|  |
| --- |
| Университет итмо, кафедра вт |
| Лабораторная работа №3 по дисциплине «Прикладная математика» |
| «Арифметическое кодирование» |
| Группа Р3302 |
| **Выполнили:**  **Орлова Кристина Александровна**  **Лалетина Екатерина Александровна** |
| **Преподаватель: Тропченко Андрей Александрович** |

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

**Задание**

Для выполнения лабораторной работы №3 необходимо:

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

2. Построить арифметический код 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. Вычислить коэффициент сжатия данных как процентное отношение длины закодированного файла к длине исходного файла.

3. Определить зависимость коэффициента сжатия данных от длины блока при арифметическом кодировании.

4. Декодировать файл, закодированный арифметическим кодом, и сравнить полученный файл с исходным текстом на английском языке.

5. Проанализировать полученные данные

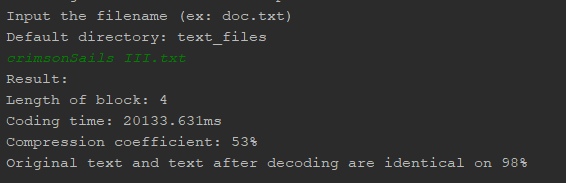
**Исходный код**

**const** fs = require("fs");  
**const** bigdecimal = require("bigdecimal");  
**let** fileContent;  
**let** alphabet = {}; // объект, поля которого - символы, а их значение - вероятность  
**let** codeArray = []; // массив кодов блоков символов  
**let** searchedContent = ''; // сообщение после декодирования  
**let** cumulative = []; // массив кумулативных вероятностей  
**let** textLength = 0;  
**let** blockLength = 4; // длина блока считываемых символов  
  
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:');  
 console.log('Length of block:', blockLength);  
  
 setProbability();  
 countCumulativeProbability();  
 console.time('Coding time');  
 coding();  
 console.timeEnd('Coding time');  
 printCompressionCoefficient();  
 decoding();  
 checkContentOnIdentity();  
 } **catch** (err) {  
 console.log('Error:', err)  
 }  
  
 process.exit();  
 });  
}  
  
*/\*\*  
 \* Считывает содержимое файла и формирует по нему алфавит символов  
 \*  
 \** ***@param*** *fileName  
 \*/***function** readFile(fileName) {  
 fileContent = fs.readFileSync(fileName, 'utf8');  
 textLength = fileContent.length;  
  
 **for** (**let** i = 0; i < textLength; i++) {  
 **let** symbol = fileContent[i];  
  
 **if** (alphabet[symbol] === undefined) {  
 alphabet[symbol] = 1;  
 } **else** {  
 alphabet[symbol]++;  
 }  
 }  
}  
  
*/\*\*  
 \* Меняет в alphabet значение с частоты на вероятность  
 \*/***function** setProbability() {  
 **for** (**let** key **in** alphabet) {  
 alphabet[key] = alphabet[key] / textLength;  
 }  
}  
  
*/\*\*  
 \* Считает кумулативные вероятности  
 \*/***function** countCumulativeProbability() {  
 cumulative.push(0);  
  
 **for** (**let** key **in** alphabet) {  
 **if** (cumulative.length === 1) {  
 cumulative.push(alphabet[key]);  
 } **else** {  
 cumulative.push(cumulative[Object.keys(alphabet).indexOf(key)] + alphabet[key]);  
 }  
 }  
}  
  
*/\*\*  
 \* Переводит дробное десятичное с целой частью 0 в двоичное  
 \*  
 \** ***@param*** *digit  
 \** ***@param*** *ranks  
 \** ***@return*** *{Number}  
 \*/***function** translateIntoBinary(digit, ranks) {  
 **let** result = '';  
  
 **for** (**let** i = 0; i < ranks; i++) {  
 digit \*= 2;  
  
 **if** (digit > 1) {  
 digit -= 1;  
 result += 1;  
 } **else** {  
 result += 0;  
 }  
 }  
  
 **return** parseInt(result);  
}  
  
*/\*\*  
 \* Арифметическое кодирование  
 \*/***function** coding() {  
 **let** cumulativeBig = [];  
  
 cumulative.forEach((probability) => {  
 cumulativeBig.push(**new** bigdecimal.BigDecimal(probability));  
 });  
  
 **for** (**let** blockStartingSymbolIndex = 0; blockStartingSymbolIndex < textLength; blockStartingSymbolIndex += blockLength) {  
 **let** leftLimit = **new** bigdecimal.BigDecimal(0);  
 **let** rightLimit = **new** bigdecimal.BigDecimal(1);  
 **let** intervalLength = **new** bigdecimal.BigDecimal(1);  
  
 **let** blockEndingSymbolIndex = 0;  
  
 **if** (blockStartingSymbolIndex + blockLength > textLength - 1) {  
 **if** (blockStartingSymbolIndex === 0) {  
 blockEndingSymbolIndex = textLength;  
 } **else** {  
 blockEndingSymbolIndex = blockStartingSymbolIndex + (textLength - 1) % blockStartingSymbolIndex + 1;  
 }  
 } **else** {  
 blockEndingSymbolIndex = blockStartingSymbolIndex + blockLength;  
 }  
  
