, в которой содержится скобочная запись бинарного дерева.

Метод BinaryTree::getElementString.

Является рекурсивным методом. Ничего не принимает. Возвращает строку, в которой содержится элементы узлов бинарного дерева в том порядке, в котором они размещены в скобочной записи бинарного дерева.

2.6. Класс Encoder

Базовый класс кодировщика. Объявляет интерфейс, наследуемый далее конкретными кодировщиками для его реализации. Поля и методы класса приведены в таблице 7 и 8.

Таблица 7 - Поля класса *Encoder*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
protected	*CharactersFrequency frequencies_*	Хранит частоту символов в обработанном тексте.	-
protected	*BinaryTree<Character>* tree_*	Хранит бинарное дерево кодирования для закодированного текста.	-
protected	*CharacterCodes codes_*	Хранит коды символов текста.	-

Таблица 8 - Методы класса *Encoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*~Encoder()*

public	*const BinaryTree<Character>**	*getCodingTree() const*
public	*const CharacterCodes &*	*getCharacterCodes() const*
public	*const CharacterFrequencies&*	*getCharacterFrequencies() const*
public	*const CharacterFrequencies&*	*calculateCharacterFrequencies(const std::string& text)*
public	*BitSequence*	*encodeText(const std::string& text) = 0*

Метод Encoder::~Encoder.

Деструктор. Ничего не принимает. Очищает выделенную под бинарное дерево кодирования динамическую память.

Метод Encoder::getCodingTree.

Ничего не принимает. Возвращает бинарное дерево кодирования для закодированного при помощи метода *encodeText* тексте.

Метод Encoder::getCharacterCodes.

Ничего не принимает. Возвращает коды символов закодированного при помощи метода *encodeText* тексте.

Метод Encoder::getCharacterFrequencies.

Ничего не принимает. Возвращает частоту символов в обработанном при помощи метода *calculateTextCharacterFrequencies* тексте.

Метод Encoder::calculateCharacterFrequencies.

Принимает на вход *text* – некоторый текст. Считает количество вхождений символов в текст (частоту встречи каждого символа текста) и возвращает его.

Метод Encoder::encodeText.

Чистый виртуальный метод. Реализуется классами наследниками. Принимает на вход *text* – текст для кодирования. Используется для кодирования текста *text* некоторым алгоритмом. Возвращает закодированный текст – последовательность битов.

2.7. Класс ShannonFanoEncoder

Класс кодировщика алгоритмом Шеннона-Фано. Используется для кодирования текста алгоритмом Шеннона-Фано. Является наследником класса *Encoder*. Методы класса приведены в таблице 9.

Таблица 9 - Методы класса *ShannonFanoEncoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
private	*BinaryTree<Character>**	*calculateCharactersTreeAndCodes(CharacterFrequencies& frequency, BitSequence& path)*
public	*BitSequence*	*encodeText(const std::string& text) override*

Метод ShannonFanoEncoder::calculateCharactersTreeAndCodes.

Является рекурсивным методом. Принимает на вход три аргумента: *frequency* – частоты символов текста, которые нужно распределить в поддеревья текущего узла дерева, *codes* – ссылка на вектор кодов символов, который будет заполняться кодами по ходу работы метода, и *path* – путь от корня до текущего узла дерева в виде битовой последовательности, где 0 – это переход в левое поддерево, а 1 – в правое. Распределяет символы из списка частот символов *frequency* при помощи алгоритма Шеннона-Фано по листьям создаваемого

бинарного дерева кодирования, а также сопоставляет каждому символу алфавита свой код. Возвращает созданное бинарное дерево.

Метод ShannonFanoEncoder::encodeText.

Реализация виртуального метода *encodeText*. Принимает на вход *text* – текст для кодирования. Используется для кодирования текста *text* алгоритмом Шеннона-Фано. Возвращает закодированный текст – последовательность битов.

2.8. Класс HuffmanEncoder

Класс кодировщика алгоритмом Хаффмана. Используется для кодирования текста алгоритмом Хаффмана. Является наследником класса *Encoder*. Методы класса приведены в таблице 10.

Таблица 10 - Методы класса *HuffmanEncoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
private	*void*	*calculateCharactersTree()*
private	*void*	*calculateCharacterCodes(const BinaryTree<Character>* tree, BitSequence& path)*
public	*BitSequence*	*encodeText(const std::string& text) override*

Метод HuffmanEncoder::calculateCharactersTree.

Ничего не принимает. Распределяет символы из списка частот символов *frequency_* при помощи алгоритма Хаффмана по листьям создаваемого бинарного дерева кодирования *tree_*. Ничего не возвращает.

Метод HuffmanEncoder::calculateCharactersCodes.

Является рекурсивным методом. Принимает на вход два аргумента: *tree* – указатель на текущий узел дерева и *path* – путь от корня до текущего узла дерева в виде битовой последовательности, где 0 – это переход в левое поддерево, а 1 –

в правое. Обходит бинарное дерево кодирования *tree_* и сопоставляет каждому символу алфавита свой код. Ничего не возвращает.

Метод HuffmanEncoder::encodeText.

Реализация виртуального метода *encodeText*. Принимает на вход *text* – текст для кодирования. Используется для кодирования текста *text* алгоритмом Хаффмана. Возвращает закодированный текст – последовательность битов.

2.9. Класс Decoder

Класс декодировщика. Используется для декодирования текста, закодированным при помощи алгоритмов Шеннона-Фано и Хаффмана. Поля и методы класса приведены в таблице 11 и 12.

Таблица 11 - Поля класса *Decoder*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
protected	*BinaryTree<Character>* tree_*	Хранит дерево кодирования для закодированного текста.	*nullptr*

Таблица 12 - Методы класса *Decoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*~Decoder()*
public	*const BinaryTree<Character>**	*getCodingTree() const*
public	*bool*	*setCodingTree(const std::string& expression)*
public	*std::string*	*decodeText(BitSequence& sequence)*

Метод Decoder::~Decoder.

Деструктор. Очищает выделенную под бинарное дерево кодирования динамическую память.

Метод Decoder::getCodingTree.

Ничего не принимает. Возвращает дерево кодирования.

Метод Decoder::setCodingTree.

Принимает на вход *expression* – строку, содержащую скобочную запись бинарного дерева кодирования. Создает бинарное дерево кодирования по скобочной записи при помощи метода *createFromString*. Возвращает *true*, если дерево было успешно создано, иначе *false* – в случае, если скобочная запись бинарного дерева некорректна.

Метод Decoder::decodeText.

Принимает на вход *sequence* – битовая последовательность закодированного текста. Используется для декодирования последовательности бит *sequence* в исходный текст, используя бинарное дерево кодирования *tree_*. Возвращает декодированный текст.

2.10. Класс Utils

Класс вспомогательных функций. Методы данного класса являются статическими, что позволяет вызывать их без создания объекта класса. Методы класса приведены в таблице 13.

Таблица 13 - Методы класса *Utils*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*std::string*	*bitSequenceToString(const BitSequence& sequence)*
public	*void*	*clearInput()*

Метод Utils::bitSequenceToString.

Принимает на вход *sequence* – последовательность бит. Конвертирует последовательность бит *sequence* в строку, и возвращает ее.

Метод Utils::clearInput.

Очищает поток ввода *stdin* до первого символа перевода строки. Требуется для удаления некорректных символов, которые не были считаны с потока ввода. Например, когда программа ожидает получить число, а пользователь вводит букву, которая в итоге остается лежать в потоке ввода. Ничего не принимает; ничего не возвращает.

2.11. Функция main

Для начала объявляются следующе переменные:

- *is_loop_enabled* — хранит информацию о том, надо ли продолжать выполнение основного цикла программы;
- *is_debug_mode* — хранит информацию о том, надо ли выводить промежуточные данные;
- *logger* — хранит объект класса *Logger*.

Далее производится настройка русского языка для консоли. После у логгера *logger* вызывается метод *setOutputFile* для установки файла вывода сообщений, помимо этого устанавливается режим вывода промежуточных данных.

Далее происходит вход в основной цикл программы. Для начала считывается выбранное пользователем действие (цифра от 1 до 6). Если пользователь выбрал действие 1-2 (закодировать текст), то происходит считывание выбранного пользователем алгоритма кодирования. Если пользователь выбрал кодирований текста из файла, то также происходит

считывание пути до файла. Для выбранного действия выполняется соответствующее действие и выводится результат работы действия.

