Структура дерева кодирования

1. Алгоритм Хаффмана

Кодирование:

- Алгоритм строит дерево Хаффмана на основе частот встречаемости символов в исходном файле.
- Каждый узел дерева хранит в себе символ и частоту его встречаемости.
- Дерево строится с использованием приоритетной очереди.
- Листья дерева содержат символы, внутренние узлы содержат суммарные частоты.

Декодирование:

- Дерево Хаффмана используется для декодирования битового потока, представляющего закодированный файл.
- По мере прохождения битового потока, восстанавливается исходная последовательность символов.

2. Классы и структуры данных

Node (Узел дерева):

- Содержит в себе символ, частоту его встречаемости и ссылки на левого и правого потомка.
- Используется для построения дерева Хаффмана.

TreePriorityQueue:

- Очередь с приоритетом для узлов дерева, которая автоматически сортируется так, чтобы элементы с наименьшим приоритетом (с наименьшей частотой) находились в начале.

HuffmanTree:

- Содержит методы для построения дерева Хаффмана, создания кодов Хаффмана для символов и сохранения/чтения дерева в/из битового потока.

BitInputStream и **BitOutputStream**:

- Реализуют чтение и запись битов в поток ввода и вывода соответственно.
- Используют FileInputStream и FileOutputStream для чтения и записи байтов в файл.

HuffmanCoder:

- Содержит методы для кодирования, декодирования и информирования о файлах.
- Использует классы HuffmanTree, BitInputStream и BitOutputStream.

Main:

- Основной класс для взаимодействия с пользователем.
- Запрашивает входные файлы и выбор операции (кодирование, декодирование, информирование).

3. Основные алгоритмы:

1) Инициализация приоритетной очереди:

- В классе **HuffmanTree**, метод **buildHuffmanTree** принимает карту частот **frequencyMap** символов и создаёт приоритетную очередь **TreePriorityQueue**.
- Для каждой пары (символ, частота) создаётся узел дерева и добавляется в очередь с приоритетом.

2) Построение дерева:

Пока в приоритетной очереди не останется только один узел (корень) выполняется следующее:

- Извлекаются два узла с наименьшей частотой из приоритетной очереди.
- Создается новый внутренний узел с суммарной частотой этих двух узлов, и эти узлы становятся его потомками.
- Новый внутренний узел добавляется в приоритетную очередь.

В результате в приоритетной очереди остаётся только один узел - корень дерева.

3) Создание кодов Хаффмана:

- После построения дерева вызывается метод **buildHuffmanCodes**.
- Этот метод рекурсивно проходит по дереву, начиная с корня.
- При проходе влево добавляется 0 к текущему коду, при проходе вправо добавляется 1.
- Когда достигается лист дерева (символ), код Хаффмана для этого символа сохраняется в карте кодов.

4) Сохранение дерева:

- Метод **saveTree** класса **HuffmanTree** используется для сохранения дерева в битовом потоке.
- Каждый внутренний узел помечается битом 0, а листовой узел – битом 1.
- Для листового узла также записывается 8 бит символа.

5) Чтение дерева:

- Метод **readTree** класса **HuffmanTree** используется для чтения дерева из битового потока.
- Сначала читается бит, и если он равен 1, то читается 8 бит, представляющих символ листового узла.
- Если бит равен 0, то рекурсивно вызываются **readTree** для левого и правого поддеревьев.