На данный момент, браузером [поддерживаются](https://caniuse.com/) два основных алгоритма сжатия текстовых файлов (css, html, js):

* Gzip
* Brotli

## Как работают алгоритмы сжатия gzip, brotli?

Перед тем как приступить к описанию приведённых алгоритмов, следует рассказать о семействе алгоритмов LZ, в частности LZ77, так как оба алгоритма основываются на нём.

Алгоритмы словарного сжатия Зива-Лемпела появились во второй половине 70-х гг. Это были так называемые алгоритмы LZ77 и LZ78, разработанные совместно Зивом (Ziv) и Лемпелом (Lempel). В дальнейшем первоначальные схемы подвергались множественным изменениям, в результате чего мы сегодня имеем десятки достаточно самостоятельных алгоритмов и бессчетное количество модификаций.

LZ77 и LZ78 являются универсальными алгоритмами сжатия, в которых словарь формируется на основании уже обработанной части входного потока, т. е. адаптивно. Принципиальным отличием является лишь способ формирования фраз.

**Алгоритм LZ77**

Алгоритм LZ77 является родоначальником целого семейства словарных схем - так называемых алгоритмов со скользящим словарем, или скользящим окном. Действительно, в LZ77 в качестве словаря используется блок уже закодированной последовательности. Как правило, по мере выполнения обработки положение этого блока относительно начала последовательности постоянно меняется, словарь "скользит" по входному потоку данных. Скользящее окно имеет длину N, т. е. в него помещается N символов, и состоит из двух частей:

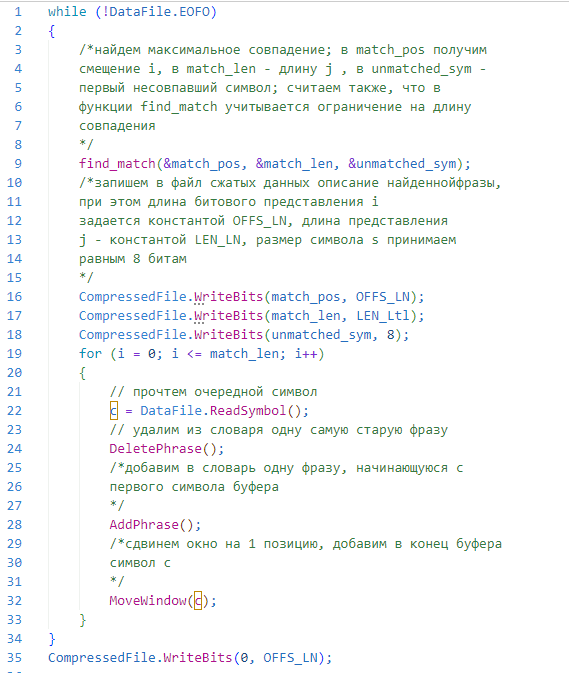
* последовательности длины W=N-n уже закодированных символов, которая и является словарем;
* упреждающего буфера, или буфера предварительного просмотра (lookahead), длины n; обычно n на порядки меньше W.

Пусть к текущему моменту времени мы уже закодировали t символов, последние W символом будут составлять наш словарь. На каждой итерации алгоритма мы ищем самое длинное вхождение префиксной строки упреждающего буфера, начиная с t+1 символа в словаре + упреждающем буфере, но важно, чтобы часть строки лежала в словаре. Полученная в результате поиска фраза кодируется с помощью двух чисел:

