

|  |  |
| --- | --- |
| **Министерство образования и науки**  **Российской Федерации**  **Государственное образовательное учреждение высшего образования «МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ**  **ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **(МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Отчет по дисциплине «Структуры и алгоритмы обработки данных»**

**Практическая работа №5**

**«Коды Хаффмана»**

**Выполнил:**

Учебная группа 1бПМ\_\_\_\_\_\_\_\_\_  
ФИО Греча К. П. \_\_\_\_\_

**Принял:**

Должность Старший преподаватель\_

Звание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО \_ Кутейников И. А.\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г.

Цель и задачи:

Написать программу, реализующую кодирование символов алфавита

входного текстового файла в виде двоичных кодов:

- фиксированной длины;

- переменной длины.

Для генерации кодов переменной длины использовать жадный

алгоритм Хаффмана (код Хаффмана).

Реализовать меню с пунктами:

1 Открыть текстовый файл;

2 Вывести содержимое текстового файла;

3 Вывести символы алфавита с указанием их частоты появления с

сортировкой по частоте;

4 Сгенерировать коды для символов алфавита входного файла

4.1 Вывести алфавит входного файла с кодами фиксированной

длины для каждого символа алфавита;

4.2 Вывести алфавит входного файла с кодами Хаффмана для

каждого символа алфавита;

5 Сжать содержимое текстового файла с помощью кодов фиксированной

длины с сохранением данных в файл;

6 Сжать содержимое текстового файла с помощью кодов Хаффмана с

сохранением данных в файл.

7 Сравнить размеры файлов исходного текстового файла и двух

Зашифрованных.

Алгоритм для генерации кодов фиксированной длины:

public static Map<Character, String> generateLengthCodes(String content) {  
 Set<Character> charset = new HashSet<>();  
 for (char c : content.toCharArray()) {  
 charset.add(c);  
 }  
//вычисляется мин. кол-во бит и округляется до ближ. большего  
 int bitsPerChar = (int) Math.*ceil*(Math.*log*(charset.size()) / Math.*log*(2));  
 Map<Character, String> codes = new HashMap<>();  
 int counter = 0;  
  
 for (char c : charset) {  
 String binaryString = Integer.*toBinaryString*(counter); //получаем двоичное значение  
 //форматируем строку через битсПерЧар, если бинари стринг меньше то метод добавит пробелы  
 binaryString = String.*format*("%" + bitsPerChar + "s", binaryString).replace(' ', '0');//реплейс нужен для замены пробелов нулями  
 codes.put(c, binaryString); //добавляем в мап символ и значение  
 counter++;  
 }  
 return codes;  
 }  
  
 public static String encodeInf(Map<Character, String> codes, String content) {  
 StringBuilder encoded = new StringBuilder();  
 for (char c : content.toCharArray()) {  
 encoded.append(codes.get(c)); //строим закодированную строку  
 }  
 return encoded.toString();  
 }  
  
 public static void saveToFile(String encodedInf, String outputPath)  
 throws IOException {  
 BitSet bitSet = new BitSet(encodedInf.length()); //создаем битсет и храним каждый символ из энкодидсонтент  
 int bitCounter = 0;  
 for (char c : encodedInf.toCharArray()) { //по умолч. все биты уст. на фолс  
 if (c == '1') {  
 bitSet.set(bitCounter);//уст. соответствующий бит на тру  
 }  
 bitCounter++; //отслеживаем позицию бита  
 }  
 try (FileOutputStream outputStream = new FileOutputStream(outputPath)) { //записываем в файл по пути оутпутПат  
 outputStream.write(bitSet.toByteArray());  
 }  
 }

Методы для генерации кодов Хаффмана:

private static Map<Character, String> generateHuffmanCodes(String inf) {  
 // Подсчитываем частоты символов  
 Map<Character, Integer> freqMap = new HashMap<>();  
 for (char c : inf.toCharArray()) {  
 freqMap.put(c, freqMap.getOrDefault(c, 0) + 1);  
 }  
 // cоздаем приоритетную очередь и добавляем туда листья для каждого символа  
 PriorityQueue<HuffmanNode> queue = new PriorityQueue<>();  
 for (Map.Entry<Character, Integer> entry : freqMap.entrySet()) {  
 queue.add(new HuffmanNode(entry.getKey(), entry.getValue()));  
 }  
 // cтроим дерево Хаффмана  
 while (queue.size() > 1) {  
 HuffmanNode left = queue.poll(); //извлечение двух узлов с наименьшей частотой  
 HuffmanNode right = queue.poll();//наименьший приоретет  
 //создание род. дерева  
 HuffmanNode parent = new HuffmanNode('X', left.frequency + right.frequency);  
 parent.left = left;  
 parent.right = right;  
 queue.add(parent); //добавляем родительский узел обратно в очередь  
 }  
  
 HuffmanNode node = queue.poll();// Генерируем коды для символов  
 Map<Character, String> codes = new HashMap<>();  
 *generateCodes*(node, "", codes);  
  
 return codes;  
}  
  
private static void generateCodes(HuffmanNode node, String code, Map<Character, String> codes) {  
 if (node != null) {  
 if (node.left == null && node.right == null) {  
 codes.put(node.character, code); //если файл листовый то добавляем его бинарный путь в кодс  
 }  
 *generateCodes*(node.left, code + "0", codes);  
 *generateCodes*(node.right, code + "1", codes);  
 }  
}  
private static String encodeInfWithHuffmanCodes(Map<Character, String> huffmanCodes, String content) {  
 StringBuilder encoded = new StringBuilder();  
 for (char c : content.toCharArray()) {  
 encoded.append(huffmanCodes.get(c)); //также строим строку  
 }  
 return encoded.toString();  
}

class HuffmanNode implements Comparable<HuffmanNode> {  
 char character;  
 int frequency;  
 HuffmanNode left, right;  
  