3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Пользователь выбирает действие (вводит цифру от 1 до 6). В зависимости от выбранного действия выполняется:

- Кодирование текста, введенного с консоли, при помощи одного из алгоритмов: Шеннона-Фано и Хаффмана. (1)
    - Алгоритм Шеннона-Фано. (1)
    - Алгоритм Хаффмана. (2)
- Кодирование текста, введенного с файла, при помощи одного из алгоритмов: Шеннона-Фано и Хаффмана. (2)
    - Алгоритм Шеннона-Фано. (1)
    - Алгоритм Хаффмана. (2)
- Декодирование текста, введенного с файла. (3)
- Включение вывода промежуточных данных. (4)
- Выключение вывода промежуточных данных. (5)
- Выход из программы. (6)

ЗАКЛЮЧЕНИЕ

В ходе работы над поставленным заданием был изучен алгоритм кодирования и декодирования Шеннона-Фано и Хаффмана, а также была разработана программа с консольным интерфейсом, которая способна закодировать или раскодировать содержимое файла алгоритмом Шеннона-Фано или Хаффмана. Программа была успешно протестирована на работоспособность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм Шеннона-Фано - Википедия // Википедия. URL: https://ru.wikipedia.org/wiki/Алгоритм_Шеннона_—_Фано (дата обращения: 05.12.2020).

2. Код Хаффмана - Википедия // Википедия. URL: https://ru.wikipedia.org/wiki/Код_Хаффмана (дата обращения: 05.12.2020).

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

Таблица А.1 - Примеры тестовых случаев на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	Выберите одно из вышеперечисленных действий: -1	Выбрано некорректное действие.
2.	Выберите одно из вышеперечисленных действий: 9	Выбрано некорректное действие.
3.	Выберите одно из вышеперечисленных действий: 2 Выбранное действие: 2 Выберите одно из вышеперечисленных алгоритмов: 4	Выбран некорректный алгоритм. Выберите алгоритм снова:
4.	Выберите одно из вышеперечисленных действий: 2 Выбранное действие: 2 Выберите одно из вышеперечисленных алгоритмов: 1 Выбранный алгоритм: 1 Введите путь к файлу: not_exist.txt Введенный путь к файлу: not_exist.txt	При открытии файла произошла ошибка: not_exist.txt
5.	Выберите одно из вышеперечисленных действий: 3 Выбранное действие: 3 Считывание дерева кодирования из файла 'encoded_text.txt'...	Скобочная запись дерева кодирования некорректна: ('\0'('\0'('\0'('\D FFD S Fsdf

Таблица А.2 - Примеры тестовых случаев на корректных данных

№ п/п	Входные данные	Выходные данные
6.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 1 Выбранный алгоритм: 1 Введите текст: Hello, мир! Введенный текст: Hello, мир!	[Дерево кодирования] ('\0'('\0('l'//)('\0('и'//)('\0('м'//)('р'//))))('\0'('\0(' ' '//)('\0('!'//)(','//)))('\0('H'//)('\0('e'//)('o'//))))) [Закодированный текст] 11011100000111110111000110010011110 10

7.	Выберите одно из вышеперечисленных действий: 3 Выбранное действие: 3 [Дерево кодирования] ('\0'('\0'('l'//)('\0'('и'//)('\0'('м'//)('р'//))))('\0' ('\0'(' '//)('\0'('!'//)(','//)))('\0'('Н'//)('\0'('е'//)('о'//))))) [Закодированный текст] 11011100000111110111000110010011110 10	[Декодированный текст] Hello, мир!
8.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 2 Выбранный алгоритм: 2 Введите текст: Привет, world! Введенный текст: Привет, world!	[Дерево кодирования] ('\0'('\0'('\0'('\0'('е'//)('и'//))('\0'('П'//)('в'//)))('\0' ('\0'(' '//)('!'//))('\0'('р'//)('т'//))))('\0'('\0'('\0'('l'//)('o'//))('\0',''//)('d'//)))('\0'('r'//)('w'//)))) [Закодированный текст] 00100110000100110000011111010010011110 01110100010110101
9.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 1 Выбранный алгоритм: 1 Введите текст: Введенный текст:	[Дерево кодирования] ('\0'//) [Закодированный текст]
10.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 2 Выбранный алгоритм: 2 Введите текст: Введенный текст:	[Дерево кодирования] ('\0'//) [Закодированный текст]
11.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 1 Выбранный алгоритм: 1 Введите текст: ааааа Введенный текст: ааааа	[Дерево кодирования] ('\0'/('a'//)) [Закодированный текст] 11111

12.	Выберите одно из вышеперечисленных действий: 1 Выбранное действие: 1 Выберите одно из вышеперечисленных алгоритмов: 2 Выбранный алгоритм: 2 Введите текст: bbbbbb Введенный текст: bbbbbb	[Дерево кодирования] ('\0'('b'//)) [Закодированный текст] 111111
13.	Выберите одно из вышеперечисленных действий: 2 Выбранное действие: 2 Выберите одно из вышеперечисленных алгоритмов: 2 Выбранный алгоритм: 2 Введите путь к файлу: input_text.txt Введенный путь к файлу: input_text.txt [Текст из файла] Facebook will launch the Libra digital currency in January 2021, according to the Financial Times FT sources. The currency will be issued in a "cut format", the newspaper specifies. Whereas Libra was originally supposed to be backed by a basket of traditional currencies, now it will be a currency backed only by the US dollar at a 1: 1 rate. The exact timing of the Libra launch depends on the approval of the Swiss Financial Markets Authority the Libra Association is headquartered in Geneva.	[Дерево кодирования] ('\0'('\0'('\0'('\0'('e'//)('a'//))(' '//))('\0'('\0'('\0'('s'//)('l'//))('i'//))('\0'('\0'('\0'('y'//)('\0'('k'//)('.'//)))('\0'('\0'('\0'('\0'('U'//)('W'//))('\ 0'('J'//)('M'//)))('\0'('v'//)('\0'('q'//)('x'//))))('\0'(' T'//)('g'//))))('\0'('\0'('p'//)('\0'('\0'('A'//)('S'//))('\ 0'('"'//)('2'//))))('h'//)))))('\0'('\0'('\0'('\0'('\0'('\0'('F'//)('L'//))('w'//))('b'//))('r'//))('\0'('t'//)('n'//)))(' \0'('\0'('\0'('u'//)('d'//))('c'//))('\0'('\0'('\0'('\0'('m'/ /)('\0'('\0':'//)('G'//))('0'//)))('\0',''//)('1'//)))('\0' (' '//)('f'//)))('o'//))))) [Закодированный текст] 1000000000111010000100011111111101100 1000110000101010100101001001010010000 1110001011110101111001101001111000000 1100000101011000110010001001110010101 0110111010110100001010010011101110001 0011001000010111101011000001010110110 0101101001000011011110000001100101100 0001011101111110001101110111111001111 1 0010001000111011101111110011100101011 0110110111001101011110011010011110000 0011000000010110110001101111010101000 1010010010110110010111100000000010000 0110000000110110001010001111110001001 1101000001000011001111101011101001101 1001111000000111011100010011001000010 1111010110000011000010101010010100100 1100010000001010101000010001100000001 1001001010110110010001001011101101101 1100010100011110111111100111100000001 1010011101101110010001101001111000000 1101100001000010100001110000010111000 0001001001010000111000000110101011110 1101010000010000110011001011010001011

