Сжатие изображений

**Сжатие изображений** — применение [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [сжатия данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C) к изображениям, хранящимся в [цифровом виде](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D1%82_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0). В результате сжатия уменьшается размер изображения, из-за чего уменьшается время передачи изображения по сети и экономится пространство для хранения.

Сжатие изображений подразделяют на [сжатие с потерями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D1%81_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8F%D0%BC%D0%B8) качества и [сжатие без потерь](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C). Сжатие без потерь часто предпочтительней для искусственно построенных изображений, таких как [графики](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8), иконки программ, либо для специальных случаев, например, если изображения предназначены для последующей обработки алгоритмами распознавания изображений. Алгоритмы сжатия с потерями при увеличении степени сжатия как правило порождают хорошо заметные человеческому глазу [артефакты](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%82%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D1%8B_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F).

## **Примеры алгоритмов**

### Алгоритмы сжатия без потерь

* [RLE](https://ru.wikipedia.org/wiki/RLE) — используется в форматах [PCX](https://ru.wikipedia.org/wiki/PCX) — в качестве основного метода и в форматах [BMP](https://ru.wikipedia.org/wiki/BMP), [TGA](https://ru.wikipedia.org/wiki/Truevision_TGA), [TIFF](https://ru.wikipedia.org/wiki/TIFF) в качестве одного из доступных.
* [LZW](https://ru.wikipedia.org/wiki/LZW) — используется в формате [GIF](https://ru.wikipedia.org/wiki/GIF)
* [Deflate](https://ru.wikipedia.org/wiki/Deflate) — используется в формате [PNG](https://ru.wikipedia.org/wiki/PNG)

# 

**Кодирование длин серий(RLE)**

**Кодирование длин серий** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***r****un-****l****ength****e****ncoding*, *RLE*) или **кодирование повторов** — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [сжатия данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), 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 [чёрного](https://ru.wikipedia.org/wiki/%D0%A7%D1%91%D1%80%D0%BD%D1%8B%D0%B9) цвета на сплошном [белом](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D0%BB%D1%8B%D0%B9_%D1%86%D0%B2%D0%B5%D1%82) фоне. При построчном чтении [пикселей](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D1%81%D0%B5%D0%BB%D1%8C) такого изображения будут встречаться серии белых (фон) и чёрных (буквы) [пикселей](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D1%81%D0%B5%D0%BB%D1%8C). Буквой B обозначим чёрный пиксель, а буквой W — белый. Рассмотрим некую произвольную строку изображения длиной 51 символ:

WWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW

Посчитаем количество символов:

1. 4 символа «B»;
2. 47 символов «W».

Итого найдено 5 серий. Заменим серии на число повторов и сам повторяющийся символ:

9**W**3**B**24**W**1**B**14**W**

Получилась последовательность из 12 символов. Исходная последовательность состояла из 51 символа. Данные были сжаты в 51/12≈4.25 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](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) можно разделить на две части: положительные и [отрицательные](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) 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](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) 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](https://ru.wikipedia.org/wiki/ASCII), как в рассмотренных примерах, однако принцип остаётся тем же.

## **Применение**

Очевидно, что такое кодирование эффективно для данных, содержащих большое количество серий, например, для простых графических изображений, таких как [иконки](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D1%87%D0%BE%D0%BA_(%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0)) и графические рисунки. Однако это кодирование плохо подходит для изображений с плавным переходом тонов, таких как фотографии.

Распространённые форматы для упаковки данных с помощью RLE включают в себя [PackBits](https://ru.wikipedia.org/w/index.php?title=PackBits&action=edit&redlink=1), [PCX](https://ru.wikipedia.org/wiki/PCX) и [ILBM](https://ru.wikipedia.org/wiki/ILBM).

Методом кодирования длин серий могут быть сжаты произвольные [файлы с двоичными данными](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), поскольку спецификации на форматы файлов часто включают в себя повторяющиеся байты в области выравнивания данных. Тем не менее, современные системы сжатия (например, [Deflate](https://ru.wikipedia.org/wiki/Deflate)) чаще используют алгоритмы на основе [LZ77](https://ru.wikipedia.org/wiki/LZ77), которые являются обобщением метода кодирования длин серий и оперируют с последовательностями символов вида «BWWBWWBWWBWW».

