|  |
| --- |
| Университет итмо, кафедра вт |
| Лабораторная работа №2 по дисциплине «Прикладная математика» |
| «Построение оптимальных кодов» |
| Группа Р3302 |
| **Выполнили:**  **Орлова Кристина Александровна**  **Лалетина Екатерина Александровна** |
| **Преподаватель: Тропченко Андрей Александрович** |

|  |
| --- |
| *27.10.18* |

**Задание**

Для выполнения лабораторной работы №2 необходимо:

1. На языках С, С++ или С# реализовать процедуру построения оптимального кода Хаффмана.

2. Построить код Хаффмана для текстового файла. Требования к текстовому файлу:

• документ на английском языке

• размер текстового файла – 20 - 60кБ

• документ должен быть осмысленным

В процедуре необходимо предусмотреть возможность ввода имени файла, для которого будет строиться код, и распечатать полученную кодовую таблицу.

3. На языках С, С++ или С# реализовать процедуру построения кода Шеннона - Фано

4. Построить код Шеннона - Фано для текстового файла (требования к файлу и процедуре см. п. 2 и распечатать полученную кодовую таблицу.

5. Сравнить энтропию закодированных оптимальными кодами файлов и исходного файла в виде таблицы.

6. Проанализировать полученные результаты.

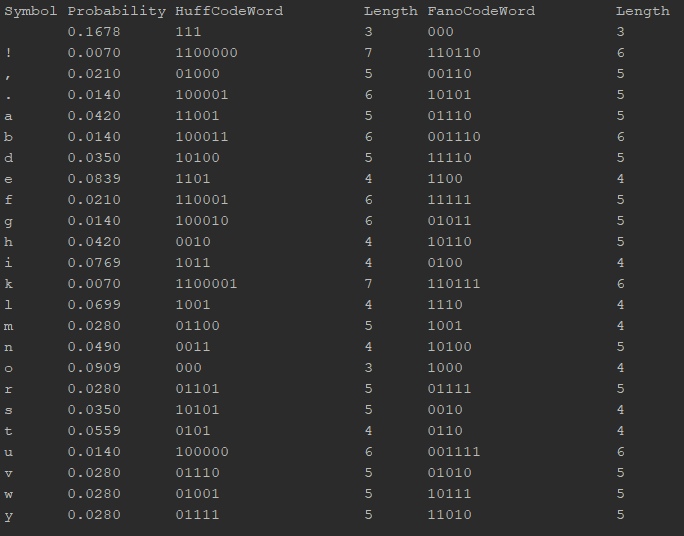
**Исходный код**

**const** fs = require("fs");  
**let** fileContent;  
**let** alphabet = {}; // объект, поля которого - символы, а их значения - объект  
**let** huffArray = {}; // объект с ключом в символ, значение которого вероятность  
**let** fanoArray = {}; // объект с ключом в символ, значение которого вероятность  
**let** realLength = 0;  
**let** separator = 'г'; // разделяющий символ (для разделения ключей в одной строке)  
  
main();  
  
**function** main() {  
 **let** stdIn = process.openStdin();  
  
 console.log('Input the filename (ex: doc.txt)\nDefault directory: text\_files');  
  
 stdIn.on('data', **function** (filename) {  
 filename = "text\_files//" + filename.toString().substr(0, filename.length - 1);  
  
 **try** {  
 readFile(filename);  
  
 console.log('Result:');  
  
 fillAdditionalInfo();  
 huffmanAlgorithm();  
 fanoAlgorithm();  
 printAlphabet();  
 printEntropy();  
 } **catch** (err) {  
 console.log('Error: no such txt file')  
 }  
  
 process.exit();  
 });  
}  
  
*/\*\*  
 \* Считывает содержимое файла и формирует по нему алфавит символов  
 \*  
 \** ***@param*** *fileName  
 \*/***function** readFile(fileName) {  
 fileContent = fs.readFileSync(fileName, 'utf8');  
 **const** textLength = fileContent.length;  
  