		1100001001000000010100000110000010101 1000110010001001100001000101000001111 1100101010110111010110110001010010100 1011000001010001100001110001110011110 1000000011001001101011110011000100000 0110001000111010110010000011001001100 0101100000100010011000100010100001100 1000001010001111111101100110101001000 1110010101101001011111101100010100100 1110111000100110010000101111010101000 0010001110010001101111111000010010101 1010001100001010101001010010011000100 0000100010011101110001001100100001011 1101011000001100010001110101100100000 1100100111111011010010110000011000101 1000001101001111000000101101000001110 1010011100111110100101001000110010010 0011010001000100111100111110001000011 1100110011001000110100000011001111101 0111010011011001111000000100000110101 1100011101101000110100101111000001011 0110110111001111111101100110100111000 0001100000101011000110010001001010010 0011100010111101011110011100100000111 0000001011110010100000111111011001101 0011110000001000101110001110010011111 0110101000010100100111111110110011010 0111100000010111010110000101010100001 0000011000000001011011000110111101010 1 0001010010010110100110001100101100100 0001010010000010111010011000101001111 1111100101011010011000001101001111000 0001100000101011000110010001001011101 0001000010001111110101010001101001011 1111011001010101000001011110000000111 0010110101101100000011001101000001001 0000110010010101101100111100010100001 0110000011010100001011001111101 0
14.	Выберите одно из вышеперечисленных действий: 4 Выбранное действие: 4	Вывод промежуточных данных включен.
15.	Выберите одно из вышеперечисленных действий: 5 Выбранное действие: 5	Вывод промежуточных данных выключен.

```
Доступные действия:

  1) Закодировать текст - Ввод с консоли.
  2) Закодировать текст - Ввод с файла.
  3) Раскодировать текст - Ввод с файла.
  4) Включить вывод промежуточных данных (уже включен).
  5) Отключить вывод промежуточных данных.
  6) Выйти из программы.

Выберите одно из вышеперечисленных действий: 1
Выбранное действие: 1

Доступные алгоритмы кодирования:

  1) Алгоритм Шеннона-Фано.
  2) Алгоритм Хаффмана.

Выберите одно из вышеперечисленных алгоритмов: 1
Выбранный алгоритм: 1

Введите текст: Привет, World!
Введенный текст: Привет, World!

Кодирование текста...

[Шаг 1] Считаем частоту вхождений символов текста и сортируем их по убыванию количества вхождений с учетом лексиграфического порядка.

Частота вхождений символов в текст: П(1) в(1) е(1) и(1) р(1) т(1)  (1) !(1) ,(1) W(1) d(1) l(1) o(1) r(1)

[Шаг 2] Построение дерева кодирования Шеннона-Фано и сопоставление кода каждому символу алфавита текста.

Частота символов для текущего узла дерева (путь до узла - ): Пвеирт !,Wdlor(14)
Разделяем список частот. Индекс разделяющего элемента: 6.
Частоты символов для левого поддерева: Пвеирт (7)
Частоты символов для правого поддерева: !,Wdlor(7)
Создаем левое поддерево для текущего узла дерева (путь до узла - 0):
    Частота символов для текущего узла дерева (путь до узла - 0): Пвеирт (7)
    Разделяем список частот. Индекс разделяющего элемента: 2.
    Частоты символов для левого поддерева: Пве(3)
    Частоты символов для правого поддерева: ирт (4)
    Создаем левое поддерево для текущего узла дерева (путь до узла - 00):
        Частота символов для текущего узла дерева (путь до узла - 00): Пве(3)
        Разделяем список частот. Индекс разделяющего элемента: 0.
        Частоты символов для левого поддерева: П(1)
        Частоты символов для правого поддерева: ве(2)
        Создаем левое поддерево для текущего узла дерева (путь до узла - 000):
            Частота символов для текущего узла дерева (путь до узла - 000): П(1)
            Помещаем символ 'П' в текущий узел дерева (путь до узла - 000) и задаем в качестве кода текущего символа путь до текущего узла дерева.
        Создаем правое поддерево для текущего узла дерева (путь до узла - 001):
            Частота символов для текущего узла дерева (путь до узла - 001): ве(2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: в(1)
            Частоты символов для правого поддерева: е(1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 0010):
                Частота символов для текущего узла дерева (путь до узла - 0010): в(1)
                Помещаем символ 'в' в текущий узел дерева (путь до узла - 0010) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 0011):
```

```
            Создаем левое поддерево для текущего узла дерева (путь до узла - 0010):
                Частота символов для текущего узла дерева (путь до узла - 0010): в(1)
                Помещаем символ 'в' в текущий узел дерева (путь до узла - 0010) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 0011):
                Частота символов для текущего узла дерева (путь до узла - 0011): е(1)
                Помещаем символ 'е' в текущий узел дерева (путь до узла - 0011) и задаем в качестве кода текущего символа путь до текущего узла дерева.
    Создаем правое поддерево для текущего узла дерева (путь до узла - 01):
        Частота символов для текущего узла дерева (путь до узла - 01): ирт (4)
        Разделяем список частот. Индекс разделяющего элемента: 1.
        Частоты символов для левого поддерева: ир(2)
        Частоты символов для правого поддерева: т (2)
        Создаем левое поддерево для текущего узла дерева (путь до узла - 010):
            Частота символов для текущего узла дерева (путь до узла - 010): ир(2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: и(1)
            Частоты символов для правого поддерева: р(1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 0100):
                Частота символов для текущего узла дерева (путь до узла - 0100): и(1)
                Помещаем символ 'и' в текущий узел дерева (путь до узла - 0100) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 0101):
                Частота символов для текущего узла дерева (путь до узла - 0101): р(1)
                Помещаем символ 'р' в текущий узел дерева (путь до узла - 0101) и задаем в качестве кода текущего символа путь до текущего узла дерева.
        Создаем правое поддерево для текущего узла дерева (путь до узла - 011):
            Частота символов для текущего узла дерева (путь до узла - 011): т (2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: т(1)
            Частоты символов для правого поддерева:  (1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 0110):
                Частота символов для текущего узла дерева (путь до узла - 0110): т(1)
                Помещаем символ 'т' в текущий узел дерева (путь до узла - 0110) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 0111):
                Частота символов для текущего узла дерева (путь до узла - 0111):  (1)
                Помещаем символ ' ' в текущий узел дерева (путь до узла - 0111) и задаем в качестве кода текущего символа путь до текущего узла дерева.
Создаем правое поддерево для текущего узла дерева (путь до узла - 1):
    Частота символов для текущего узла дерева (путь до узла - 1): !,Wdlor(7)
    Разделяем список частот. Индекс разделяющего элемента: 2.
    Частоты символов для левого поддерева: !,W(3)
    Частоты символов для правого поддерева: dlor(4)
    Создаем левое поддерево для текущего узла дерева (путь до узла - 10):
        Частота символов для текущего узла дерева (путь до узла - 10): !,W(3)
        Разделяем список частот. Индекс разделяющего элемента: 0.
        Частоты символов для левого поддерева: !(1)
        Частоты символов для правого поддерева: ,W(2)
        Создаем левое поддерево для текущего узла дерева (путь до узла - 100):
            Частота символов для текущего узла дерева (путь до узла - 100): !(1)
            Помещаем символ '!' в текущий узел дерева (путь до узла - 100) и задаем в качестве кода текущего символа путь до текущего узла дерева.
        Создаем правое поддерево для текущего узла дерева (путь до узла - 101):
            Частота символов для текущего узла дерева (путь до узла - 101): ,W(2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: ,(1)
            Частоты символов для правого поддерева: W(1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 1010):
                Частота символов для текущего узла дерева (путь до узла - 1010): ,(1)
                Помещаем символ ',' в текущий узел дерева (путь до узла - 1010) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 1011):
                Частота символов для текущего узла дерева (путь до узла - 1011): W(1)
                Помещаем символ 'W' в текущий узел дерева (путь до узла - 1011) и задаем в качестве кода текущего символа путь до текущего узла дерева.
    Создаем правое поддерево для текущего узла дерева (путь до узла - 11):
        Частота символов для текущего узла дерева (путь до узла - 11): dlor(4)
        Разделяем список частот. Индекс разделяющего элемента: 1.
```

```
            Частота символов для текущего узла дерева (путь до узла - 1010): ,(1)
            Помещаем символ ',' в текущий узел дерева (путь до узла - 1010) и задаем в качестве кода текущего символа путь до текущего узла дерева.
        Создаем правое поддерево для текущего узла дерева (путь до узла - 1011):
            Частота символов для текущего узла дерева (путь до узла - 1011): W(1)
            Помещаем символ 'W' в текущий узел дерева (путь до узла - 1011) и задаем в качестве кода текущего символа путь до текущего узла дерева.
    Создаем правое поддерево для текущего узла дерева (путь до узла - 11):
        Частота символов для текущего узла дерева (путь до узла - 11): dlor(4)
        Разделяем список частот. Индекс разделяющего элемента: 1.
        Частоты символов для левого поддерева: dl(2)
        Частоты символов для правого поддерева: or(2)
        Создаем левое поддерево для текущего узла дерева (путь до узла - 110):
            Частота символов для текущего узла дерева (путь до узла - 110): dl(2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: d(1)
            Частоты символов для правого поддерева: l(1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 1100):
                Частота символов для текущего узла дерева (путь до узла - 1100): d(1)
                Помещаем символ 'd' в текущий узел дерева (путь до узла - 1100) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 1101):
                Частота символов для текущего узла дерева (путь до узла - 1101): l(1)
                Помещаем символ 'l' в текущий узел дерева (путь до узла - 1101) и задаем в качестве кода текущего символа путь до текущего узла дерева.
        Создаем правое поддерево для текущего узла дерева (путь до узла - 111):
            Частота символов для текущего узла дерева (путь до узла - 111): or(2)
            Разделяем список частот. Индекс разделяющего элемента: 0.
            Частоты символов для левого поддерева: o(1)
            Частоты символов для правого поддерева: r(1)
            Создаем левое поддерево для текущего узла дерева (путь до узла - 1110):
                Частота символов для текущего узла дерева (путь до узла - 1110): o(1)
                Помещаем символ 'o' в текущий узел дерева (путь до узла - 1110) и задаем в качестве кода текущего символа путь до текущего узла дерева.
            Создаем правое поддерево для текущего узла дерева (путь до узла - 1111):
                Частота символов для текущего узла дерева (путь до узла - 1111): r(1)
                Помещаем символ 'r' в текущий узел дерева (путь до узла - 1111) и задаем в качестве кода текущего символа путь до текущего узла дерева.

[Шаг 3] Кодирование каждого символа текста:

Символу 'П' соответствует код 000
Символу 'р' соответствует код 0101
Символу 'и' соответствует код 0100
Символу 'в' соответствует код 0010
Символу 'е' соответствует код 0011
Символу 'т' соответствует код 0110
Символу ',' соответствует код 1010
Символу ' ' соответствует код 0111
Символу 'W' соответствует код 1011
Символу 'o' соответствует код 1110
Символу 'r' соответствует код 1111
Символу 'l' соответствует код 1101
Символу 'd' соответствует код 1100
Символу '!' соответствует код 100

Кодирование текста завершено.

[Дерево кодирования] ('\0'('\0'('\0'('П'//)('\0'('в'//)('е'////)('\0'('\0'('и'//)('р'///)('\0'('т'//)(' '//////)('\0'('\0'('!'//)('\0',''//)('W'////)('\0'('\
0'('d'//)('l'//))('\0'('o'//)('r'//////)
[Закодированный текст] 0000101010000100011011010100111101111011111011100100

Закодированный текст записан в файл 'encoded_text.txt'.
```

```
Доступные действия:

  1) Закодировать текст - Ввод с консоли.
  2) Закодировать текст - Ввод с файла.
  3) Раскодировать текст - Ввод с файла.
  4) Включить вывод промежуточных данных (уже включен).
  5) Отключить вывод промежуточных данных.
  6) Выйти из программы.

Выберите одно из вышеперечисленных действий: 1
Выбранное действие: 1

Доступные алгоритмы кодирования:

  1) Алгоритм Шеннона-Фано.
  2) Алгоритм Хаффмана.

Выберите одно из вышеперечисленных алгоритмов: 2
Выбранный алгоритм: 2

Введите текст: Hello, мир!
Введенный текст: Hello, мир!

Кодирование текста...

[Шаг 1] Считаем частоту вхождений символов текста и сортируем их по убыванию количества вхождений с учетом лексиграфического порядка.

Частота вхождений символов в текст: l(2) и(1) м(1) р(1)  (1) !(1) ,(1) H(1) e(1) o(1)

[Шаг 2] Построение дерева кодирования Хаффмана:

Список узлов: l(2) и(1) м(1) р(1)  (1) !(1) ,(1) H(1) e(1) o(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: eo(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 1.

Список узлов: l(2) eo(2) и(1) м(1) р(1)  (1) !(1) ,(1) H(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: ,H(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 2.

Список узлов: l(2) eo(2) ,H(2) и(1) м(1) р(1)  (1) !(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел:  !(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 3.

Список узлов: l(2) eo(2) ,H(2)  !(2) и(1) м(1) р(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: мр(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 4.

Список узлов: l(2) eo(2) ,H(2)  !(2) мр(2) и(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: мри(3)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
```

```
Список узлов: l(2) ео(2) ,Н(2) и(1) м(1) р(1)  (1) !(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел:  !(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 3.

Список узлов: l(2) ео(2) ,Н(2)  !(2) и(1) м(1) р(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: мр(2)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 4.

Список узлов: l(2) ео(2) ,Н(2)  !(2) мр(2) и(1)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: мри(3)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла в начало списка.

Список узлов: мри(3) l(2) ео(2) ,Н(2)  !(2)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: ,Н !(4)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла в начало списка.

Список узлов: ,Н !(4) мри(3) l(2) ео(2)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: leo(4)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла на место с индексом 1.

Список узлов: ,Н !(4) leo(4) мри(3)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: leомри(7)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла в начало списка.

Список узлов: leомри(7) ,Н !(4)
Соеденим два узла с наименьшими весами (два последних узла) в один узел: leомри,Н !(11)
Вставляем новый узел в список узлов таким образом, чтобы список остался упорядоченным.
Вставка нового узла в начало списка.

[Шаг 3] Сопоставление кода каждому символу алфавита текста при помощи обхода дерева кодирования:

Символу 'l' соответствует код 000
Символу 'е' соответствует код 0010
Символу 'о' соответствует код 0011
Символу 'м' соответствует код 0100
Символу 'р' соответствует код 0101
Символу 'и' соответствует код 011
Символу ',' соответствует код 100
Символу 'Н' соответствует код 101
Символу ' ' соответствует код 110
Символу '!' соответствует код 111

[Шаг 4] Кодирование каждого символа текста:

'Н' -> 101
'е' -> 0010
'l' -> 000
'l' -> 000
'о' -> 0011
',' -> 100
' ' -> 110
'м' -> 0100
```

```
[Шаг 3] Сопоставление кода каждому символу алфавита текста при помощи обхода дерева кодирования:

Символу 'l' соответствует код 000
Символу 'е' соответствует код 0010
Символу 'о' соответствует код 0011
Символу 'м' соответствует код 0100
Символу 'р' соответствует код 0101
Символу 'и' соответствует код 011
Символу ',' соответствует код 100
Символу 'Н' соответствует код 101
Символу ' ' соответствует код 110
Символу '!' соответствует код 111

[Шаг 4] Кодирование каждого символа текста:

'Н' -> 101
'е' -> 0010
'l' -> 000
'l' -> 000
'о' -> 0011
',' -> 100
' ' -> 110
'м' -> 0100
'и' -> 011
'р' -> 0101
'!' -> 111

Кодирование текста завершено.

[Дерево кодирования] ('\0'('\0'('\0'('l'//)('\0'('е'//)('о'////)('\0'('\0'('м'//)('р'//))('и'////)('\0'('\0'(','//)('Н'///)('\0'(' '//)('!'/////)
[Закодированный текст] 10100100000000011100110010001101011111

Закодированный текст записан в файл 'encoded_text.txt'.
```

```
Доступные действия:

  1) Закодировать текст - Ввод с консоли.
  2) Закодировать текст - Ввод с файла.
  3) Раскодировать текст - Ввод с файла.
  4) Включить вывод промежуточных данных (уже включен).
  5) Отключить вывод промежуточных данных.
  6) Выйти из программы.

Выберите одно из вышеперечисленных действий: 3
Выбранное действие: 3
Считывание дерева кодирования из файла 'encoded_text.txt'...

[Дерево кодирования] ('\0'('\0'('и'//)('\0'('р'//)('\0'('П'//)('в'//))))('\0'('\0'('е'//)('\0'('м'//)('т'//)))('\0'(' '//)('\0'('!'//)(','//))))))

Считывание последовательности бит из файла 'encoded_text.txt'...

[Закодированный текст] 0110010000111100101111111101010000101110

Декодирование текста...

Спускаемся по дереву кодирования до листьев дерева. Если встречаем 0, то идем в левое поддерева, если 1 - то в правое.
При встрече листа дерева считываем очередной символ декодированного текста из листа, после чего переходим в корень дерева кодирования.
0110 соответствует символу 'П'
010 соответствует символу 'р'
00 соответствует символу 'и'
0111 соответствует символу 'в'
100 соответствует символу 'е'
1011 соответствует символу 'т'
1111 соответствует символу ','
110 соответствует символу ' '
1010 соответствует символу 'м'
00 соответствует символу 'и'
010 соответствует символу 'р'
1110 соответствует символу '!'

Декодирование текста завершено.

[Декодированный текст]
Привет, мир!

Декодированный текст записан в файл 'decoded_text.txt'.
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BinaryTree.h

