Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 7

ОТЧЕТ

по лабораторной работе №6

Тема: «Методы сжатия информации»

Выполнил: студент группы АСУ-17-1б

Хохряков Денис

Проверил: доцент кафедры ИТАС

Шереметьев В.Г.\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_

# ЦЕЛЬ РАБОТЫ

Получить практические навыки по применению различных методов сжатия информации. Получить сравнительную характеристику сжатия информации, используя различные комбинации методов. Сделать вывод.

# ЗАДАНИЕ

Выполнить первое сжатие файла, используя алгоритм Хаффмана, и повторное способом кодирования серий (RLE).

# КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм Хаффмана - один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

RLE - Кодирование длин серий или кодирование повторов — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании (упаковке, сжатии) строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов.

# ХОД РАБОТЫ

1. Подсчитываем кол-во повторяющихся букв и строим дерево Хаффмана (соединяем самые редкие символы в бинарное дерево и так по цепочке).
2. Из дерева получаем таблицу кодов символов (для каждого символа смотрим путь до него, записывая поворот в левое поддерево как 0, а в правое – как 1)
3. Далее просто преобразовываем строку по данной таблице. О декодировании можно не беспокоиться – имея таблицу кодов, получатель точно расшифрует сжатую информацию
4. Преобразовываем строку в HEX формат, чтобы сжать ее по алгоритму RLE
5. Жадным алгоритмом сжимаем строку, кодируя неповторяющиеся символы как 0x<строка> (где x – кол-во неповторяющихся символов), а повторяющиеся как x<строка> (где x – кол-во повторяющихся символов)

# РАБОТА ПРОГРАММЫ

Программа на лету генерирует дерево Хаффмана и производит сжатие по обоим алгоритмам.

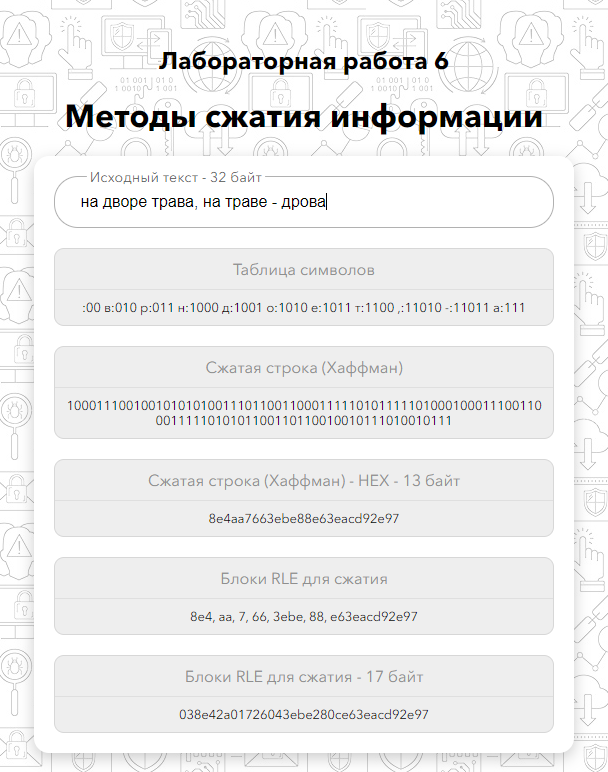
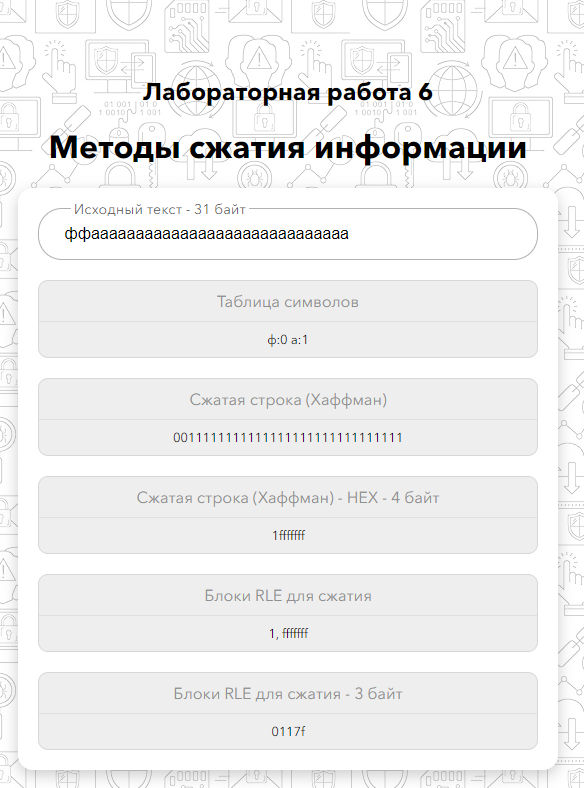
 