Звуковые данные, которые имеют длинные последовательные серии байт (такие как низкокачественные [звуковые семплы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%BF%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) могут быть сжаты с помощью RLE после того, как к ним будет применено [Дельта-кодирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D1%8C%D1%82%D0%B0-%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

# Алгоритм Лемпеля — Зива — Велча

**Алгори́тм Ле́мпеля — Зи́ва — Уэлча** (**Lempel-Ziv-Welch**, **LZW**) — это универсальный [алгоритм сжатия данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) без потерь, созданный [Авраамом Лемпелем](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8C,_%D0%90%D0%B2%D1%80%D0%B0%D0%B0%D0%BC), [Яаковом Зивом](https://ru.wikipedia.org/wiki/%D0%97%D0%B8%D0%B2,_%D0%AF%D0%B0%D0%BA%D0%BE%D0%B2) и [Терри Велчем](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D1%87,_%D0%A2%D0%B5%D1%80%D1%80%D0%B8). Он был опубликован Велчем в 1984 году[[1]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_bcf6c4b5d55cf9ae-1) в качестве улучшенной реализации алгоритма [LZ78](https://ru.wikipedia.org/wiki/LZ78), опубликованного Лемпелем и Зивом в 1978 году[[2]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_0f4d07904e6fbb80-2). Алгоритм разработан так, чтобы его было достаточно просто реализовать как программно, так и аппаратно[[1]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_bcf6c4b5d55cf9ae-1).

Акроним «LZW» указывает на фамилии изобретателей алгоритма: Лемпель, Зив и Велч, но многие[[*кто?*](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%98%D0%B7%D0%B1%D0%B5%D0%B3%D0%B0%D0%B9%D1%82%D0%B5_%D0%BD%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D1%85_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9)] утверждают, что, поскольку патент принадлежал Зиву, то метод должен называться *алгоритмом Зива — Лемпеля — Велча*.



**Описание**

Данный алгоритм при кодировании (сжатии) сообщения динамически создаёт словарь фраз: определённым последовательностям символов (фразам) ставятся в соответствие группы битов (коды) фиксированной длины (например, 12-битные, как предлагается в исходной статье Велча[[1]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_bcf6c4b5d55cf9ae-1)). Словарь инициализируется всеми 1-символьными фразами (в случае 8-битных символов — это 256 фраз). По мере кодирования алгоритм просматривает текст символ за символом слева направо. При чтении алгоритмом очередного символа в данной позиции находится строка W максимальной длины, совпадающая с какой-то фразой из словаря. Затем код этой фразы подаётся на выход, а строка WK, где K — это символ, следующий за W во входном сообщении, вносится в словарь в качестве новой фразы и ей присваивается какой-то код (так как W выбрана [жадно](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), WK ещё не содержится в словаре). Символ K используется в качестве начала следующей фразы. Более формально данный алгоритм можно описать следующей последовательностью шагов:

1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входной фразы W первым символом сообщения.
2. Если КОНЕЦ\_СООБЩЕНИЯ, то выдать код для W и завершить алгоритм.
3. Считать очередной символ K из кодируемого сообщения.
4. Если фраза WK уже есть в словаре, то присвоить входной фразе W значение WK и перейти к Шагу 2.
5. Иначе выдать код W, добавить WK в словарь, присвоить входной фразе W значение K и перейти к Шагу 2.

Алгоритму декодирования на входе требуется только закодированный текст: соответствующий словарь фраз легко воссоздаётся посредством имитации работы алгоритма кодирования .

## **Реализация**

Примечательной особенностью алгоритма LZW является простота реализации, благодаря которой он до сих пор очень популярен, несмотря на зачастую худшую степень сжатия по сравнению с такими аналогами, как [LZ77](https://ru.wikipedia.org/wiki/LZ77)[[3]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_7fd43dd3b64b2dfa-3). Обычно LZW реализуется с помощью [префиксного дерева](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), содержащего фразы из словаря: для нахождения W нужно просто прочитать как можно более длинную строку из корня дерева, затем для добавления новой фразы WK нужно присоединить к найденному узлу нового сына по символу K, а кодом фразы W может выступать индекс узла в массиве, содержащем все узлы.