```
#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include <sstream>


template <typename T>
class BinaryTree {
private:
    T element_;                    // Корень дерева
    size_t weight_ = 0;            // Вес корня дерева
    BinaryTree* right_ = nullptr;  // Указатель на правое поддерево
    BinaryTree* left_ = nullptr;   // Указатель на левое поддерево

public:
    BinaryTree();
    explicit BinaryTree(const T& element, size_t weight = 0);
    explicit BinaryTree(const std::string& expression);
    ~BinaryTree();

    bool createFromString(const char*& character);
    void setElement(const T& value);
    T getElement() const;
    size_t getWeight() const;
    void setWeight(size_t value);
    BinaryTree* getRightSubtree();
    BinaryTree* getLeftSubtree();
    const BinaryTree* getRightSubtree() const;
    const BinaryTree* getLeftSubtree() const;
    void setRightSubtree(BinaryTree* subtree);
    void setLeftSubtree(BinaryTree* subtree);
    bool isLeaf() const;
    std::string getString() const;
    std::string getElementString() const;
};

template<>
inline BinaryTree<char>::BinaryTree(): element_('\0') {}


template<typename T>
BinaryTree<T>::BinaryTree(const T& element, size_t weight): element_(element), weight_(weight) {}


template<>
inline BinaryTree<char>::BinaryTree(const std::string& expression) {
    const char* start = expression.c_str();
    bool correct = createFromString(start);

    if (!correct) {
        delete left_;
        delete right_;
        left_ = nullptr;
        right_ = nullptr;
    }
}
```

```
template <typename T>
BinaryTree<T>::~BinaryTree() {
    delete right_;
    delete left_;
}


template<>
inline bool BinaryTree<char>::createFromString(const char*& character) {
    // Очищаем поддеревья (в случае, если до вызова метода дерево уже использовалось)
    delete right_;
    delete left_;
    right_ = nullptr;
    left_ = nullptr;