 **for** (**let** i = 0; i < textLength; i++) {  
 **let** symbol = fileContent[i].toLowerCase();  
  
 **if** (symbol.charCodeAt(0) >= 32 && symbol.charCodeAt(0) <= 126) {  
 **if** (alphabet[symbol] === undefined) {  
 alphabet[symbol] = 1;  
 } **else** {  
 alphabet[symbol]++;  
 }  
  
 realLength++;  
 }  
 }  
}  
  
*/\*\*  
 \* Добавляет в alphabet информацию о вероятности, кодовом слове и его длине Хаффмана и Шеннона Фано для каждого символа,  
 \* формирует huffArray, fanoArray на основе посчитанной вероятности  
 \*/***function** fillAdditionalInfo() {  
 **for** (**let** key **in** alphabet) {  
 alphabet[key] = {  
 'probability': alphabet[key] / realLength, 'huffCodeWord': '', 'huffLength': 0, 'fanoCodeWord':  
 '', 'fanoLength': 0  
 };  
 huffArray[key] = alphabet[key].probability;  
 fanoArray[key] = alphabet[key].probability;  
 }  
}  
  
*/\*\*  
 \* Считает кодовое слово и его длину алгоритмом Хаффмана  
 \*/***function** huffmanAlgorithm() {  
 **const** maxProbability = 2;  
  
 **while** (Object.keys(huffArray).length > 1) {  
 **let** firstCandidate = {'key': '', 'probability': maxProbability},  
 secondCandidate = {'key': '', 'probability': maxProbability};  
  
 // находим наименьшие  
 **for** (**let** key **in** huffArray) {  
 **if** (huffArray[key] < firstCandidate.probability || huffArray[key] < secondCandidate.probability) {  
 **if** (firstCandidate.probability === maxProbability) {  
 firstCandidate.probability = huffArray[key];  
 firstCandidate.key = key;  
 } **else if** (secondCandidate.probability === maxProbability) {  
 secondCandidate.probability = huffArray[key];  
 secondCandidate.key = key;  
 } **else if** (huffArray[key] < firstCandidate.probability && huffArray[key] < secondCandidate.probability) {  
 **if** (firstCandidate.probability > secondCandidate.probability) {  
 firstCandidate.probability = huffArray[key];  
 firstCandidate.key = key;  
 } **else** {  
 secondCandidate.probability = huffArray[key];  
 secondCandidate.key = key;  
 }  
 } **else if** (huffArray[key] < firstCandidate.probability) {  
 firstCandidate.probability = huffArray[key];  
 firstCandidate.key = key;  
 } **else if** (huffArray[key] < secondCandidate.probability) {  
 secondCandidate.probability = huffArray[key];  
 secondCandidate.key = key;  
 }  
 }  
 }  
  
 // делаем перестановку: слева меньше, справа больше  
 **if** (firstCandidate.probability > secondCandidate.probability) {  
 **const** timing = firstCandidate;  
  
 firstCandidate = secondCandidate;  
 secondCandidate = timing;  
 }  
  
 // меняем кодовое слово для тех, что в левой ветке  
 firstCandidate.key.split(separator).forEach((key) => {  
 alphabet[key].huffCodeWord = 0 + alphabet[key].huffCodeWord;  
 alphabet[key].huffLength++;  
 });  
  
 // меняем кодовое слово для тех, что в правой ветке  
 secondCandidate.key.split(separator).forEach((key) => {  
 alphabet[key].huffCodeWord = 1 + alphabet[key].huffCodeWord;  
 alphabet[key].huffLength++;  
 });  
  
 // добавляем новую ветку  
 huffArray[firstCandidate.key + separator + secondCandidate.key] = firstCandidate.probability + secondCandidate.probability;  
  
 // удаляем старые вершины  
 **delete** huffArray[firstCandidate.key];  
 **delete** huffArray[secondCandidate.key];  
 }  
}  
  
*/\*\*  
 \* Считает кодовое слово и его длину алгоритмом Шеннона Фано  
 \*/***function** fanoAlgorithm() {  
 **let** mainBunch = [fanoArray];  
  
 **while** (mainBunch.length > 0) {  
 **let** newBunch = [];  
  
 mainBunch.forEach((bunch) => {  
 **const** {firstBunchObject, secondBunchObject} = findPartition(bunch);  
  
 **if** (Object.keys(firstBunchObject).length > 1) {  
 newBunch.push(firstBunchObject);  
 }  
  
 **if** (Object.keys(secondBunchObject).length > 1) {  
 newBunch.push(secondBunchObject);  
 }  
  
 // меняем кодовое слово для элементов левой кучи  
 **for** (**let** key **in** firstBunchObject) {  
 alphabet[key].fanoCodeWord += 0;  
 alphabet[key].fanoLength += 1;  
 }  
  
