**Описание алгоритма RLE.**

**Кодирование длин серий** ( англ. ***r****un-****l****ength* ***e****ncoding*, *RLE*) 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.

**Пример использования.**

Рассмотрим изображение, содержащее текст черного цвета на сплошном белом фоне. При построчном чтении пикселей такого изображения будут встречаться серии белых (фон) и чёрных (буквы) пикселей. Буквой B обозначим чёрный пиксель, а буквой W — белый. Рассмотрим некую произвольную строку изображения длиной 67 символов:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW

Посчитаем количество повторяющихся символов:

1. 12 символов «W»;
2. 1 символ «B»;
3. 12 символов «W»;
4. 3 символа «B»;
5. 24 символа «W»;
6. 1 символ «B»;
7. 14 символов «W».

Итого найдено 7 серий. Заменим серии на число повторов и сам повторяющийся символ:

12**W**1**B**12**W**3**B**24**W**1**B**14**W**

Получилась последовательность из 18 символов. Исходная последовательность состояла из 67 символов. Данные были сжаты в 67/18≈3.72 раза.

Возьмём строку, состоящую из большого количества неповторяющихся символов:

ABCABCABCDDDFFFFFF

После сжатия методом RLE такая строка будет выглядеть так:

**1**A**1**B**1**C**1**A**1**B**1**C**1**A**1**B**1**C**3**D**6**F

Исходная строка состоит из 18 символов, а сжатая — из 22. Размер данных увеличился в 22/18≈1.22 раза.

Чтобы после сжатия размер данных не увеличивался, алфавит, в котором записаны длины серий, делят на две части (обычно равные). Например, алфавит целых чисел можно разделить на две части: положительные и отрицательные числа. Положительные числа используют для записи количества повторов одного символа, а отрицательные — для записи количества неодинаковых символов, следующих друг за другом.

Посчитаем символы с учётом вышесказанного:

* сначала друг за другом следуют 9 не одинаковых символов: «ABCABCABC»;
* затем записаны 3 символа «D»;
* наконец записаны 6 символов «F».

Сжатая строка запишется в виде:

**-9**ABCABCABC**3**D**6**F

Исходная строка состоит из 18 символов, а сжатая — из 15. Размер данных уменьшился в 18/15=1.2 раза.

Допустим, реализация метода RLE для записи длин серий (для подсчёта количества символов) использует переменную целочисленного типа со знаком «signed char». В такую переменную можно записать числа от -128 до 127 включительно. Как же быть, если длина серии равна 128 символам и более? В этом случае серию разделяют на части так, чтобы длина части не превышала 127 символов. Например, серия, состоящая из 256 символов «A», будет закодирована следующей строкой (256=127+127+2):

**127**A**127**A**2**A

Запись на некотором языке программирования алгоритма RLE с учётом этих ограничений нетривиальна.

Конечно, кодирование, которое используется для хранения изображений, оперирует с двоичными данными, а не с символами ASCII, как в рассмотренных примерах, однако принцип остаётся тем же.

**Следующий Пример кодирования.**

Пусть есть строка



Если некоторый символ встречается много (пока не уточняем сколько) раз подряд, образуя цепочку, заменим цепочку на специальную трехбайтовую последовательность <Escape\_symbol, Counter, Symbol>:

1. Escape\_symbol — экранирующий символ (иначе — эспкейп-символ), показывающий, что следующие за ним два байта надо интерпретировать специальным образом. Без ограничения общности в качестве Escape symbol можно выбрать символ # .
2. Counter — число повторов (длина цепочки). Отводя под число повторов один байт, мы автоматически разрешаем цепочки длиной не более 255 символов. Число повторов удобно представить как символ с ASCII-кодом, равным числу повторов.
3. Symbol — сам повторяющийся символ.

Вспомним, что в таблице ASCII-кодам 65, 69 и 38 соответствуют английские буквы А и Е и амиесанд &. Таким образом, рассматриваемую строку можно представить в виде



Трехбайтовые последовательности <Escape\_symbol, Counter, Symbol> выделили подчеркиванием.