    // Если скобочная запись начинается с '(', то это непустое БД
    if (*character == '(') {
        character++;

        if (*character == '\'') {
            character++;

            if (*character == '\\') {
                character++;

                if (*character == '0') {
                    element_ = '\0';
                    character++;
                } else {
                    return false;
                }

            } else if (*character != '\0') {
                element_ = *character;
                character++;
            } else {
                return false;
            }

            if (*character == '\'') {
                character++;
            } else {
                return false;
            }
        }

        // Создаем левое поддерево
        if (*character != '/') {
            left_ = new BinaryTree;
            bool correct = left_->createFromString(character);

            // Если не удалось корректно считать скобочную запись, то выходим
            if (!correct) {
                return false;
            }

        } else {
            character++;
        }
```

```
            // Создаем правое поддерево
            if (*character != '/') {
                right_ = new BinaryTree;
                bool correct = right_->createFromString(character);

                // Если не удалось корректно считать скобочную запись, то
выходим
                if (!correct) {
                    return false;
                }

            } else {
                character++;
            }

            // Если встречаем конец скобочной записи, то выходим
            if (*character == ')') {
                character++;
                return true;
            }
        }

        return false;
    }


    template<typename T>
    void BinaryTree<T>::setElement(const T& value) {
        element_ = value;
    }


    template<typename T>
    T BinaryTree<T>::getElement() const {
        return element_;
    }


    template<typename T>
    size_t BinaryTree<T>::getWeight() const {
        return weight_;
    }


    template<typename T>
    void BinaryTree<T>::setWeight(size_t value) {
        weight_ = value;
    }


    template<typename T>
    BinaryTree<T>* BinaryTree<T>::getRightSubtree() {
        return right_;
    }


    template<typename T>
    BinaryTree<T>* BinaryTree<T>::getLeftSubtree() {
        return left_;
    }
```

```
template<typename T>
const BinaryTree<T>* BinaryTree<T>::getRightSubtree() const {
    return right_;
}


template<typename T>
const BinaryTree<T>* BinaryTree<T>::getLeftSubtree() const {
    return left_;
}


template<typename T>
void BinaryTree<T>::setRightSubtree(BinaryTree* subtree) {
    delete right_;
    right_ = subtree;
}


template<typename T>
void BinaryTree<T>::setLeftSubtree(BinaryTree* subtree) {
    delete left_;
    left_ = subtree;
}


template<typename T>
bool BinaryTree<T>::isLeaf() const {
    return right_ == nullptr && left_ == nullptr;
}


template <>
inline std::string BinaryTree<char>::getString() const {
    std::ostringstream result;

    result << "('";

    if (element_ != '\0') {
        result << std::string(1, element_);
    } else {
        result << "\\0";
    }

    result << "'";

    if (left_ != nullptr) {
        result << left_->getString();
    } else {
        result << '/';
    }

    if (right_ != nullptr) {
        result << right_->getString();
    } else {
        result << '/';
    }

    result << ")";

    return result.str();
}
```

```
template<>
inline std::string BinaryTree<char>::getElementString() const {
    std::ostringstream result;

    if (element_ != '\0') {
        result << std::string(1, element_);
    }

    if (left_ != nullptr) {
        result << left_->getElementString();
    }

    if (right_ != nullptr) {
        result << right_->getElementString();
    }

    return result.str();
}


#endif // BINARY_TREE_H
```

Название файла: Color.h

```
#ifndef COLOR_H
#define COLOR_H


enum class Color {
      Black = 0,
      Blue = 1,
      Green = 2,
      Cyan = 3,
      Red = 4,
      Magenta = 5,
      Brown = 6,
      LightGray = 7,
      DarkGray = 8,
      LightBlue = 9,
      LightGreen = 10,
      LightCyan = 11,
      LightRed = 12,
      LightMagenta = 13,
      Yellow = 14,
      White = 15
};


#endif // COLOR_H
```

Название файла: Decoder.h

```
#ifndef DECODER_H
#define DECODER_H

#include <vector>
#include <string>
#include <map>

#include "BinaryTree.h"
```

```
using Character = char;
using BitSequence = std::vector<bool>;
using CharacterCodes = std::map<Character, BitSequence>;


class Decoder final {
protected:
	BinaryTree<Character>* tree_ = nullptr;

public:
	~Decoder();

	const BinaryTree<Character>* getCodingTree() const;
	bool setCodingTree(const std::string& expression);
	std::string decodeText(const BitSequence& sequence);
};


#endif // DECODER_H
```

Название файла: Decoder.cpp

```
#include "Decoder.h"
#include "Logger.h"
#include "Exception.h"


Decoder::~Decoder() {
    delete tree_;
}


const BinaryTree<Character>* Decoder::getCodingTree() const {
    return tree_;
}


bool Decoder::setCodingTree(const std::string& expression) {
	delete tree_;
	tree_ = new BinaryTree<Character>;
    const char* start = expression.c_str();
    return tree_->createFromString(start);
}


std::string Decoder::decodeText(const BitSequence& sequence) {
	std::stringstream encoded_text;
	BinaryTree<char>* subtree = tree_;

	Logger::log("Спускаемся по дереву кодирования до листьев дерева. Если встречаем 0, то идем в левое поддерева, если 1 - то в правое.\n", MessageType::Debug);
	Logger::log("При встрече листа дерева считываем очередной символ декодированного текста из листа, после чего переходим в корень дерева кодирования.\n", MessageType::Debug);

	// Проход по битам закодированного текста
	for (const auto& bit : sequence) {
		Logger::log(std::to_string(bit), MessageType::Debug);
```

```
                  // В зависимости от значения бита происходит переход либо в
правое поддерево, либо в левое поддерево
                  if (bit) {
                        subtree = subtree->getRightSubtree();
                  } else {
                        subtree = subtree->getLeftSubtree();
                  }

                  if (subtree == nullptr) {
                        throw Exception("Subtree is nullptr.");
                  }

                  // Если достигнут лист дерева (очередной символ текста), то он
добавляется в текст, и происходит переход в корень дерева
                  if (subtree->isLeaf()) {
                        Logger::log(" соответствует символу '" + std::string(1,
subtree->getElement()) + "'\n", MessageType::Debug);
                        encoded_text << subtree->getElement();
                        subtree = tree_;
                  }
            }

            return encoded_text.str();
      }
```

Название файла: Encoder.h

```
      #ifndef ENCODER_H
      #define ENCODER_H

      #include <map>
      #include <vector>
      #include <string>

      #include "BinaryTree.h"

      using Character = char;
      using BitSequence = std::vector<bool>;
      using CharacterCodes = std::map<Character, BitSequence>;
      using CharacterFrequency = std::pair<Character, size_t>;
      using CharacterFrequencies = std::vector<CharacterFrequency>;


      class Encoder {
      protected:
            CharacterFrequencies frequencies_;        // Частота символов в тексте
            BinaryTree<Character>* tree_;             // Дерево кодирования
            CharacterCodes codes_;                          //    Коды    символов
алфавита

      public:
            virtual ~Encoder();

            const BinaryTree<Character>* getCodingTree() const;
            const CharacterCodes& getCharacterCodes() const;
            const CharacterFrequencies& getCharacterFrequencies() const;
            const    CharacterFrequencies&    calculateCharacterFrequencies(const
std::string& text);
            virtual BitSequence encodeText(const std::string& text) = 0;
      };
```

```
#endif // ENCODER_H
```

Название файла: Encoder.cpp

```
#include <algorithm>

#include "Encoder.h"
#include "Logger.h"


Encoder::~Encoder() {
	delete tree_;
}


const BinaryTree<Character>* Encoder::getCodingTree() const {
	return tree_;
}


const CharacterCodes& Encoder::getCharacterCodes() const {
	return codes_;
}


const CharacterFrequencies& Encoder::getCharacterFrequencies() const {
	return frequencies_;
}


const CharacterFrequencies& Encoder::calculateCharacterFrequencies(const
std::string& text) {
	frequencies_.clear();

	Logger::log("\n[Шаг 1] Считаем частоту вхождений символов текста и
сортируем их по убыванию количества вхождений с учетом лексиграфического
порядка.\n\n", MessageType::Debug);

	// Пробегаемся по символам текста и подсчитываем их
	for (auto& character : text) {
		bool found = false;

		for (auto& element : frequencies_) {
			if (element.first == character) {
				element.second++;
				found = true;
				break;
			}
		}

		if (!found) {
			frequencies_.push_back(CharacterFrequency(character, 1));
		}
	}

	// Сортируем частоты символов по убыванию с учетом лексеграфического
порядка
	std::sort(frequencies_.begin(), frequencies_.end(), [](const
CharacterFrequency& f1, const CharacterFrequency& f2) {
		if (f1.second != f2.second) {
			return f1.second > f2.second;
		} else {
```