### Кодирование фраз

Использование для фраз кодов фиксированной длины (12 битов в описании Велча[[1]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_bcf6c4b5d55cf9ae-1)) может негативно сказаться на эффективности сжатия, так как, во-первых, для начальных кодируемых символов этот подход скорее будет раздувать данные, а не сжимать (если символ занимает 8 битов), и во-вторых, общий размер словаря (212=4096) получается не так велик. Первая проблема решается кодированием выходной последовательности [алгоритмом Хаффмана](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0) (возможно, [адаптивным](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0%D0%BF%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0)) или [арифметическим кодированием](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Для решения второй используют другие подходы.

Первый простой вариант — применить какой-нибудь оптимальный [универсальный код](https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) типа [кода Левенштейна](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0) или [кода Элиаса](https://ru.wikipedia.org/wiki/%D0%9E%D0%BC%D0%B5%D0%B3%D0%B0-%D0%BA%D0%BE%D0%B4_%D0%AD%D0%BB%D0%B8%D0%B0%D1%81%D0%B0). В таком случае теоретически словарь может расти неограниченно.

Другой более распространённый вариант — изменять максимальный возможный размер словаря с ростом числа фраз.[[4]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_97c9335d8eba9b9d-4) Изначально, например, максимальный размер словаря полагается 29 (28 кодов при этом уже заняты фразами для кодирования 8-битовых одиночных символов) и на код фразы отводится 9 битов. Когда число фраз становится 29, максимальный размер становится 210 и на коды отводится 10 битов. И так далее. Таким образом, теоретически словарь может быть сколь угодно большим. Этот метод продемонстрирован в [примере](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80) ниже (обычно, тем не менее, максимальный размер словаря ограничивается; например в LZC — популярной модификации LZW, рассматриваемой ниже — длины кодов растут от 9 до 16 битов.).

В большинстве реализаций последнего метода число битов, выделяемых на код фразы, увеличивается до добавления новой фразы WK, переполняющей словарь, но после записи на выход кода W. Например, пусть в данный момент в процессе работы алгоритма длина кода — p битов, и алгоритм собирается выдать код фразы W и добавить новую фразу WK в словарь; если код WK равен 2p (то есть WK переполняет словарь), то сначала выдаётся p-битовый код W и только после этого p увеличивается на один, чтобы последующие коды занимали p+1 битов. Среди ранних реализаций LZW существуют такие, которые увеличивают p до выдачи кода W, то есть код W, выдаваемый перед добавлением WK в словарь, уже занимает p+1 битов (что не является необходимым, так как код W меньше 2p). Такое поведение называется «ранним изменением» (early change). Эта путаница в реализациях побудила [Adobe](https://ru.wikipedia.org/wiki/Adobe) поддерживать оба варианта LZW в PDF (используются ли «ранние изменения», указывается с помощью специального флага в заголовке сжимаемых данных).

# Deflate

**Deflate** — это [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [сжатия без потерь](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C), использующий комбинацию алгоритмов [LZ77](https://ru.wikipedia.org/wiki/LZ77) и [Хаффмана](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0). Изначально был описан [Филом Кацем](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BB_%D0%9A%D0%B0%D1%86) для второй версии его архиватора [PKZIP](https://ru.wikipedia.org/wiki/PKZIP), который впоследствии был определён в [RFC 1951](https://tools.ietf.org/html/rfc1951) (1996 год).

Deflate считается свободным от всех существующих патентов, и пока оставался в силе патент на [LZW](https://ru.wikipedia.org/wiki/LZW) (он применяется в формате [GIF](https://ru.wikipedia.org/wiki/GIF)), это привело к использованию Deflate не только в формате [ZIP](https://ru.wikipedia.org/wiki/ZIP), для которого Кац изначально его спроектировал, но также в компрессоре/декомпрессоре [gzip](https://ru.wikipedia.org/wiki/Gzip) и в [PNG](https://ru.wikipedia.org/wiki/PNG)-изображениях.