 HuffmanNode(char character, int frequency) {  
 this.character = character;  
 this.frequency = frequency;  
 }  
  
 // Сравнение для приоритетной очереди основано на частоте появления символа.  
 @Override  
 public int compareTo(HuffmanNode other) {  
 return Integer.*compare*(this.frequency, other.frequency);  
 }  
}

Методы для сортировки по частоте:

private static Map<Character, Integer> getFrequencyMap(String content) {  
 Map<Character, Integer> freqMap = new HashMap<>();  
 for (char ch : content.toCharArray()) {  
 int z=freqMap.getOrDefault(ch, 0); //если мап содержит счетчик вернет его частоту, если его нет то 0  
 freqMap.put(ch, z + 1); //символу сопоставляется его частота появления  
 }  
 return freqMap;  
}  
  
private static List<Map.Entry<Character, Integer>> sortFrequencyMap(Map<Character, Integer> freqMap) {  
 //в конструкторе передаем все сеты мап-пары ключ-значения  
 List<Map.Entry<Character, Integer>> sortedList = new ArrayList<>(freqMap.entrySet());  
 sortedList.sort(Map.Entry.<Character, Integer>*comparingByValue*().reversed()); //компаратор сортирует по частоте  
 return sortedList;  
}  
  
private static void printFrequencyList(List<Map.Entry<Character, Integer>> list) {  
 for (Map.Entry<Character, Integer> entry : list) {  
 System.*out*.println(entry.getKey() + " : " + entry.getValue());  
 }  
}

**Результат:**

**Ввод:**

1. Открыть текстовый файл

2. Вывести содержимое текстового файла

3. Вывести символы алфавита с указанием их частоты появления с сортировкой по частоте;

4. Сгенерировать коды для символов алфавита входного файла

5. Сжать содержимое текстового файла с помощью кодов фиксированной длины с сохранением данных в файл

6. Сжать содержимое текстового файла с помощью кодов Хаффмана с сохранением данных в файл

7. Сравнить размеры файлов исходного текстового файла и двух зашифрованных

Выберите пункт меню: 1

Введите путь к файлу:

**C:/Intellij Idea/Lab5CIAOD/lab4.txt**

Файл успешно открыт: lab4.txt

Выберите пункт меню: 2

Содержимое файла lab4.txt:

Кроссовки, 2, 50

Ручка, 1, 2

Очки, 2, 15

Форма, 4, 20

Кошелек, 1, 35

Ключи, 5, 10

**Вывод:**

1. Открыть текстовый файл

2. Вывести содержимое текстового файла

3. Вывести символы алфавита с указанием их частоты появления с сортировкой по частоте;

4. Сгенерировать коды для символов алфавита входного файла

5. Сжать содержимое текстового файла с помощью кодов фиксированной длины с сохранением данных в файл

6. Сжать содержимое текстового файла с помощью кодов Хаффмана с сохранением данных в файл

7. Сравнить размеры файлов исходного текстового файла и двух зашифрованных

Выберите пункт меню: 5

Файл успешно закодирован и сохранен под именем 'encoded\_file.bin'

Закодированный текст (фиксированная длина) до 80 символов:

**0100000000110110000100001110111000111000101110110101010100100110101010101010111000110001...**

1. Открыть текстовый файл

2. Вывести содержимое текстового файла

3. Вывести символы алфавита с указанием их частоты появления с сортировкой по частоте;

4. Сгенерировать коды для символов алфавита входного файла

5. Сжать содержимое текстового файла с помощью кодов фиксированной длины с сохранением данных в файл

6. Сжать содержимое текстового файла с помощью кодов Хаффмана с сохранением данных в файл

7. Сравнить размеры файлов исходного текстового файла и двух зашифрованных

Выберите пункт меню: 6

Файл успешно закодирован под именем 'encoded\_file\_huffman.bin'

Закодированный текст (код Хаффмана) до 80 символов:

**1101111111011110001010010111110001111111001100010110011101101100000111001011001110000110...**

1. Открыть текстовый файл

2. Вывести содержимое текстового файла

3. Вывести символы алфавита с указанием их частоты появления с сортировкой по частоте;

4. Сгенерировать коды для символов алфавита входного файла

5. Сжать содержимое текстового файла с помощью кодов фиксированной длины с сохранением данных в файл

6. Сжать содержимое текстового файла с помощью кодов Хаффмана с сохранением данных в файл

7. Сравнить размеры файлов исходного текстового файла и двух зашифрованных

Выберите пункт меню: 7

Размер исходного файла: **121 байт**

Размер файла с фиксированной длиной кода: **54 байт**

Размер файла с кодом Хаффмана: **48 байт**

Разница между размером исходного файла с файлом фикс. длины **67 байт**

Разница между размером исходного файла с файлом с кодом Хаффмана **73 байт**

**Заключение:**

В процессе выполнения лабораторной работы было выполненно кодирование кодами фиксированной длины и кодами Хаффмана. Также было проведенно сравнение размером закодированных и исходного файла.