```
                        return f1.first < f2.first;
                  }
            });

            Logger::log("Частота вхождений символов в текст: ",
MessageType::Debug);

            for (auto& f : frequencies_) {
                  Logger::log(std::string(1, f.first) + "(" +
std::to_string(f.second) + ") ", MessageType::Debug);
            }

            Logger::log("\n", MessageType::Debug);

            return frequencies_;
      }
```

Название файла: Exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <string>


class Exception final {
private:
      const std::string message_;

public:
      explicit Exception(const std::string& message);
      const std::string& getMessage() const;
};


#endif // EXCEPTION_H
```

Название файла: HuffmanEncoder.h

```
#ifndef HUFFMAN_ENCODER_H
#define HUFFMAN_ENCODER_H

#include "Encoder.h"


class HuffmanEncoder final: public Encoder {
private:
      void calculateCharactersTree();
      void calculateCharacterCodes(const BinaryTree<Character>* tree,
BitSequence& path);

public:
      BitSequence encodeText(const std::string& text) override;
};


#endif // HUFFMAN_ENCODER_H
```

Название файла: HuffmanEncoder.cpp

```
#include <algorithm>

#include "HuffmanEncoder.h"
#include "Logger.h"
#include "Utils.h"


void HuffmanEncoder::calculateCharactersTree() {
	std::vector<BinaryTree<Character>*> tree_nodes(frequencies_.size());
// Вектор узлов дерева кодирования

	for (size_t i = 0; i < tree_nodes.size(); i++) {
	     tree_nodes[i] = new BinaryTree<Character>(frequencies_[i].first,
frequencies_[i].second);
	}

	// До тех пор, пока в векторе узлов дерева кодирования больше 1 узла
	while (tree_nodes.size() > 1) {
		size_t nodes_size = tree_nodes.size();

		Logger::log("\nСписок узлов: ", MessageType::Debug);

		for (size_t i = 0; i < nodes_size; i++) {
			Logger::log(tree_nodes[i]->getElementString() + "(" +
std::to_string(tree_nodes[i]->getWeight()) + ") ", MessageType::Debug);
		}

		// Производим объединение двух последних узлов дерева
кодирования (с наименьшими весами)
		BinaryTree<Character>* new_node = new
BinaryTree<Character>('\0', tree_nodes[nodes_size - 1]->getWeight() +
tree_nodes[nodes_size - 2]->getWeight());
		new_node->setLeftSubtree(tree_nodes[nodes_size - 2]);
		new_node->setRightSubtree(tree_nodes[nodes_size - 1]);
		tree_nodes.resize(nodes_size - 2);
		nodes_size -= 2;

		Logger::log("\nСоеденим два узла с наименьшими весами (два
последних узла) в один узел: " + new_node->getElementString() + "(" +
std::to_string(new_node->getWeight()) + ")\n", MessageType::Debug);
		Logger::log("Вставляем новый узел в список узлов таким образом,
чтобы список остался упорядоченным.\n", MessageType::Debug);

		// Вставляем новый узел в вектор узлов дерева кодирования
		if (nodes_size == 0 || new_node->getWeight() >
tree_nodes[0]->getWeight()) {
			tree_nodes.insert(tree_nodes.begin(), new_node);
			Logger::log("Вставка нового узла в начало списка.\n",
MessageType::Debug);
		} else if (new_node->getWeight() <= tree_nodes[nodes_size -
1]->getWeight()) {
			tree_nodes.insert(tree_nodes.end(), new_node);
			Logger::log("Вставка нового узла в конец списка.\n",
MessageType::Debug);
		} else {
			for (auto i = tree_nodes.begin(); i < tree_nodes.end() -
1; ++i) {
				if ((*i)->getWeight() >= new_node->getWeight() &&
(*(i + 1))->getWeight() < new_node->getWeight()) {
					tree_nodes.insert(i + 1, new_node);
					Logger::log("Вставка нового узла на место с
индексом " + std::to_string(std::distance(tree_nodes.begin(), i + 1)) + ".\n",
MessageType::Debug);
```

```
                                    break;
                              }
                        }
                  }
            }

            if (frequencies_.size() == 0) {
                  tree_nodes.push_back(new BinaryTree<Character>);
            } else if (frequencies_.size() == 1) {
                  BinaryTree<Character>*          new_node          =          new
BinaryTree<Character>('\0', tree_nodes[0]->getWeight());
                  new_node->setRightSubtree(tree_nodes[0]);
                  tree_nodes[0] = new_node;
            }

            // Устанавливаем новое дерево кодирования
            delete tree_;
            tree_ = tree_nodes[0];
      }


      void  HuffmanEncoder::calculateCharacterCodes(const  BinaryTree<Character>*
tree, BitSequence& path) {
            // Если достигли листа дерева - то добавляем в словарь кодов символ и
соответствующий ему код
            if (tree->isLeaf()) {
                  Logger::log("Символу '" + std::string(1, tree->getElement()) +
"'   соответствует   код   "   +   Utils::bitSequenceToString(path)   +   "\n",
MessageType::Debug);
                  codes_[tree->getElement()] = path;
                  return;
            }

            // Иначе производим поиск листьев в поддеревьях данного дерева
            const BinaryTree<Character>* left = tree->getLeftSubtree();
            const BinaryTree<Character>* right = tree->getRightSubtree();

            if (left != nullptr) {
                  path.push_back(false);
                  calculateCharacterCodes(left, path);
                  path.pop_back();
            }

            if (right != nullptr) {
                  path.push_back(true);
                  calculateCharacterCodes(right, path);
                  path.pop_back();
            }
      }


      BitSequence HuffmanEncoder::encodeText(const std::string& text) {
            BitSequence encoded_text;

            // Находим частоту символов текста
            calculateCharacterFrequencies(text);

            // Строим дерево Хаффмана
            Logger::log("\n[Шаг 2] Построение дерева кодирования Хаффмана:\n",
MessageType::Debug);

            calculateCharactersTree();
```

```
            Logger::log("\n[Шаг 3] Сопоставление кода каждому символу алфавита
текста при помощи обхода дерева кодирования:\n\n", MessageType::Debug);

            calculateCharacterCodes(tree_, encoded_text);

            Logger::log("\n[Шаг 4] Кодирование каждого символа текста:\n\n",
MessageType::Debug);

            // Пробегаемся по символам текста и кодируем их
            for (auto& character : text) {
                  std::stringstream code_string;
                  BitSequence& code = codes_[character];

                  for (size_t i = 0; i < code.size(); i++) {
                        encoded_text.push_back(code[i]);
                        code_string << code[i];
                  }

                  Logger::log("'" + std::string(1, character) + "' -> " +
code_string.str() + "\n", MessageType::Debug);
            }
            Logger::log("\n", MessageType::Debug);

            return encoded_text;
      }
```

Название файла: Logger.h

```
      #ifndef LOGGER_H
      #define LOGGER_H

      #include <fstream>

      #include "MessageType.h"
      #include "Color.h"


      class Logger final {
          int indent_size_ = 4;         // Размер отступа
          bool debug_mode_ = false;     // Режим вывода отладочных сообщений
          bool file_output_ = false;    // Вывод сообщений в файл
          std::ofstream file_;          // Дескриптор выходного файла

          Logger() = default;
          Logger(const Logger&) = delete;
          Logger(Logger&&) = delete;
          Logger& operator=(const Logger&) = delete;
          Logger& operator=(Logger&&) = delete;
          ~Logger();

      public:
          static Logger& getInstance();
          static void log(const std::string& message, MessageType type =
MessageType::Common, int indents = 0);
          static void setConsoleColor(Color text_color, Color background_color);
          void setOutputFile(const std::string& filepath);
          void setDebugMode(bool value);
          bool getDebugMode();
          static std::string getCurrentDateTime();
      };
```

```
#endif // LOGGER_H
```