 // меняем кодовое слово для элементов правой кучи  
 **for** (**let** key **in** secondBunchObject) {  
 alphabet[key].fanoCodeWord += 1;  
 alphabet[key].fanoLength += 1;  
 }  
 });  
  
 mainBunch = newBunch;  
 }  
}  
  
*/\*\*  
 \* Решение проблемы разделения на две равные части  
 \*  
 \** ***@param*** *object  
 \** ***@return*** *{{firstBunchObject: {...}, secondBunchObject: {...}}}  
 \*/***function** findPartition(object) {  
 **let** bunchObject = object;  
 **let** firstBunchObject = {'sum': 0};  
 **let** secondBunchObject = {'sum': 0};  
  
 **while** (Object.keys(bunchObject).length > 0) {  
 **let** searchedField = {'key': '', 'probability': 0};  
  
 // ищем наибольшее значение  
 **for** (**let** key **in** bunchObject) {  
 **if** (bunchObject[key] > searchedField.probability) {  
 searchedField.probability = bunchObject[key];  
 searchedField.key = key;  
 }  
 }  
  
 // кладем найденное наибольшее в кучку  
 **if** (firstBunchObject.sum === 0 || firstBunchObject.sum < secondBunchObject.sum) {  
 firstBunchObject[searchedField.key] = searchedField.probability;  
 firstBunchObject.sum += searchedField.probability;  
 } **else** {  
 secondBunchObject[searchedField.key] = searchedField.probability;  
 secondBunchObject.sum += searchedField.probability;  
 }  
  
 // удаляем поле  
 **delete** bunchObject[searchedField.key];  
 }  
  
 **delete** firstBunchObject['sum'];  
 **delete** secondBunchObject['sum'];  
  
 **return** {  
 firstBunchObject: firstBunchObject,  
 secondBunchObject: secondBunchObject  
 };  
}  
  
*/\*\*  
 \* Считает и выписывает тремя разными способами энтропию файла  
 \*/***function** printEntropy() {  
 **let** entropy = 0;  
 **let** huffEntropy = 0;  
 **let** fanoEntropy = 0;  
  
 **for** (**let** key **in** alphabet) {  
 entropy += -alphabet[key].probability \* Math.log(alphabet[key].probability) / Math.log(2);  
 huffEntropy += alphabet[key].probability \* alphabet[key].huffLength;  
 fanoEntropy += alphabet[key].probability \* alphabet[key].fanoLength;  
 }  
  
 console.log('File\'s entropy:', entropy, 'bit');  
 console.log('File\'s Huffman entropy:', huffEntropy, 'bit');  
 console.log('File\'s Shannon Fano entropy:', fanoEntropy, 'bit');  
}  
  
*/\*\*  
 \* Печатает символы, их вероятности, кодовые слова и их длины по Шеннону и Хаффману  
 \*/***function** printAlphabet() {  
 **let** keysArray = [];  
  
 **for** (**let** key **in** alphabet) {  
 keysArray.push(key);  
 }  
  
 console.log('Symbol', 'Probability', 'HuffCodeWord ', 'Length', 'FanoCodeWord ', 'Length');  
  
 keysArray.sort().forEach((key) => {  
 console.log(key, **new** Array(5).join(" "), alphabet[key].probability.toFixed(4), **new** Array(5).join(" "),  
 alphabet[key].huffCodeWord, **new** Array(20 - alphabet[key].huffLength).join(" "), alphabet[key].huffLength,  
 **new** Array(6 - alphabet[key].huffLength.toString().length).join(" "), alphabet[key].fanoCodeWord,  
 **new** Array(20 - alphabet[key].fanoLength).join(" "), alphabet[key].fanoLength);  
 });  
  
 console.log();  
}

**Описание входных данных программ**

Для тестирования использовалась первая глава произведения «Алые паруса» на английском языке размером 36 кб.

**Тестирование программ**



**Анализ результатов работы программы**

|  |  |  |
| --- | --- | --- |
| Исходный файл | Код Хаффмана | Код Шеннона - Фано |
| **4.168** | **4.2098** | **4.2657** |

Полученные энтропии близки друг к другу. Однако, энтропия при кодировании Хаффмана более близка к энтропии файла, чем при кодировании Шеннона – Фано. Это можно объяснить тем, что при кодировании Шеннона – Фано нужно решать проблему разделения элементов на 2 равные по сумме вероятностей части. Результат решения данной проблемы не всегда близок к идеальному, что рождает отклонения от истинного разделения, из-за которых символы приобретают неидеальные кодовые слова.

**Заключение**

В результате выполнения работы были изучены способы оптимального кодирования методом Хаффмана и Шеннона – Фано, по полученным кодовым словам были рассчитаны энтропии файлов в бит.