**Описание структуры программы**

**1. Структура программы**

Программа реализует кодирование и декодирование файлов методом Хаффмана с управлением через командную строку.

**Основные компоненты:**

1. \*\*Класс \*\***HuffmanNode**
   * Узел дерева Хаффмана, содержащий символ, частоту и ссылки на дочерние узлы.
2. **Методы построения дерева**
   * buildTree(Map<Character, Integer>) — строит дерево Хаффмана из частотного словаря.
   * generateCodes(HuffmanNode, String) — генерирует таблицу кодов Хаффмана, рекурсивно обходя дерево.
3. **Кодирование и декодирование**
   * encode(String) — преобразует текст в закодированную битовую строку.
   * decode(String) — декодирует битовую строку обратно в текст.
4. **Взаимодействие с файлами**
   * main(String[] args) — обработка аргументов командной строки:
     + encode <input> <output> — кодирует файл и сохраняет результат.
     + decode <input> <output> — декодирует файл обратно в текст.
   * Используется сериализация (ObjectOutputStream / ObjectInputStream) для сохранения словаря и битовой последовательности.

**2. Описание структуры закодированного файла**

**Формат хранения данных:**

Закодированный файл состоит из двух частей:

1. **Словарь Хаффмана** — содержит соответствие символов их бинарным кодам.
2. **Закодированные данные** — последовательность битов, представляющих входной файл.

**1. Словарь Хаффмана:**

Словарь хранится в виде сериализованного объекта Java (ObjectOutputStream). Формат:

* Map<Character, String> — ключом является символ, значением — его бинарный код.
* Записывается с использованием writeObject().

Пример (текст "1111111111"):

{'1': "0"}

Пример (текст "11111111112222233333"):

{'1': "0", '2': "10", '3': "11"}

**2. Закодированные данные:**

* Представляют собой строку бит (String), содержащую закодированное представление исходного файла.
* Также сохраняется в сериализованном виде.

Пример (для текста "11111111112222233333" с кодами выше):

0000000000101010101011111111

**Пример структуры закодированного файла:**

Файл encoded.huff содержит:

1. Словарь:

{'1': "0", '2': "10", '3': "11"}

1. Закодированные данные:

0000000000101010101011111111

**Декодирование:**

1. Загружается словарь Хаффмана из файла.
2. Читается последовательность бит.
3. Последовательность бит заменяется на исходные символы с использованием словаря.