Название файла: Logger.cpp

```
#include <iostream>
#include <ctime>

#include "Logger.h"
#include "Windows.h"


Logger::~Logger() {
    file_.close();
}


Logger& Logger::getInstance() {
    static Logger instance;
    return instance;
}


void Logger::setDebugMode(bool value) {
    debug_mode_ = value;
}


bool Logger::getDebugMode() {
    return debug_mode_;
}


void Logger::setOutputFile(const std::string& filepath) {
    file_.close();
    file_.open(filepath);

    // Проверка открытия файла
    if (!file_.is_open()) {
        file_output_ = false;
        Logger::log("An error occurred while opening the file to write logs:
" + filepath + "\n", MessageType::Error);
        return;
    }

    file_output_ = true;
}


void Logger::log(const std::string& message, MessageType type, int indents)
{
    Logger& logger = Logger::getInstance();

    // Если включен режим отладки и сообщение - отладочное, то происходит
выход из функции
    if (type == MessageType::Debug && !logger.debug_mode_) {
        return;
    }

    std::string indent(logger.indent_size_ * indents, ' '); // Отступ от
начала строки

    if (type == MessageType::Common || type == MessageType::Debug) {
```

```
            std::cout << indent << message;
        } else {
            std::cerr << indent << message;
        }

        if (logger.file_output_) {
            logger.file_ << indent << message;
        }
    }


    void Logger::setConsoleColor(Color text_color, Color background_color) {
        HANDLE h_console = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(h_console,
(WORD)((static_cast<int>(background_color) << 4) | static_cast<int>(text_color)));
    }


    std::string Logger::getCurrentDateTime() {
        tm timeinfo;                                              // Структура с
информацией о времени
        time_t timestamp = time(nullptr);                    // Временная метка
        errno_t  error  =  localtime_s(&timeinfo,  &timestamp);  //  Получение
информации о времени
        char buffer[40];                                     // Буфер для строки

        //  Если  возникла  ошибка  при  получении  информации  о  времени,  то
возвращаем "00-00-00_00-00-00"
        if (error) {
            return "00-00-00_00-00-00";
        } else {
            strftime(buffer, sizeof(buffer), "%d-%m-%y_%H-%M-%S", &timeinfo);
        }

        return buffer;
    }
```

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <memory>

#include "Logger.h"
#include "Exception.h"
#include "Windows.h"
#include "ShannonFanoEncoder.h"
#include "HuffmanEncoder.h"
#include "Decoder.h"
#include "Utils.h"


int main() {
      bool is_loop_enabled = true;     // Продолжать ли работу основного
цикла работы программы
      bool  is_debug_mode  =  true;          //  Включен  ли  режим  вывода
промежуточных данных
      Logger& logger = Logger::getInstance();

      // Настройка русского языка для консоли
      SetConsoleCP(1251);
```

```
            SetConsoleOutputCP(1251);

            // Настройка файла вывода сообщений логгера и установка режима вывода
промежуточных данных
            logger.setOutputFile("logs\\"   +   Logger::getCurrentDateTime()   +
".log");
          logger.setDebugMode(is_debug_mode);

          // Основной цикл работы программы
          while (is_loop_enabled) {
              int action = -1;    // Выбранное действие
              int algorithm = -1; // Выбранный алгоритм

              // Считывание выбора действия пользователя
              Logger::log("\nДоступные действия:\n\n  1) Закодировать текст - Ввод
с консоли.\n  2) Закодировать текст - Ввод с файла.\n  3) Раскодировать текст -
Ввод с файла.\n");

              if (is_debug_mode) {
                  Logger::setConsoleColor(Color::Green, Color::Black);
                  Logger::log("   4) Включить вывод промежуточных данных (уже
включен).\n");
                  Logger::setConsoleColor(Color::LightGray, Color::Black);
              } else {
                  Logger::log("  4) Включить вывод промежуточных данных.\n");
              }

              if (!is_debug_mode) {
                  Logger::setConsoleColor(Color::Green, Color::Black);
                  Logger::log("   5) Отключить вывод промежуточных данных (уже
выключен).\n");
                  Logger::setConsoleColor(Color::LightGray, Color::Black);
              } else {
                  Logger::log("  5) Отключить вывод промежуточных данных.\n");
              }

              Logger::log("  6) Выйти из программы.\n\n");
              std::cout << "Выберите одно из вышеперечисленных действий: ";
              std::cin >> action;
              Utils::clearInput();

              if (action < 1 || action > 6) {
                  Logger::log("Выбрано некорректное действие.\n");
                  continue;
              }

              Logger::log("Выбранное действие: " + std::to_string(action) + "\n");

              // Выбор алгоритма кодирования
              if (action < 3) {
                  Logger::log("\nДоступные   алгоритмы   кодирования:\n\n      1)
Алгоритм Шеннона-Фано.\n  2) Алгоритм Хаффмана.\n\n");
                  std::cout << "Выберите одно из вышеперечисленных алгоритмов: ";
                  std::cin >> algorithm;
                  Utils::clearInput();

                  while (algorithm < 1 || algorithm > 2) {
                      std::cout   <<   "Выбран   некорректный   алгоритм.   Выберите
алгоритм снова: ";
                      std::cin >> algorithm;
                      Utils::clearInput();
                  }
```

```
                Logger::log("Выбранный алгоритм: " + std::to_string(algorithm)
+ "\n\n");
            }

            // Кодирование текста
            if (action == 1 || action == 2) {
                std::stringstream text;

                // Считывание текста с консоли
                if (action == 1) {
                    std::string line;

                    std::cout << "Введите текст: ";
                    std::getline(std::cin, line);
                    text << line;

                    Logger::log("Введенный текст: " + line + "\n\n");
                }
                // Считывание текста с файла
                else {
                    std::string path, line;
                    std::ifstream input_file;

                    std::cout << "Введите путь к файлу: ";
                    std::getline(std::cin, path);
                    Logger::log("Введенный путь к файлу: " + path + "\n");

                    input_file.open(path);

                    // Если файл не удалось открыть
                    if (!input_file.is_open()) {
                        Logger::log("При открытии файла произошла ошибка: " +
path + "\n", MessageType::Error);
                        input_file.close();
                        continue;
                    }

                    Logger::log("Считывание текста из файла '" + path +
"'...\n");

                    while (!input_file.eof()) {
                        std::getline(input_file, line);
                        text << line << "\n";
                    }

                    Logger::log("[Текст из файла]\n" + text.str() + "\n");
                    input_file.close();
                }

                std::shared_ptr<Encoder> encoder;    // Указатель на кодировщик
текста
                std::string tree_expression;         // Скобочная запись дерева
кодирования
                std::string encoded_text_string;        // Строка, содержащая
закодированный текст
                CharacterCodes character_codes;         // Коды закодированных
символов
                BitSequence encoded_text;            // Последовательность бит
закодированного текста

                if (algorithm == 1) {
                    encoder = std::make_shared<ShannonFanoEncoder>();      //
Кодировщик Шеннона-Фано
```

```
                } else {
                    encoder = std::make_shared<HuffmanEncoder>();          //
Кодировщик Хаффмана
                }

                Logger::log("Кодирование текста...\n");

                // Кодирование текста
                encoded_text = encoder->encodeText(text.str());
                tree_expression = encoder->getCodingTree()->getString();
                character_codes = encoder->getCharacterCodes();
                encoded_text_string = Utils::bitSequenceToString(encoded_text);

                Logger::log("Кодирование текста завершено.\n\n");
                Logger::log("[Дерево кодирования] " + tree_expression + "\n");
                Logger::log("[Закодированный текст] " + encoded_text_string +
"\n\n");

                // Сохранение результата в файл
                std::ofstream output_file("encoded_text.txt");

                // Если файл не удалось открыть
                if (!output_file.is_open()) {
                    Logger::log("При открытии файла произошла ошибки:
encoded_text.txt\n", MessageType::Error);
                    output_file.close();
                    continue;
                }

                output_file << tree_expression << "\n\n" << encoded_text_string;

                Logger::log("Закодированный текст записан в файл
'encoded_text.txt'.\n");
                output_file.close();
            }
            // Декодирование текста
            else if (action == 3) {
                std::ifstream input_file("encoded_text.txt");
                std::stringstream expression; // Скобочная запись дерева
кодирования
                std::string line("\\");

                // Если файл не удалось открыть
                if (!input_file.is_open()) {
                    Logger::log("При открытии файла произошла ошибки:
encoded_text.txt\n", MessageType::Error);
                    input_file.close();
                    continue;
                }

                Logger::log("Считывание дерева кодирования из файла
'encoded_text.txt'...\n\n");

                // Считываем скобочную запись дерева кодирования
                while (line != "") {
                    line = "";
                    std::getline(input_file, line);
                    expression << line << "\n";
                }

                Decoder decoder;            // Декодировщик
                BitSequence encoded_text;       // Последовательность бит
закодированного текста
```

```
                if (!decoder.setCodingTree(expression.str())) {
                    Logger::log("Скобочная запись дерева кодирования
некорректна: " + expression.str() + "\n", MessageType::Error);
                    continue;
                } else {
                    Logger::log("[Дерево кодирования] " +
decoder.getCodingTree()->getString() + "\n\n");
                }

                Logger::log("Считывание последовательности бит из файла
'encoded_text.txt'...\n\n");

                // Считываем последовательность бит закодированного текста
                std::getline(input_file, line);
                encoded_text.reserve(line.size());
                input_file.close();

                for (auto& character : line) {
                    if (character == '1') {
                        encoded_text.push_back(true);
                    }
                    else if (character == '0') {
                        encoded_text.push_back(false);
                    }
                }

                Logger::log("[Закодированный текст] " + line + "\n\n");
                Logger::log("Декодирование текста...\n\n");

