**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 | 5 |
|  | Формальная постановка задачи | 5 |
| 1. | Описание алгоритма | 6 |
| 1.1. | Кодирование алгоритмом Шеннона-Фано | 6 |
| 1.2. | Кодирование алгоритмом Хаффмана | 7 |
| 1.3. | Декодирование | 7 |
| 2. | Описание структур данных и функций | 9 |
| 2.1. | Перечисление MessageType | 9 |
| 2.2. | Перечисление Color | 10 |
| 2.3. | Класс Logger | 11 |
| 2.4. | Класс Exception | 13 |
| 2.5. | Класс BinaryTree | 14 |
| 2.6. | Класс Encoder | 17 |
| 2.7. | Класс ShannonFanoEncoder | 19 |
| 2.8. | Класс HuffmanEncoder | 20 |
| 2.9. | Класс Decoder | 21 |
| 2.10. | Класс Utils | 22 |
| 2.11. | Функция main | 23 |
| 3. | Описание интерфейса пользователя | 25 |
|  | Заключение | 26 |
|  | Список использованных источников | 27 |
|  | Приложение А. Тестирование | 28 |
|  | Приложение Б. Исходный код программы | 32 |

**введение**

**Цель работы.**

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

**Задачи.**

* Изучение языка программирования С++;
* 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.

**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 |
|  | Выберите одно из вышеперечисленных действий: -1 | Выбрано некорректное действие. |
|  | Выберите одно из вышеперечисленных действий: 9 | Выбрано некорректное действие. |
|  | Выберите одно из вышеперечисленных действий: 2  Выбранное действие: 2  Выберите одно из вышеперечисленных алгоритмов: 4 | Выбран некорректный алгоритм. Выберите алгоритм снова: |
|  | 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 |
|  | 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 |
|  | 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] 1101110000011111011100011001001111010 |
|  | Выберите одно из вышеперечисленных действий: 3  Выбранное действие: 3  [Дерево кодирования] ('\0'('\0'('l'//)('\0'('и'//)('\0'('м'//)('р'//))))('\0'('\0'(' '//)('\0'('!'//)(','//)))('\0'('H'//)('\0'('e'//)('o'//)))))  [Закодированный текст] 1101110000011111011100011001001111010 | [Декодированный текст]  Hello, мир! |
|  | 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] 001001100001001100000111101001001111001110100010110101 |
|  | 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] |
|  | 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] |
|  | Выберите одно из вышеперечисленных действий: 1  Выбранное действие: 1  Выберите одно из вышеперечисленных алгоритмов: 1  Выбранный алгоритм: 1  Введите текст: aaaaa  Введенный текст: aaaaa | [Дерево кодирования] ('\0'/('a'//))  [Закодированный текст] 11111 |
|  | 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 |
|  | 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] 100000000011101000010001111111110110010001100001010101001010010010100100011100010111101011110011010011110000001100000101011000110010001001110010101011011101011010000101001001110111000100110010000101111010110000010101101100101101001000011011110000001100101100000101110111111000110111011111100111110010001000111011101111110011100101011011011011100110101111001101001111000000110000000101101100011011110101010001010010010110110010111100000000010000011000000011011000101000111111000100111010000010000110011111010111010011011001111000000111011100010011001000010111101011000001100001010101001010010011000100000010101010000100011000000011001001010110110010001001011101101101110001010001111011111110011110000000110100111011011100100011010011110000001101100001000010100001110000010111000000100100101000011100000011010101111011010100000100001100110010110100010111100001001000000010100000110000010101100011001000100110000100010100000111111001010101101110101101100010100101001011000001010001100001110001110011110100000001100100110101111001100010000001100010001110101100100000110010011000101100000100010011000100010100001100100000101000111111110110011010100100011100101011010010111111011000101001001110111000100110010000101111010101000001000111001000110111111100001001010110100011000010101010010100100110001000000100010011101110001001100100001011110101100000110001000111010110010000011001001111110110100101100000110001011000001101001111000000101101000001110101001110011111010010100100011001001000110100010001001111001111100010000111100110011001000110100000011001111101011101001101100111100000010000011010111000111011010001101001011110000010110110110111001111111101100110100111100000011000001010110001100100010010100100011100010111101011110011100100000111000000101111001010000011111101100110100111100000010001011100011100100111110110101000010100100111111110110011010011110000001011101011000010101010000100000110000000101101100011011110101010001010010010110100110001100101100100000101001000001011101001100010100111111111001010110100110000011010011110000001100000101011000110010001001011101000100001000111111010101000110100101111110110010101010000010111100000001110010110101101100000011001101000001001000011001001010110110011110001010000101100000110101000010110011111010 |
|  | Выберите одно из вышеперечисленных действий: 4  Выбранное действие: 4 | Вывод промежуточных данных включен. |
|  | Выберите одно из вышеперечисленных действий: 5  Выбранное действие: 5 | Вывод промежуточных данных выключен. |

**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 ошибка: 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 ошибка: 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');

}