

Ранее нами были изучены способы кодировки Шеннона-Фано и Хаффмана, позволяющие составить оптимальный префиксный код. Однако эти алгоритмы можно улучшить, введя дополнительные шаги кодирования.














RLE, run-length encoding или кодирование длин серий - алгоритм сжатия данных, заменяющий повторяющиеся символы на один символ и число его повторов. Его реализация довольно проста, в реализации данной работы для хранения информации о числе повторов выделяется один байт. Первый бит этого байта содержит информацию о том, повторяющаяся ли в дальнейшем последовательность будет перечислена или нет. Следующие 7 бит содержат информацию о количестве символов. Если последовательность неповторяющаяся, то в 7 битах будет указано количество, сколько следующих байт нужно будет считать без повторов, если же следует последовательность повторяющаяся, то следующий байт будет повторяться в исходном файле столько, сколько указано в следующих 7 битах информационного байта. В первом случае можно сохранить таким образом до 128 включительно байт. 0 байт не может быть, поэтому к числу байт нужно прибавить 1, например 0 неповторяющихся байт должны быть интерпретированы как 1 неповторяющийся байт. В случае повторяющихся байтов как минимум 2 байта будут повторяться, поэтому к числу нужно прибавить 2, таким образом можно записать до 129 включительно элементов.

Для решения проблемы с большим количеством повторов/уникальных символов можно расширить размер информационного байта или записать символы в несколько подходов.

Были проведены несколько испытаний для различных последовательностей файлов с различными комбинациями кодировок:

ads.txt












Текст содержит случайные русские символы, цифры и т.д.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
HUF only		1.896	3129
SF only		1.896	3130
Original		1.000	5933
RLE>SF		0.862	6880
RLE>HUF		0.862	6880
SF>RLE		0.501	11835
HUF>RLE		0.501	11835
RLE only		0.458	12946
RLE2		0.455	13048
SF2		0.455	13048
HUF2		0.455	13048
RLE>SF>RLE		0.044	136269
RLE>HUF>RLE		0.044	136269

В этом случае хорошо себя показали сообщения, закодированные методом Хаффмана и Шеннона-Фано, так как в тексте используется не весь алфавит. Кроме того, русские символы, закодированные в UTF-8 состоят из двух байтов, первый байт достаточно часто повторяется, что позволило ещё сильнее сократить данные в кодировках. Остальные вариации незначительно или сильно увеличили объём информации.

api-ms-win-core-datetime-l1-1-0.dll














Файл динамической библиотеки

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
RLE>SF>RLE		Не прошёл	
RLE>HUF>RLE		Не прошёл	
HUF only		1.085	21711
SF only		1.079	21835
Original		1.000	23560
SF>RLE		0.112	210637
HUF>RLE		0.112	210637
RLE>SF		0.016	1512511
RLE>HUF		0.016	1512511
RLE only		0.015	1526600
RLE2		0.006	3746664
SF2		0.006	3746664
HUF2		0.006	3746664

Файл содержит случайные равномерно распределённые байты, поэтому кодирование по Шеннону-Фано и Хаффману немного уменьшило конечный результат, но не сильно. Применение RLE так же не имело смысла, так как повторяющихся байтов в файле мало, и его применение приведёт только к увеличению информации.

black.dll












Графическое изображение, содержащее только чёрные пиксели

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
SF>RLE		148.661	1265
HUF>RLE		148.661	1265
RLE>SF		109.271	1721
RLE>HUF		109.271	1721
RLE only		49.856	3772
RLE>SF>RLE		37.899	4962
RLE>HUF>RLE		37.899	4962
RLE2		13.547	13882
SF2		13.547	13882
HUF2		13.547	13882
SF only		7.894	23823
HUF only		7.894	23823
Original		1.000	188056

Здесь алгоритм RLE показывает себя замечательно, в файле содержатся только повторяющиеся пиксели, RLE может очень легко сократить такие последовательности. Добавив алгоритм Шеннона-Фано или Хаффмана сократим повторяющиеся цепочки информационных байтов и байтов идущих после них, кодирование RLE>HUF или RLE>SF вырвалось вперёд. Остальные алгоритмы так же справились неплохо.