## **Формат потока данных**

Deflate-поток содержит серии блоков. Перед каждым блоком находится трёхбитовый заголовок:

* Один бит: флаг последнего блока.
* 1: блок последний.
* 0: блок не последний.
* Два бита: метод, с помощью которого были закодированы данные.
* 00: данные не закодированы (в блоке находятся непосредственно выходные данные).
* 01: данные закодированы по методу [статического Хаффмана](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0).
* 10: данные закодированы по методу [динамического Хаффмана](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0).
* 11: зарезервированное значение (ошибка).

Большая часть блоков кодируется с помощью метода 10 (динамический Хаффман), который предоставляет оптимизированное дерево кодов Хаффмана для каждого нового блока. Инструкции для создания дерева кодов Хаффмана следуют непосредственно за заголовком блока.

Компрессия выполняется в два этапа:

* замена повторяющихся строк указателями (алгоритм LZ77);
* замена символов новыми символами, основываясь на частоте их использования (алгоритм Хаффмана).

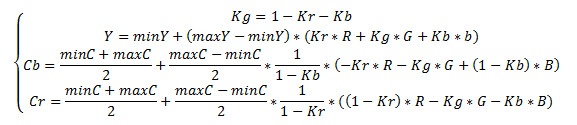
### Алгоритмы сжатия с потерями

* Наиболее популярным примером формата изображения, где используется сжатие с потерями, является [JPEG](https://ru.wikipedia.org/wiki/JPEG)
* На мобильных платформах применяется перевод изображения в [палитровый](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BB%D0%B8%D1%82%D1%80%D0%B0_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0)) формат.
* [JPEG 2000](https://ru.wikipedia.org/wiki/JPEG_2000)
* [Алгоритм фрактального сжатия](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%84%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F)
* [DXTC](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%BE%D0%BC%D0%BF%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%83%D1%80_S3_(DXTC)&action=edit&redlink=1) — компрессия текстур, реализованная в графическом API DirectX и поддерживаемая на аппаратном уровне современными видеокартами.
* [Дифференциальное сжатие](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D0%B5&action=edit&redlink=1) — сжатие основано на формировании граничных условий и выбора дифференциальных уравнений, решение выбранных дифференциальных уравнений, с вычисленными при сжатии граничными условиями, позволяет восстановить исходное изображение.

##### **Алгоритм сжатия JPEG**

JPEG на данный момент один из самых распространенных способов сжатия изображений с потерями. Опишем основные шаги, лежащие в основе этого алгоритма. Будем считать, что на вход алгоритма сжатия поступает изображение с глубиной цвета 24 бита на пиксел (изображение представлено в цветовой модели RGB).

###### **Перевод в цветовое пространство YCbCr**

В цветовой модели YCbCr мы представляем изображение в виде яркостной компоненты (Y) и двух цветоразностных компонент (Cb,Cr). Человеческий глаз более восприимчив к яркости, а не к цвету, поэтому алгоритм JPEG вносит по возможности минимальные изменения в яркостную компоненту (Y), а в цветоразностные компоненты могут вноситься значительные изменения. Перевод осуществляется по следующей формуле:  
[](https://habrastorage.org/storage/habraeffect/8c/14/8c143193049d8a3274a58b983de3fb3d.PNG)  
Выбор Kr и Kb зависит от оборудования. Обычно берётся Kb=0.114;Kr=0.299. В последнее время также используется Kb=0.0722;Kr=0.2126, что лучше отражает характеристики современных устройств отображения.

###### **Субдискретизация компонент цветности**

После перевода в цветовое пространство YCbCr выполняется дискретизация. Возможен один из трёх способов дискретизации:  
4

* :4:4 – отсутствует субдискретизация;
* 4:2:2 – компоненты цветности меняются через одну по горизонтали;
* 4:2:0 – компоненты цветности меняются через одну строку по горизонтали, при этом по вертикали они меняются через строку.

При использовании второго или третьего способа мы избавляется от 1/3 или 1/2 информации соответственно. Очевидно, что чем больше информации мы теряем, тем сильнее будут искажения в итоговом изображении.