Сразу отметим, что среди символов ASCII есть непечатные, такие как символы с кодами от 0 до 31, означающие конец файла, возврат каретки, вертикальную табуляцию и т. п. Чтобы изобразить их, применяем условную запись — код в рамке. Строку



можно представить в сжатом виде



Таким образом, вместо 36 символов мы использовали лишь 18, сжали строку в два раза.

Как происходит декодирование? Просматривая сжатую строку, декодировщик ищет спецсимвол # . Встретив # , декодировщик понимает, что следующие два символа надо обработать по специальной схеме: повторить нужное число раз соответствующий символ. Например, встретив , декодировщик развернет ее в строку 000000000 . Cформулируем правила кодирования.

1. Цепочки длиной менее трех символов не выгодно кодировать, так как в результате получается три байта. Таким образом, кодируем цепочки, длина которых от четырех байт включительно.
2. Если в сжимаемой строке имеются цепочки длиной более 255 символов, то их длину нельзя представить в виде одного байта. Выход такой: от длинной цепочки отрезать куски по 255 байт пока это возможно и кодировать эти куски обычным способом, т.е. писать и повторяемый символ. В результате от длинной цепочки останется хвост, длина которого не более 255, его можно закодировать обычным способом.
3. Если в сжимаемой строке имеется символ, совпадающий со спецсимволом # , то просто так его переписать в сжатую строку нельзя, иначе это вызовет ошибку при декодировании. Спецсимволы надо кодировать всегда.

Проиллюстрируем все эти правила на примере



Заметим, что 823 = 3\*255 + 55, 513 = 2\*255 + 3, ASCII-коду 55 соответствует цифра 7. В результате получим



Приведем еще пример. Строка будет закодирована как # # #. потому что ASCII-коду 36 соответствует символ # . В итоге первый символ # — это спецсимвол, второй # — 36 повторов, третий # — символ, который надо повторить 36 раз. Изложенный алгоритм сжатия допускает одну оптимизацию. Заметим, что цепочки, состоящие не из # , кодируются лишь в том случае, если их длина не менее 4 символов; следовательно, символы с кодами 0 , 1 , 2 , 3 не используются в качестве Counter. Можно сделать своего рода сдвиг на 4 и кодировать цепочку аааа не как , но как . Аналогично для цепочки ааааа вместо получим и т.д. Таким образом удастся цепочку длиной 259 символов упаковать в три байта: ; без оптимизации та же цепочка из 259 символов потребовала бы 6 байт: . Эффективность алгоритма сжатия оценивают по коэффициенту сжатия. Коэффициентом сжатия назовем отношение длины исходного файла к длине сжатого файла.

**Постановка задачи**

1.В текстовом файле input.txt записана строка, но в ней нет цифр. Необходимо закодировать ее так: Дано ABCABCABCDDDFFFFFF, Получилось ABCABCABC**3**D**6**F.

Закодированную строку сохранить в файле code.txt. Вычислить коэффициент сжатия (Коэффициентом сжатия назовем отношение длины исходного файла к длине сжатого файла**)**.

2.Раскодировать строку из файла code.txt (строку может ввести преподаватель), результат сохранить в decode.txt. (Предложение по проверке, является символ цифрой. Будем проверять, содержится символ в строке digit=’0123456789’).

**Указания к решению**

Объект FileSystemObject обеспечивает доступ к файловой системе Windows. Его конструктор имеет вид: new ActiveXObject (“Scripting.FileSystemObject”). Сценарий

может создать только один экземпляр данного объекта, сколько бы раз в нем ни вызывался данный конструктор.

Для чтения из текстового файла или записи в текстовый файл необходимо этот файл сначала открыть (на чтение или запись соответственно). Приведем два образца кода.

Откроем файл на запись и запишем строку:

fso = new ActiveXObject("Scripting.FileSystemObject");

fh = fso.OpenTextFile("c:\\file.txt", 2, true);

fh.WriteLine("Тестовая строка.");

fh.Close();

Откроем файл на чтение и прочитаем все его содержимое и выведем на экран:

fso = new ActiveXObject ("Scripting.FileSystemObject");

fh = fso.OpenTextFile("c:\\test.txt");

s = fh.ReadAll();

fh.Close();

WSH.echo(s);

**Работа со строками**.

Одно из самых частых действий со строкой – это получение ее длины.

length – длина или, иными словами, количество символов (включая пробелы) в строке; целое число.

var str = "My\n"; // 3 символа. Третий - перевод строки

alert( str.length ); // 3

Чтобы получить символ, используйте вызов charAt(позиция). Первый символ имеет позицию 0:

var str = "jQuery";

alert( str.charAt(0) ); // "j"

В JavaScript нет отдельного типа «символ», так что charAt возвращает строку, состоящую из выбранного символа.

