Алгоритм Хаффмана

Алгоритм Хаффмана реализован в классе `HuffmanCoder`. Класс предоставляет два публичных метода для работы с файлами.

Метод-кодировщик. Принимает 2 аргумента: `String filePath` - путь до текстового файла с кодировкой UTF-8 и `String codeFilePath` - путь до текстового файла, в который будем записан полученный код.

```
public void encode(String filePath, String codeFilePath)
```

Метод-декодировщик. Принимает 2 аргумента: `String codeFilePath` - путь до текстового файла, хранящего код Хаффмана, `String filePath` - путь до текстового файла, в который будет декодирован зашифрованный ранее код.

```
public void decode(String codeFilePath, String filePath)
```

Кодировщик

Описание работы.

1. Очищаются метаданные, хранящиеся в классе, которые могли быть заполнены при предыдущих обращениях к методам `encode` и `decode`. К этим метаданным относятся `HashMap<Character, Integer> characterRateMap` - HashMap с частотами символов в кодируемом тексте, `HashMap<Character, String> codeMap` - HashMap с кодами символов, полученными в результате работы алгоритма
2. Вызывается приватный метод `private void readFileAndCalculateCharacterRate(String filePath)`. В данном методе происходит обход входного текстового файла и происходит подсчёт частот символов. Для хранения частот был выбран HashMap, так как необходимые операции в нём выполняются в среднем за O(1).
3. Вызывается приватный метод `private void buildHuffmanTree()`. В данном методе происходит сортировка символов по их частотам и построение дерева Хаффмана на основе этих данных. Для сортировки используется `PriorityQueue<Node>`. Для хранения узлов дерева Хаффмана используется внутренний класс `Node`, в каждом узле содержатся символ, его частота, левые и правый дочерние узлы. Вершина дерева - узел `Node root`.

4. Вызывается приватный метод `private void fillCodeTable(Node node, String code)`. В данном методе заполняется таблица, в которой каждому символу ставится в соответствие её код Хаффмана. Таблица представлена в виде HashMap (структура выбрана из-за того, что необходимые операции в ней выполняются в среднем за O(1)). Алгоритм заполнения таблицы реализован рекурсивно, обход идёт от вершины дерева к его листьям, в методе для каждого узла происходят рекурсивные подвызовы данного метода для его дочерних узлов.
5. Вызывается приватный метод `private void writeCodeFile(String filePath, String codeFilePath)`. В методе происходит проход по входному текстовому файлу, символы которого записывается в выходной текстовый файл в закодированном виде, коды берутся из `codeMap`. В выходной текстовый файл также записывается таблица символов (необходима для декодирования), это происходит до записи непосредственно самого кода.

Декодировщик

Описание работы.

1. Вызывается приватный метод `private void readCodeFile(String codeFilePath, String filePath)`. Работа метода описана в последующих пунктах.
2. Происходит считывание файла, хранящего таблицу символов и код Хаффмана. Изначально считывается таблица символов и сохраняется в `HashMap<String, Character> charactersMap` (ключи - коды, значения - символы, которым сопоставлены эти коды).
3. Происходит считывание самого кода Хаффмана. Считанная на предыдущем шаге таблица ставит в соответствие коду определенный символ, который записывается в файл с декодированным текстом. Таким образом, происходит полное декодирование кода Хаффмана.

Арифметическое кодирование

Алгоритм арифметического кодирования реализован в классе `ArithmeticCoder`. Класс предоставляет два публичных метода для работы с файлами.

Метод-кодировщик. Принимает 2 аргумента: `String filePath` - путь до текстового файла с кодировкой UTF-8 и `String codeFilePath` - путь до текстового файла, в который будем записан полученный код.

```
public void encode(String filePath, String codeFilePath)
```

Метод-декодировщик. Принимает 2 аргумента: `String codeFilePath` - путь до текстового файла, хранящего арифметический код, `String filePath` - путь до текстового файла, в который будет декодирован зашифрованный ранее код.

```
public void decode(String codeFilePath, String filePath)
```

Кодировщик

Описание работы.