img_random.bmp














Случайное изображение

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
RLE>SF>RLE		Не прошёл	
RLE>HUF>RLE		Не прошёл	
HUF only		1.019	46241
SF only		1.016	46396
Original		1.000	47138
SF>RLE		0.218	216441
HUF>RLE		0.218	216441
RLE>SF		0.006	8019305
RLE>HUF		0.006	8019305
RLE only		0.006	8289314
RLE2		0.001	41060624
SF2		0.001	41060624
HUF2		0.001	41060624

Алгоритмы Хаффмана и Шеннона-Фано снова выбиваются вперёд. Изображение больше не содержит повторяющихся цепочек байтов. Алгоритмы с применением RLE менее эффективны и приводят к увеличению объёма памяти. Стоит также отметить, что алгоритм Хаффмана оказался эффективней алгоритма Шеннона-Фано.

img_random.png














То же изображение, но в нём уже используются алгоритм сжатия LZ77.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
HUF only		1.002	25719
Original		1.000	25780
SF only		0.999	25807
SF>RLE		0.935	27567
HUF>RLE		0.935	27567
RLE>SF		0.017	1515766
RLE>HUF		0.017	1515766
RLE only		0.017	1519434
RLE>SF>RLE		0.016	1613993
RLE>HUF>RLE		0.016	1613993
RLE2		0.011	2453141
SF2		0.011	2453141
HUF2		0.011	2453141

Повторное сжатие не дало лучших результатов и привело лишь к увеличению объёма памяти. Только алгоритм Хаффмана позволил немного уменьшить изображение.

laughter_of_terror_ru.txt














Последовательность из нескольких русских символов, пробелов и переносов строки в UTF-8.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
HUF only		2.785	158
SF only		2.667	165
SF>RLE		2.667	165
HUF>RLE		2.667	165
Original		1.000	440
RLE>SF		0.971	453
RLE>HUF		0.971	453
RLE>SF>RLE		0.921	478
RLE>HUF>RLE		0.921	478
RLE only		0.365	1204
RLE2		0.362	1214
SF2		0.362	1214
HUF2		0.362	1214

Последовательность состоит из ограниченного алфавита, повторяющихся цепочек нет. Именно поэтому кодирование только при помощи Шеннона-Фано или Хаффмана снова эффективно, причём кодирование только Хаффманом оказывается эффективней. Последовательность содержит мало повторяющихся цепочек, поэтому применение RLE эффективно только в конце, когда символы уже преобразованы алгоритмами Хаффмана или Шеннона-Фано.

lorem_ipsum.txt

Последовательность из латинских символов.












Название	Шкала	Коэффициент сжатия	Размер (в байтах)
HUF only		1.862	5798
SF only		1.859	5807
Original		1.000	10797
SF>RLE		0.167	64837
HUF>RLE		0.167	64837
RLE>SF		0.077	139431
RLE>HUF		0.077	139431
RLE only		0.042	257270
RLE2		0.042	259280
SF2		0.042	259280
HUF2		0.042	259280
RLE>SF>RLE		0.000	36658785
RLE>HUF>RLE		0.000	36658785

В обычном тексте символы не будут равномерно распределены, поэтому алгоритмы Шеннона-Фано и Хаффмана снова вырываются вперёд. Причём Хаффман всё ещё оказывается эффективней. Повторяющиеся байты в обычном тексте повторяются не так часто, следовательно применение RLE не сильно эффективно. Повторное кодирование и тройного кодирования всё так же неэффективны и приводят к сильному увеличению размера файла.

metal_alphabet_lyrics.txt

Последовательность из повторяющихся латинских символов, разделённых переводами














строки.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
RLE>SF		10.581	62
RLE>HUF		10.581	62
RLE>SF>RLE		10.413	63
RLE>HUF>RLE		10.413	63
RLE only		5.165	127
RLE2		5.125	128
SF2		5.125	128
HUF2		5.125	128
SF>RLE		1.749	375
HUF>RLE		1.749	375
HUF only		1.652	397
SF only		1.648	398
Original		1.000	656