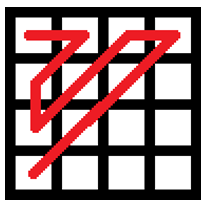
###### **Дискретное косинусное преобразование**

Изображение разбивается на компоненты 8\*8 пикселов, к каждой компоненте применятся ДКП. Это приводит к уплотнению энергии в коде. Преобразования применяются к компонентам независимо.

###### **Квантование**

Человек практически не способен замечать изменения в высокочастотных составляющих, поэтому коэффициенты, отвечающие за высокие частоты можно хранить с меньшей точностью. Для этого используется покомпонентное умножение (и округление) матриц, полученных в результате ДКП, на матрицу квантования. На данном этапе тоже можно регулировать степень сжатия (чем ближе к нулю компоненты матрицы квантования, тем меньше будет диапазон итоговой матрицы).

###### **Зигзаг-обход матриц**

Зигзаг-обход матрицы – это специальное направление обхода, представленное на рисунке:  
[](https://habrastorage.org/storage/habraeffect/20/10/2010488340013692050caac9f4b6a5cb.PNG)  
При этом для большинства реальных изображений в начале будут идти ненулевые коэффициенты, а ближе к концу будут идти нули.

###### **RLE- кодировние**

Используется особый вид RLE-кодирования: выводятся пары чисел, причём первое число в паре кодирует количество нулей, а второе – значение после последовательности нулей. Т.е. код для последовательности 0 0 15 42 0 0 0 44 будет следующим (2;15)(0;42)(3;44).

###### **Кодирование методом Хаффмана**

Используется описанный выше алгоритм Хаффмана. При кодировании используется заранее определённая таблица.  
Алгоритм декодирования заключается в обращении выполненных преобразований.  
К достоинствам алгоритма можно отнести высокую степень сжатие (5 и более раз), относительно невысокая сложность (с учётом специальных процессорных инструкций), патентная чистота. Недостаток – артефакты, заметные для человеческого глаза.

##### **Фрактальное сжатие**

Фрактальное сжатие – это относительно новая область. Фрактал – сложная геометрическая фигура, обладающая свойством самоподобия. Алгоритмы фрактального сжатия сейчас активно развиваются, но идеи, лежащие в их основе можно описать следующей последовательностью действий.  
Процесс сжатия:

1. Разделение изображения на неперекрывающиеся области (домены). Набор доменов должен покрывать всё изображение полностью.
2. Выбор ранговых областей. Ранговые области могут перекрываться и не покрывать целиком всё изображение.
3. Фрактальное преобразование: для каждого домена подбирается такая ранговая область, которая после аффинного преобразования наиболее точно аппроксимирует домен.
4. Сжатие и сохранение параметров аффинного преобразования. В файл записывается информация о расположении доменов и ранговых областей, а также сжатые коэффициенты аффинных преобразований.

Этапы восстановления изображения:

1. Создание двух изображений одинакового размера A и B. Размер и содержание областей не имеют значения.
2. Изображение B делится на домены так же, как и на первой стадии процесса сжатия. Для каждого домена области B проводится соответствующее аффинное преобразование ранговых областей изображения A, описанное коэффициентами из сжатого файла. Результат помещается в область B. После преобразования получается совершенно новое изображение.
3. Преобразование данных из области B в область A. Этот шаг повторяет шаг 3, только изображения A и B поменялись местами.
4. Шаги 3 и 4 повторяются до тех пор, пока изображения A и B не станут неразличимыми.

Точность полученного изображения зависит от точности аффинного преобразования.  
Сложность алгоритмов фрактального сжатия в том, что используется целочисленная арифметика и специальные довольно сложные методы, уменьшающие ошибки округления.  
Отличительной особенностью фрактального сжатия является его ярко выраженная ассиметрия. Алгоритмы сжатия и восстановления существенно различаются (сжатие требует гораздо большего количества вычислений).

Источники, дополнительные материалы:

<http://aco.ifmo.ru/el_books/image_processing/3_04.html>

<https://habr.com/ru/articles/116697/>

<https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9>

<https://youtu.be/IbxTg8S-W5s>