1. Очищаются метаданные, хранящиеся в классе, которые могли быть заполнены при предыдущих обращениях к методам `encode` и `decode`. К этим метаданным относятся `HashMap<Character, Long> characterRateMap` - HashMap с частотами символов в кодируемом тексте, `HashMap<Character, ArrayList<Long>> segmentsMap` - HashMap, в котором каждому уникальному символу ставится в соответствие лист, хранящий 2 элемента - концы отрезка, длина которого соответствует частоте символа, `Long numberOfCharacters` - количество символов в тексте.
2. Вызывается приватный метод `private void readFileAndCalculateCharacterRate(String filePath)`. В данном методе происходит обход входного текстового файла и происходит подсчёт частот символов. Для хранения частот был выбран HashMap, так как необходимые операции в нём выполняются в среднем за O(1).
3. Вызывается приватный метод `private void buildSegments()`. В данном методе происходит построение таблицы отрезков. Заполняется HashMap, в котором каждому уникальному символу ставится в соответствие лист, хранящий 2 элемента - концы отрезка, длина которого соответствует частоте символа (структура выбрана из-за того, что необходимые операции в ней выполняются в среднем за O(1)).
4. Вызывается приватный метод `private void writeCodeFile(String filePath, String codeFilePath)`. В методе происходит проход по входному текстовому файлу, символы которого записывается в выходной текстовый файл в закодированном виде. В выходной текстовый файл также записывается таблица отрезков (необходима для декодирования, достаточно только левых концов) и количество символов в исходном тексте, это происходит до записи непосредственно самого кода. Запись кода. Данная реализация арифметического кодирования основывается на оперировании с целыми числами (`Long`), так как арифметика чисел с плавающей запятой работает медленно, и при этом происходит потеря точности. Реализация происходит при помощи двух целых переменных start и end, они имеют длину в 18 десятичных

цифр. Эти переменные хранят границы текущего подынтервала, но алгоритм не позволяет им неограниченно расти. Как только самые левые цифры переменных start и end становятся одинаковыми, они уже не меняются в дальнейшем. Поэтому эти числа выдвигаются за пределы переменных start и end и сохраняются в выходном файле в двоичном представлении. Таким образом, эти переменные хранят не весь код, а только самую последнюю его часть. После сдвига цифр справа дописывается 0 в переменную start, а в переменную end цифра 9.

Декодировщик

Описание работы.

1. Вызывается приватный метод `private void readCodeFile(String codeFilePath, String filePath)`. Работа метода описана в последующих пунктах.
2. Происходит считывание файла, хранящего таблицу отрезков (левых концов) и арифметический код. Изначально считывается таблица отрезков (левых концов) и сохраняется в `ArrayList<Character> characters` (сами символы) и в `ArrayList<Long> points` (левые концы отрезков). Индексы символов и соответствующих им точек совпадают. Также в `ArrayList<Long> points` записывается количество всех символов в тексте (правый конец).
3. Далее декодируется сам арифметический код.
   Описание одной итерации декодирования.
   1. Считывается из кода число `long frame` (записывается уже в десятичном представлении), размер - 18 символов.
   2. Вычиляется число `long index = (long) ((frame - start) * ((double) numberOfCharacters / (end - start + 1)) - ((double) 1 / (end - start + 1)))` - точка на соответствующем интервале.
   3. При помощи бинарного поиска с ключом `index` определяется нужный интервал. И соответственно, определяется символ, который записывается в текстовый файл с декодированным текстом.
   4. Пересчитываются значения start и end, так как происходит переход к подынтервалу.
   5. Если левые части start и end совпадают, то происходит сдвиг этих чисел влево. После сдвига цифр справа дописывается 0 в переменную start, а в переменную end цифра 9.
   6. Считывается новое значение `long frame`, осуществляется переход к следующей итерации.

Алгоритм преобразования Барроуза — Уилера с последующим сжатием алгоритмом "Стопки книг"

Алгоритм BWT реализован в классе `BurrowsWheelerTransformCoder`. Класс предоставляет два публичных метода для работы с файлами.

Метод-кодировщик. Принимает 2 аргумента: `String filePath` - путь до текстового файла с кодировкой UTF-8 и `String codeFilePath` - путь до текстового файла, в который будем записан полученный код.

```
public void encode(String filePath, String codeFilePath)
```

Метод-декодировщик. Принимает 2 аргумента: `String codeFilePath` - путь до текстового файла, хранящего код, `String filePath` - путь до текстового файла, в который будет декодирован зашифрованный ранее код.

```
public void decode(String codeFilePath, String filePath)
```

Кодировщик

Описание работы.