**Все строки имеют внутреннюю кодировку Юникод.**

Неважно, на каком языке написана страница, находится ли она в windows-1251 или utf-8. Внутри JavaScript-интерпретатора все строки приводятся к единому «юникодному» виду. Каждому символу соответствует свой код.

Есть метод для получения символа по его коду:

String.fromCharCode(code)

Возвращает символ по коду code:

alert( String.fromCharCode(1072) ); // 'а'

…И метод для получения цифрового кода из символа:

str.charCodeAt(pos)

Возвращает код символа на позиции pos. Отсчет позиции начинается с нуля.

alert( "абрикос".charCodeAt(0) ); // 1072, код 'а'

Для поиска подстроки есть метод indexOf(подстрока[, начальная\_позиция]).

Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено. Например:

var str = "Widget with id";

alert( str.indexOf("Widget") ); // 0, т.к. "Widget" найден прямо в начале str

alert( str.indexOf("id") ); // 1, т.к. "id" найден, начиная с позиции 1

alert( str.indexOf("widget") ); // -1, не найдено, так как поиск учитывает регистр

<https://learn.javascript.ru/string>

## Численное преобразование

## Численное преобразование происходит в математических функциях и выражениях, а также при сравнении данных различных типов (кроме сравнений ===, !==).

Для преобразования к числу в явном виде можно вызвать Number(val), либо, что короче, поставить перед выражением унарный плюс "+":

var a = +"123"; // 123

var a = Number("123"); // 123, тот же эффект

| Значение | Преобразуется в... |
| --- | --- |
| undefined | NaN |
| null | 0 |
| true / false | 1 / 0 |
| Строка | Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки "считывается" число, при ошибке результат NaN. |

Например:

// после обрезания пробельных символов останется "123"

alert( +" \n 123 \n \n" ); // 123

Ещё примеры:

* Логические значения:
* alert( +true ); // 1

alert( +false ); // 0

* Сравнение разных типов – значит численное преобразование:

alert( "\n0 " == 0 ); // true

При этом строка "\n0" преобразуется к числу, как указано выше: начальные и конечные пробелы обрезаются, получается строка "0", которая равна 0.

* С логическими значениями:

alert( "\n" == false );

alert( "1" == true );

Здесь сравнение "==" снова приводит обе части к числу. В первой строке слева и справа получается 0, во второй 1.

**Помощь.**

**1 часть**

Откроем файл на чтение и прочитаем все его содержимое:

fso = new ActiveXObject ("Scripting.FileSystemObject");

fh = fso.OpenTextFile("c:\\test.txt");

inText = fh.ReadAll();

fh.Close();

node js

let fs = require('fs');

inText = fs.readFileSync('output.txt');

/\* в строке inText находим длины цепочек подряд идущих символов\*/

i = 0

n = 1;

while (i < inText.length){

while(inText.charAt(i) == inText.charAt(i+n))

n++;

WSH.echo (inText.charAt(i)," - ", n)

i += n;

n = 1;

}

Откроем файл на запись и запишем строку:

fso = new ActiveXObject("Scripting.FileSystemObject");

fh = fso.OpenTextFile("c:\\file.txt", 2, true);

fh.WriteLine("Тестовая строка.");

fh.Close();

node js

let fs = require('fs');

fs.writeFileSync('output.txt', testString);

**2 часть**

Раскодируем строку.

Есть закодированная строка inText.

i = 0

n = 1;

number=’0123456789’

while (i < inText.length){

strnum=’’

while(inText.charAt(i) находим в строке number, т.е пока inText.charAt(i) цифра) {

strnum=strnum+ inText.charAt(i)

i ++;

}

If (strnum.length>0)

k= преобразуем strnum в число

else k=1

for (j=1; j<=k; j++)

строка=строка+ inText.charAt(i)

i ++;

}