 **for** (**let** blockSymbolIndex = blockStartingSymbolIndex; blockSymbolIndex < blockEndingSymbolIndex; blockSymbolIndex++) {  
 **const** id = Object.keys(alphabet).indexOf(fileContent[blockSymbolIndex]);  
 **const** newLeftLimit = leftLimit.add(intervalLength.multiply(cumulativeBig[id]));  
  
 rightLimit = leftLimit.add(intervalLength.multiply(cumulativeBig[id + 1]));  
 leftLimit = newLeftLimit;  
 intervalLength = rightLimit.subtract(leftLimit);  
 }  
  
 codeArray.push({  
 'decimal': leftLimit, 'binary': translateIntoBinary(leftLimit,  
 Math.ceil(Math.abs(Math.log(parseFloat(intervalLength)) / Math.log(2))))  
 });  
 }  
}  
  
*/\*\*  
 \* Считает и печатает коэффициент сжатия  
 \*/***function** printCompressionCoefficient() {  
 **let** codeSize = 0;  
  
 codeArray.forEach((object) => {  
 codeSize += object.binary.toString().length;  
 });  
  
 console.log('Compression coefficient:', Math.round((codeSize / (8 \* textLength)) \* 100) + '%',  
 (codeSize > (8 \* textLength)) ? '(not efficient)' : '');  
}  
  
*/\*\*  
 \* Выдает предполагаемый индекс символа в алфавите  
 \*  
 \** ***@param*** *digit  
 \** ***@return*** *{number}  
 \*/***function** getSupposedSymbolIndex(digit) {  
 **let** supposedSymbolIndex = 0;  
  
 **for** (**let** i = 0; i < cumulative.length; i++) {  
 **const** probability = cumulative[i];  
  
 **if** (digit >= probability && digit < 1) {  
 supposedSymbolIndex = cumulative.indexOf(probability);  
 } **else** {  
 **if** ((Math.abs(digit - cumulative[supposedSymbolIndex + 1]) <  
 Math.abs(digit - cumulative[supposedSymbolIndex]) \* 0.01) && digit < 1) {  
 **return** supposedSymbolIndex + 1;  
 }  
  
 **return** supposedSymbolIndex;  
 }  
 }  
  
 **return** supposedSymbolIndex;  
}  
  
*/\*\*  
 \* Декодирование  
 \*/***function** decoding() {  
 codeArray.forEach((object) => {  
 **let** digitCount = blockLength;  
 **let** digit = object.decimal;  
  
 **if** (codeArray.length === 1) {  
 digitCount = textLength;  
 } **else if** (codeArray.indexOf(object) === codeArray.length - 1) {  
 digitCount = textLength - (codeArray.length - 1) \* blockLength;  
 }  
  
 **for** (**let** i = 0; i < digitCount; i++) {  
 **const** supposedSymbolIndex = getSupposedSymbolIndex(digit);  
  
 searchedContent += Object.keys(alphabet)[supposedSymbolIndex];  
  
 digit = Math.abs((digit - cumulative[supposedSymbolIndex]) /  
 (cumulative[supposedSymbolIndex + 1] - cumulative[supposedSymbolIndex]));  
 }  
 });  
}  
  
*/\*\*  
 \* Проверяет соответствие полученного текста после декодирования исходному и печатает результат  
 \*/***function** checkContentOnIdentity() {  
 **let** countOfOtherSymbols = 0;  
  
 **for** (**let** i = 0; i < textLength; i++) {  
 **if** (fileContent[i] !== searchedContent[i]) {  
 countOfOtherSymbols++;  
 }  
 }  
  
 **const** percent = Math.round(100 \* (textLength - countOfOtherSymbols) / textLength);  
  
 console.log('Original text and text after decoding are identical', (percent !== 100) ? 'on ' + percent + '%' : '');  
}

**Описание входных данных программы**

Для тестирования использовалась третья глава произведения «Алые паруса» на английском языке размером 22 кб.

**Тестирование программы**



**Анализ результатов работы программы**

Тестирование показало, что для третьей главы произведения «Алые паруса» при длине блока в 4 символа коэффициент сжатия получается 53%; исходный текст соответствует полученному после декодирования на 98% (2% несоответствий); время работы кодирования составило 20 секунд.

С увеличением длины блока увеличивается время работы программы, увеличивается количество несоответствий между файлами. С увеличением длины блока уменьшается длина полученного интервала для каждого блока. Следовательно, растет количество разрядов, необходимых для кодирования последовательности. Следовательно, увеличивается длина кода каждой последовательности и размер полученного файла, из-за чего уменьшается коэффициент сжатия.

**Заключение**

Был изучен способ арифметического кодирования, при котором составляется код не для одного символа, а для целой последовательности. Научились кодировать и декодировать текстовый файл на практике.