1. Очищаются метаданные, хранящиеся в классе, которые могли быть заполнены при предыдущих обращениях к методам `encode` и `decode`. К этим метаданным относятся `ArrayList<String> rotations` - лист с перестановками строк, `HashSet<Character> alphabet` - сет уникальных символов текста, `ArrayList<Character> tempCode` - лист символов временного кода (до применения MTF).
2. Вызывается приватный метод `private void readFileAndWriteCodeFile(String filePath, String codeFilePath))`. Работа метода описана в последующих пунктах.
3. Исходный текст кодируется порциями по 1000 символов (последняя порция может быть по размеру короче 1000 символов). Данный атрибут хранится в константе `private static final int PIECE_SIZE = 1000`. Далее описана работа с каждой порцией.

4. Вызывается приватный метод `private void fillPieceMetadata(StringBuilder piece)`. В нём происходит заполнение листа с перестановками строк (матрица BWT), заполнение сета уникальных символов (алфавита), сохранение временного кода, полученного в ходе работы BWT, в лист `tempCode`.
5. В выходной текстовый файл записывается алфавит для порции и индекс исходной строки в матрице BWT.
6. Вызывается приватный метод `encodeWithMoveToFront()`. Данный метод преобразует временный код, полученный после выполнения алгоритма BTW, в итоговый код. На этом этапе применяется алгоритм "Стопки книг" (MTF). Код записывается в выходной текстовый файл.
7. Далее происходит очистка метаданных и переход к кодированию следующей порции текста.

Декодировщик

Описание работы.

1. Очищаются метаданные, хранящиеся в классе, которые могли быть заполнены при предыдущих обращениях к методам `encode` и `decode`.
2. Вызывается приватный метод `private void readCodeFile(String codeFilePath, String filePath)`. Работа метода описана в последующих пунктах.
3. Считывание и декодирование кода так же происходит порциями. В текстовый файл с раскодированным текстом на каждой итерации будет записано 1000 символов (на последнем шаге может быть записано менее 1000 символов). Далее описана работа с каждой порцией.
4. Происходит считывание алфавита, который сохраняется в `HashSet<Character> alphabet`, и индекса исходной строки в матрице BWT, который сохраняется в переменную `Integer keyIndex`.
5. Происходит считывание кода и его декодирование. Сначала выполняется декодирование при помощи обратного преобразования MTF, далее выполняется декодирование при помощи обратного преобразования BWT (оптимизированный алгоритм, использующий вектор обратного преобразования). Полученные символы записываются в файл с декодированным текстом.
6. Далее происходит очистка метаданных и переход к кодированию следующей порции текста.

Код Хэмминга (7, 4)

Алгоритм кодирования Хэмминга реализован в классе `HammingCoder`. Класс предоставляет два публичных метода для работы с файлами.

Метод-кодировщик. Принимает 2 аргумента: `String filePath` - путь до текстового файла с кодировкой UTF-8 и `String codeFilePath` - путь до текстового файла, в который будем записан полученный код.

```
public void encode(String filePath, String codeFilePath)
```

Метод-декодировщик. Принимает 2 аргумента: `String codeFilePath` - путь до текстового файла, хранящего код Хэмминга, `String filePath` - путь до текстового файла, в который будет декодирован зашифрованный ранее код.

```
public void decode(String codeFilePath, String filePath)
```

Кодировщик

Описание работы.

1. Вызывается приватный метод `private void readFileAndWriteCodeFile(String filePath, String codeFilePath))`. Работа метода описана в последующих пунктах.
2. Происходит чтение текста исходного файла и перевод символов в массив байт.
3. Вызывается приватный метод `private String encodeBytesArray(byte[] bytes)`. Данный метод переводит массив байт в массив битов. Далее к каждым 4 битам приписываются по 3 контрольных бита.
4. Полученный код Хэмминга записывается в выходной текстовый файл.

Декодировщик

Описание работы.

1. Вызывается приватный метод `private void readCodeFile(String codeFilePath, String filePath)`. Работа метода описана в последующих пунктах.
2. Код Хэмминга считывается порциями по 14 бит. Из каждой порции восстанавлиется 1 байт исходного текстового файла. Вызывается приватный метод `private byte decodeByte(String code)`. В данном методе выполняется алгоритм декодирования по Хэммингу, который идентичен алгоритму кодирования. В ходе декодирования исправляются одиночные ошибки.
3. Получаемый массив байт преобразуется в символы исходного текстового файла. Символы записываются в файл с декодированным текстом.