Любой алгоритм в данном случае оказался эффективным. Самым эффективным оказалось применение RLE и Хаффмана или Шеннона-Фано. RLE сокращает повторяющиеся байты, а ШФ или Хаф сокращает переводы, выраженные двумя байтами.

metal_alphabet_lyrics_ru.txt

Последовательность из повторяющихся русских символов, разделённых переводами строки.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
SF>RLE		7.817	229
HUF>RLE		7.817	229
RLE>SF>RLE		5.542	323
RLE>HUF>RLE		5.542	323
HUF only		2.180	821
SF only		1.974	907
RLE>SF		1.935	925
RLE>HUF		1.935	925
Original		1.000	1790
RLE only		0.992	1804
RLE2		0.984	1819
SF2		0.984	1819
HUF2		0.984	1819

В данном случае буквы будут занимать уже два байта, поэтому эффективней окажется алгоритм, который сначала применяет алгоритм Хаффмана или Шеннона-Фано, и затем - RLE. Одним из возможных решений для увеличения эффективности RLE является расширение сегментизации до 2 байтов.

noise.bmp

Изображение из случайных равномерно распределённых байтов

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
HUF only		3.281	57313
SF only		3.277	57387
RLE>SF		2.183	86138
RLE>HUF		2.183	86138
RLE only		1.035	181624
Original		1.000	188056
SF>RLE	████████	0.024	7831501
HUF>RLE	████████	0.024	7831501
RLE2	████████████████	0.018	10353387
SF2	████████████████	0.018	10353387
HUF2	████████████████	0.018	10353387
RLE>SF>RLE	████████████████████████	0.012	16308057
RLE>HUF>RLE	████████████████████████	0.012	16308057

Результат оказался неожиданным, алгоритм Хаффмана и Шеннона-Фано позволил уменьшить размер изображения в 3 раза. Даже применение алгоритмы с применением RLE оказались эффективными.

scream_of_terror_en.txt

Последовательность из латинской буквы, пробелов и переносов строки в UTF-8.

Название	Шкала	Коэффициент сжатия	Размер (в байтах)
RLE>SF	■	14.120	25
RLE>HUF	■	14.120	25
RLE>SF>RLE	■	13.577	26
RLE>HUF>RLE	■	13.577	26
SF>RLE	■	9.051	39
HUF>RLE	■	9.051	39
SF only	■	6.537	54
HUF only	■	6.537	54
RLE only	■	5.516	64
RLE2	■	5.431	65
SF2	■	5.431	65
HUF2	■	5.431	65
Original	██	1.000	353

Эффективней всего оказалось применить алгоритм RLE, который сократит повторяющиеся символы и затем алгоритм Шеннона-Фано или Хаффмана.

Полный исходный код эксперимента:

Main.java (https://github.com/IAmProgrammist/information_theory/blob/main/src/main/java/rchat/info/lab3/Main.java)

Coder.java (https://github.com/IAmProgrammist/information_theory/blob/main/src/main/java/rchat/info/libs/Coder.java)

Файлы эксперимента:

Ссылка (https://github.com/IAmProgrammist/information_theory/tree/main)

Выводы: наименее эффективными оказались алгоритмы с двойным применением RLE, методы повторного кодирования при помощи того же метода. Чаще всего они проигрывали или вызывали переполнение кучи. Эффективней оказались алгоритмы с применением RLE до или после Шеннона-Фано или Хаффмана в ситуациях с коротким алфавитом и множеством повторяющихся символов. Такие методы будут полезны, например, при кодировании очень тёмных или очень светлых изображений, где очень часто встречаются повторяющиеся подряд пиксели. Не всегда самыми эффективными, но всегда уменьшающими размер оказались алгоритмы Шеннона-Фано или Хаффмана без применения RLE.