

|  |  |
| --- | --- |
| **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»**

**Лабораторная работа №5**

**«Коды Хаффмана»**

**Выполнил:**

Учебная группа 1бПМ\_\_\_\_\_\_\_\_\_  
ФИО Кузнецов А-С.О.\_\_\_\_\_

**Принял:**

Должность \_\_\_\_\_

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

ФИО \_ Кутейников И.А.\_\_\_\_\_\_

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

Москва 2024

Цель:

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.

Результаты лабораторной работы оформить в виде отчета с результатами

работы программы.

Код:

…

private static void openTextFile() {  
 try {  
 BufferedReader br = new BufferedReader(new FileReader("input.txt"));  
 String line;  
 while ((line = br.readLine()) != null) {  
 for (char c : line.toCharArray()) {  
 if (Character.*isLetter*(c)) {  
 *frequencyMap*.put(c, *frequencyMap*.getOrDefault(c, 0) + 1);  
 }  
 }  
 }  
 br.close();  
 System.*out*.println("Текстовый файл успешно открыт");  
 } catch (IOException e) {  
 System.*out*.println("Ошибка");  
 }  
 }  
  
 private static void displayTextFileContent() {  
 try {  
 BufferedReader br = new BufferedReader(new FileReader("input.txt"));  
 String line;  
 while ((line = br.readLine()) != null) {  
 System.*out*.println(line);  
 }  
 br.close();  
 } catch (IOException e) {  
 System.*out*.println("Ошибка");  
 }  
 }  
  
 private static void displayAlphabetWithFrequencies() {  
 *sortedFrequencyMap*.clear();  
 *frequencyMap*.entrySet().stream().sorted(Map.Entry.*comparingByValue*(Comparator.*reverseOrder*())).forEach(entry -> *sortedFrequencyMap*.put(entry.getKey(), entry.getValue()));  
 *sortedFrequencyMap*.forEach((key, value) -> System.*out*.println(key + " - " + value));  
 }  
  
 private static void generateCodesForAlphabet() {  
 *fixedLengthCodes*.clear();  
 *huffmanCodes*.clear();  
 // Генерация кодов фиксированной длины  
 int code = 0;  
 for (char c : *frequencyMap*.keySet()) {  
 *fixedLengthCodes*.put(c, String.*format*("%03d", Integer.*parseInt*(Integer.*toBinaryString*(code))));  
 code++;  
 }  
  
 // Генерация кодов Хаффмана  
 List<Node> nodes = new ArrayList<>();  
 for (char c : *frequencyMap*.keySet()) {  
 nodes.add(new Node(c, *frequencyMap*.get(c)));  
 }  
  
 PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.*comparingInt*(n -> n.frequency));  
 priorityQueue.addAll(nodes);  
  
 while (priorityQueue.size() > 1) {  
 Node left = priorityQueue.poll();  
 Node right = priorityQueue.poll();  
 Node parent = new Node('X', left.frequency + right.frequency);  
 parent.left = left;  
 parent.right = right;  
 priorityQueue.add(parent);  
 }  
  
 Node root = priorityQueue.poll();  
 *generateHuffmanCodes*(root, "");  
 }  
  
 private static void generateHuffmanCodes(Node node, String code) {  
 if (node.left == null && node.right == null) {  
 *huffmanCodes*.put(node.character, code);  
 return;  
 }  
  
 *generateHuffmanCodes*(node.left, code + "0");  
 *generateHuffmanCodes*(node.right, code + "1");  
 }  
  
 private static void displayAlphabetWithFixedLengthCodes() {  
 *fixedLengthCodes*.forEach((key, value) -> System.*out*.println(key + " - " + value));  
 }  
  
 private static void displayAlphabetWithHuffmanCodes() {  
 *huffmanCodes*.forEach((key, value) -> System.*out*.println(key + " - " + value));  
 }  
  
 private static void compressWithFixedLengthCodes() {  
 try {  
 BufferedReader br = new BufferedReader(new FileReader("input.txt"));  
 BufferedWriter bw = new BufferedWriter(new FileWriter("compressedfixedlength.bin"));  
 String line;  
 while ((line = br.readLine()) != null) {  
 for (char c : line.toCharArray()) {  
 if (*fixedLengthCodes*.containsKey(c)) {  
 bw.write(*fixedLengthCodes*.get(c));  
 }  
 }  
 }  
 br.close();  
 bw.close();  
 System.*out*.println("Файл сжат с кодами фиксированной длины");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void compressWithHuffmanCodes() {  
 try {  
 BufferedReader br = new BufferedReader(new FileReader("input.txt"));  
 BufferedWriter bw = new BufferedWriter(new FileWriter("compressedhuffman.bin"));  
 String line;  
 while ((line = br.readLine()) != null) {  
 for (char c : line.toCharArray()) {  
 if (*huffmanCodes*.containsKey(c)) {  
 bw.write(*huffmanCodes*.get(c));  
 }  
 }  
 }  
 br.close();  
 bw.close();  
 System.*out*.println("Файл сжат с кодами Хаффмана");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void compareFileSizes() {  
 File inputFile = new File("input.txt");  
 File fixedLengthFile = new File("compressedfixedlength.bin");  
 File huffmanFile = new File("compressedhuffman.bin");  
  
 long inputSize = inputFile.length();  
 long fixedLengthSize = fixedLengthFile.length();  
 long huffmanSize = huffmanFile.length();  
  
 System.*out*.println("Размер исходного файла: " + inputSize\*8 + " байт");  
 System.*out*.println("Размер сжатого файла с кодами фиксированной длины: " + fixedLengthSize + " байт");  
 System.*out*.println("Размер сжатого файла с кодами Хаффмана: " + huffmanSize + " байт");  
 }  
  
 static class Node {  
 char character;  
 int frequency;  
 Node left;  
 Node right;  
 Node(char character, int frequency) {  
 this.character = character;  
 this.frequency = frequency;  
 }  
 }  
}

Результат:

Меню:

1. Открыть текстовый файл

2. Показать содержимое текстового файла

3. Показать алфавитные символы с их частотами

4. Сгенерировать коды для алфавитных символов

4.1 Показать алфавит с кодами фиксированной длины

4.2 Показать алфавит с кодами Хаффмана

5. Сжать текстовый файл с кодами фиксированной длины

6. Сжать текстовый файл с кодами Хаффмана

7. Сравнить размеры файлов

8. Выход

Введите ваш выбор: 1

Текстовый файл успешно открыт

Введите ваш выбор: 2

AAABBBCCCAAABBB…

Введите ваш выбор: 3

A - 3009

B - 3009

C - 3009

V - 53

v - 4

Введите ваш выбор: 4.1

A - 000

B - 001

C - 010

V - 011

v - 100

Введите ваш выбор: 4.2

A - 01

B - 11

C - 10

v - 000

V - 001

Введите ваш выбор: 5

Файл сжат с кодами фиксированной длины

Введите ваш выбор: 6

Файл сжат с кодами Хаффмана

Введите ваш выбор: 7

Размер исходного файла: 72672 байт

Размер сжатого файла с кодами фиксированной длины: 27252 байт

Размер сжатого файла с кодами Хаффмана: 18225 байт

Вывод:

В ходе данной лабораторной работе мы познакомились с методами сжатия файлов Хаффмана и при помощи кодов фиксированной длины. Появилось понимание того, как оптимизировать свой код, а именно как облегчить вес файла. Также появилось лучшее понимание синтаксиса языка.