Рисунок 1. Сжатие по обоим алгоритмам Рисунок 2. Сжатие повторяющихся символов

Как видно из скриншота, алгоритм Хаффмана очень эффективно сжимает строку. Даже с учетом того, что вместе с сжатой информацией нам нужно передать таблицу символов, уменьшение информации на лицо – (с 32 байт до 13 байт + таблица)

А вот RLE справился с задачей намного хуже. В первом случае он, наоборот, сделал код еще больше за счет вставки экранированного байта «0». Поэтому использовать RLE в сжатия текста крайне неэффективно. Его следует использовать разве что для сжатия изображений, так как там часто встречаются повторяющиеся последовательности.

# ВЫВОД

Алгоритм Хаффмана, несмотря на свою простоту, способен очень эффективно сжимать текстовые данные (особенно в скороговорках, так как там чаще всего повторяются буквы). А вот RLE текст сжимает очень плохо, так как в текстовой информации повторяющиеся более чем 2 раза символы встречаются очень редко. Да и обычные данные он тоже сжимает не очень-то и хорошо.

# КОД ПРОГРАММЫ (ФУНКЦИЯ СЖАТИЯ)

# (Язык программирования – JavaScript)

//Сначала мы должны посмотреть кол-во повторений каждого символа

const chars = new Map();

for(let i = 0; i < obj.input.length; i++){

const a = obj.input[i];

chars.set(a, (chars.get(a) || 0) + 1);

}

let array = [];

for(let [key, value] of chars.entries())

array.push({key, count: value});

//Затем мы попарно преобразуем элементы массива в дерево

while(array.length > 1){

array = \_.sortBy(array, ['count']);

const newArr = array.slice(2);

newArr.push({ count: array[0].count+ array[1].count, children: array.slice(0, 2) });

array = newArr;

}

//А вот потом создаем таблицу кодов

const table = new Map();

setTable(array[0], table);

let tableStr = "";

for(let [key, value] of table)

tableStr += key+":"+value + " ";

obj.tableStr = tableStr;

//Ну, и сжимаем строку

let archiveStr = "";

for(let i = 0; i < obj.input.length; i++)

archiveStr += table.get(obj.input[i]);

obj.archiveStr = archiveStr;

//Затем преобразуем бинарную строку в HEX

let hexStr = "";

let \_str = "";

for(let i = archiveStr.length-1; i >= 0; i--){

\_str = archiveStr[i]+\_str;

if(\_str.length === 8){

hexStr = parseInt(\_str, 2).toString(16) + hexStr;

\_str = "";

}

}

if(\_str)

hexStr = parseInt(\_str, 2).toString(16) + hexStr;

obj.hexStr = hexStr;

//А вот затем мы используем наш гениальный просто жадный алгоритм – съедаем все символы до тех пор, пока не встретим повторение, или же, наоборот, два различающихся символа. Таким образом у нас образовываются блоки, содержащие либо только повторяющиеся символы, либо только различающиеся

const arr = [];

let cursor = 1;

while(cursor <= hexStr.length){

let c = cursor;

let str = "";

const condition = (hexStr[c] === hexStr[c-1])

while((hexStr[c] === hexStr[c-1]) === condition && c < hexStr.length && c-cursor < 16)

c++;

if(condition || c === hexStr.length)

c+=1;

arr.push(hexStr.slice(cursor-1, c-1));

cursor = c;

}

obj.RLE = arr;

//Ну и соединяем данные блоки по описанной схеме

obj.RLEstr = arr.map(item => (item.length <= 1 || item[0] !== item[1])?("0"+item.length.toString(16)+item): (item.length.toString(16)+item[0]) ).join("");