                // Декодирование текста
                std::string decoded_text;

                try {
                    decoded_text = decoder.decodeText(encoded_text);
                } catch (Exception& error) {
                    Logger::log("Произошла ошибка при декодировании текста: " +
error.getMessage() + "\n", MessageType::Error);
                    continue;
                }

                Logger::log("\nДекодирование текста завершено.\n\n");
                Logger::log("[Декодированный текст]\n" + decoded_text + "\n\n");

                // Сохранение результата в файл
                std::ofstream output_file("decoded_text.txt");

                // Если файл не удалось открыть
                if (!output_file.is_open()) {
                    Logger::log("При открытии файла произошла ошибка:
decoded_text.txt\n", MessageType::Error);
                    output_file.close();
                    continue;
                }

                output_file << decoded_text;

                Logger::log("Декодированный текст записан в файл
'decoded_text.txt'.\n");
                output_file.close();
            }
            // Включение или отключение вывода промежуточной информации
            else if (action == 4 || action == 5) {
```

```
                // Включение
                if (action == 4) {
                    is_debug_mode = true;
                    Logger::log("Вывод промежуточных данных включен.\n");
                }
                // Отключение
                else {
                    is_debug_mode = false;
                    Logger::log("Вывод промежуточных данных выключен.\n");
                }

                logger.setDebugMode(is_debug_mode);
            }
            // Выход из программы
            else {
                is_loop_enabled = false;
            }
        }

          return 0;
    }
```

Название файла: MessageType.h

```
    #ifndef MESSAGE_TYPE_H
    #define MESSAGE_TYPE_H


    enum class MessageType {
        Common,
        Error,
        Debug
    };


    #endif // MESSAGE_TYPE_H
```

Название файла: ShannonFanoEncoder.h

```
    #ifndef SHANNON_FANO_ENCODER_H
    #define SHANNON_FANO_ENCODER_H

    #include "Encoder.h"


    class ShannonFanoEncoder final: public Encoder {
    private:
          BinaryTree<Character>*
calculateCharactersTreeAndCodes(CharacterFrequencies&  frequency,  BitSequence&
path);

    public:
          BitSequence encodeText(const std::string& text) override;
    };


    #endif // SHANNON_FANO_ENCODER_H
```

Название файла: ShannonFanoEncoder.cpp

```
#include <sstream>
#include <cmath>

#include "ShannonFanoEncoder.h"
#include "Logger.h"
#include "Utils.h"


BinaryTree<Character>*
ShannonFanoEncoder::calculateCharactersTreeAndCodes(CharacterFrequencies&
frequency, BitSequence& path) {
	BinaryTree<Character>* tree = new BinaryTree<Character>;   //  Новый
узел дерева кодирования
	CharacterFrequencies left;
	// Частота символов для левого поддерева
	CharacterFrequencies right;
	// Частота символов для правого поддерева
	std::string path_string = Utils::bitSequenceToString(path);//   Путь
до текущего узла дерева
	long long sum = 0;                                  //    Сумма    частот
символов текущего узла дерева

	Logger::log("Частота символов для текущего узла дерева (путь до узла
- " + path_string + "): ", MessageType::Debug, path.size());

	for (size_t i = 0; i < frequency.size(); i++) {
		Logger::log(std::string(1,                frequency[i].first),
MessageType::Debug);
		sum += frequency[i].second;
	}

	Logger::log("(" + std::to_string(sum) + ")\n", MessageType::Debug);

	tree->setWeight(sum);

	// Если кодировать нечего - то возвращается пустое дерево
	if (frequency.size() == 0) {
		return tree;
	}
	// Если в списке частоты символов осталось одно значение, то
происходит присваивание текущему узлу дерева символа
	// и добавление кода текущего символа в массив кодов символов алфавита
	else if (frequency.size() == 1) {
		if (path.size() == 0) {
			path.push_back(true);

      tree->setRightSubtree(calculateCharactersTreeAndCodes(frequency, path));
			path.pop_back();
		} else {
			Logger::log("Помещаем   символ   '"   +   std::string(1,
frequency[0].first) + "' в текущий узел дерева (путь до узла - " + path_string +
") и задаем в качестве кода текущего символа путь до текущего узла дерева.\n",
MessageType::Debug, path.size());
			codes_[frequency[0].first] = path;
			tree->setElement(frequency[0].first);
		}
		return tree;
	}
	// Если в списке частоты символов осталось более одного значение, то
происходит разделение этого списка на два
	else {
		size_t middle_index = 0;
		long long left_sum = 0;
```

```
                long long right_sum = 0;
                long long min_delta = LLONG_MAX;

                for (size_t i = 0; i < frequency.size(); i++) {
                      right_sum += frequency[i].second;
                }

                // Находим такой k, при котором различие между суммой частот
двух списков минимально
                for (size_t k = 0; k < frequency.size(); k++) {
                      left_sum += frequency[k].second;
                      right_sum -= frequency[k].second;

                      if (abs(right_sum - left_sum) < abs(min_delta)) {
                            middle_index = k;
                            min_delta = right_sum - left_sum;
                      }
                }

                Logger::log("Разделяем список частот. Индекс разделяющего
элемента: " + std::to_string(middle_index) + ".\n", MessageType::Debug,
path.size());

                // Заполняем левый и правый подсписки списка
                for (size_t i = 0; i <= middle_index; i++) {
                      left.push_back(frequency[i]);
                }

                for (size_t i = middle_index + 1; i < frequency.size(); i++) {
                      right.push_back(frequency[i]);
                }

                // Если правая сумма меньше левой, то меняем их местами
                if (min_delta < 0) {
                      std::swap(left, right);
                }

                left_sum = 0;
                right_sum = 0;

                Logger::log("Частоты символов для левого поддерева: ",
MessageType::Debug, path.size());

                for (size_t i = 0; i < left.size(); i++) {
                      Logger::log(std::string(1,               left[i].first),
MessageType::Debug);
                      left_sum += left[i].second;
                }

                Logger::log("(" + std::to_string(left_sum) + ")\n",
MessageType::Debug);
                Logger::log("Частоты символов для правого поддерева: ",
MessageType::Debug, path.size());

                for (size_t i = 0; i < right.size(); i++) {
                      Logger::log(std::string(1,              right[i].first),
MessageType::Debug);
                      right_sum += right[i].second;
                }

                Logger::log("(" + std::to_string(right_sum) + ")\n",
MessageType::Debug);
          }
```

```
            // Создаем левое поддерево
            Logger::log("Создаем левое поддерево для текущего узла дерева (путь
до узла - " + path_string + "0):\n", MessageType::Debug, path.size());
            path.push_back(false);
            tree->setLeftSubtree(calculateCharactersTreeAndCodes(left, path));
            path.pop_back();

            // Создаем правое поддерево
            Logger::log("Создаем правое поддерево для текущего узла дерева (путь
до узла - " + path_string + "1):\n", MessageType::Debug, path.size());
            path.push_back(true);
            tree->setRightSubtree(calculateCharactersTreeAndCodes(right, path));
            path.pop_back();

            return tree;
      }


      BitSequence ShannonFanoEncoder::encodeText(const std::string& text) {
            BitSequence encoded_text;

            // Находим частоту символов текста
            calculateCharacterFrequencies(text);

            // Строим дерево Шеннона-Фано
            Logger::log("\n[Шаг 2] Построение дерева кодирования Шеннона-Фано и
сопоставление кода каждому символу алфавита текста.\n\n", MessageType::Debug);

            delete tree_;
            tree_ = calculateCharactersTreeAndCodes(frequencies_, encoded_text);
            encoded_text.clear();

            Logger::log("\n[Шаг 3] Кодирование каждого символа текста:\n\n",
MessageType::Debug);

            // Пробегаемся по символам текста и кодируем их
            for (auto& character : text) {
                  std::stringstream code_string;
                  BitSequence& code = codes_[character];

                  for (size_t i = 0; i < code.size(); i++) {
                        encoded_text.push_back(code[i]);
                        code_string << code[i];
                  }

                  Logger::log("Символу '" + std::string(1, character) + "'
соответствует код " + code_string.str() + "\n", MessageType::Debug);
            }

            Logger::log("\n", MessageType::Debug);

            return encoded_text;
      }
```

Название файла: Utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include <string>
#include <vector>
```

```
using BitSequence = std::vector<bool>;


class Utils final {
public:
	static std::string bitSequenceToString(const BitSequence& sequence);
	static void clearInput();
};


#endif // UTILS_H
```

Название файла: Utils.cpp

```
#include <iostream>
#include <sstream>

#include "Utils.h"


std::string Utils::bitSequenceToString(const BitSequence& sequence) {
	std::stringstream string;

	for (const auto& bit : sequence) {
		string << std::to_string(bit);
	}

	return string.str();
}


void Utils::clearInput() {
	// Удаляем из потока несчитанные символы до перевода на новую строку,
включая его
	std::cin.clear();
	while (std::cin.get() != '\n');
}
```