Консольное приложение

Описание работы

Консольное приложение может выполнять следующие команды:

- `huffman encode` - закодировать файл алгоритмом Хаффмана
- `huffman decode` - декодировать файл алгоритмом Хаффмана
- `arithmetic encode` - закодировать файл алгоритмом Арифметического кодирования
- `arithmetic decode` - декодировать файл алгоритмом Арифметического кодирования
- `bwt encode` - закодировать файл алгоритмом преобразования Барроуза — Уилера
- `bwt decode` - декодировать файл алгоритмом преобразования Барроуза — Уилера
- `hamming encode` - закодировать файл алгоритмом Хэмминга
- `hamming decode` - декодировать файл алгоритмом Хэмминга
- `compare` - сравнить содержания текстовых файлов
- `help` - список доступных команд
- `exit` - выход из программы

При выполнении команд по кодированию или декодированию приложение требует передавать пути до текстовых файлов в абсолютном виде.

Для наглядоности ниже продемонстрирован пример работы с консольным приложением:

```
Введите команду (справка - команда help):
huffman decode
Введите абсолютный путь до текстового файла, подлежащего декодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HaffmanCode.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён
расшифрованный текст:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HaffmanDecodedText.txt

Выполнено
Введите команду:
compare
Введите абсолютный путь первого текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HaffmanDecodedText.txt

Введите абсолютный путь второго текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
```

```
Содержания файлов одинаковы
Введите команду:
arithmetic encode
Введите абсолютный путь до текстового файла, подлежащего кодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён код:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\ArithmeticCode.txt
Выполнено
Введите команду:
arithmetic decode
Введите абсолютный путь до текстового файла, подлежащего декодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\ArithmeticCode.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён
расшифрованный текст:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\ArithmeticDecodedText.t

Выполнено
Введите команду:
compare
Введите абсолютный путь первого текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\ArithmeticDecodedText.t

Введите абсолютный путь второго текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Содержания файлов одинаковы
Введите команду:
bwt encode
Введите абсолютный путь до текстового файла, подлежащего кодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён код:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\BurrowsWheelerTransforn

Выполнено
Введите команду:
bwt decode
Введите абсолютный путь до текстового файла, подлежащего декодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\BurrowsWheelerTransforn

Введите абсолютный путь до текстового файла, в котором будет сохранён
расшифрованный текст:
```

```
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\BurrowsWheelerTransforr

Выполнено
Введите команду:
compare
Введите абсолютный путь первого текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\BurrowsWheelerTransforr

Введите абсолютный путь второго текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Содержания файлов одинаковы
Введите команду:
hamming encode
Введите абсолютный путь до текстового файла, подлежащего кодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён код:

C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HammingCode.txt
Выполнено
Введите команду:
hamming decode
Введите абсолютный путь до текстового файла, подлежащего декодированию:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HammingCode.txt
Введите абсолютный путь до текстового файла, в котором будет сохранён
расшифрованный текст:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HammingDecodedText.txt

Выполнено
Введите команду:
compare
Введите абсолютный путь первого текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\HammingDecodedText.txt

Введите абсолютный путь второго текстового файла:
C:\Users\Mi\Desktop\Rishat19\ТИК\Coding\src\main\resources\WarAndPeace.txt
Содержания файлов одинаковы
Введите команду:
help
Список доступных команд:
huffman encode - закодировать файл алгоритмом Хаффмана
huffman decode - декодировать файл алгоритмом Хаффмана
```

```
arithmetic encode - закодировать файл алгоритмом Арифметического кодирования
arithmetic decode - декодировать файл алгоритмом Арифметического кодирования
bwt encode - закодировать файл алгоритмом преобразования Барроуза — Уилера
bwt decode - декодировать файл алгоритмом преобразования Барроуза — Уилера
hamming encode - закодировать файл алгоритмом Хэмминга
hamming decode - декодировать файл алгоритмом Хэмминга
compare - сравнить содержания текстовых файлов
help - список доступных команд
exit - выход из программы
Введите команду:
exit

Process finished with exit code 0
```