1. смещения (offset) от начала буфера
2. длины соответствия, или совпадения

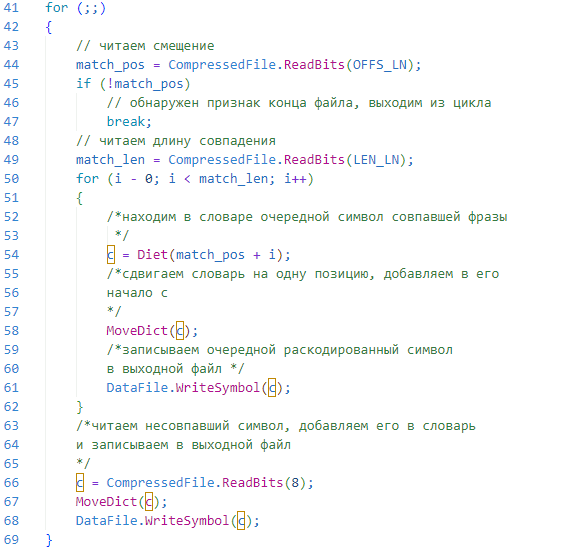
Смещение и длина соответствия играют роль указателя (ссылки), однозначно определяющего фразу. Дополнительно в выходной поток записывается символ s (подумайте зачем), непосредственно следующий за совпавшей строкой буфера.

Процесс кодирование можно описать следующим образом (Си)



Что касается декодирования сжатых данных, то оно осуществляется пу-тем простой замены кода на блок символов, состоящий из фразы словаря и явно передаваемого символа. Естественно, декодер должен выполнять Те же действия по изменению окна, что и кодер. Фраза словаря элементарно опре- деляется по смещению и длине, поэтому важным свойством LZ77 и прочих алгоритмов со скользящим окном является очень быстрая работа декодера.

Алгоритм декодирования может иметь следующий вид:



Алгоритмы со скользящим окном характеризуются сильной несимметричностью по времени - кодирование значительно медленнее декодирования, поскольку при сжатии много времени тратится на поиск фраз.

**Формат Deflate**

Формат словарного сжатия Deflate, предложенный Кацем (Katz), используется популярным архиватором GZIP. Сжатие осуществляется с помощью алгоритма типа LZH, иначе говоря, указатели и литералы кодируются по методу Хаффмана. Формат специфицирует только работу декодера, т. е. определяет алгоритм декодирования, и не налагает серьезных ограничений на реализацию кодера. В принципе в качестве алгоритма сжатия может применяться любой работающий со скользящим окном, лишь бы он исходил из стандартной процедуры обновления словаря для алгоритмов семейства LZ77 и использовал задаваемые форматом типы кодов Хаффмана

Закодированные в соответствии с форматом Deflate данные представляют собой набор блоков, порядок которых совпадает с последовательностью соответствующих блоков исходных данных. Используется 3 типа блоков закодированных данных:

1. состоящие из несжатых данных;
2. использующие фиксированные коды Хаффмана;
3. использующие динамические коды Хаффмана.

Длина блоков первого типа не может превышать 64 Кб, относительно других ограничений по размеру нет. Каждый блок типа 2 и 3 состоит из двух частей:

* описания двух таблиц кодов Хаффмана, использованных для кодирования данных блока;
* собственно закодированных данных.

Коды Хаффмана каждого блока не зависят от использованных в предыдущих блоках. Cамо описание динамически создаваемых кодов Хаффмана является, в свою очередь, также сжатым с помощью фиксированных кодов Хаффмана, таблица которых задается форматом.

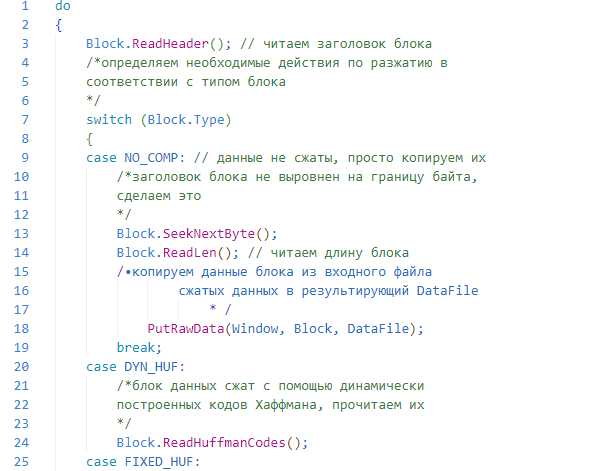
Алгоритм словарного сжатия может использовать в качестве словаря часть предыдущего блока (блоков), но величина смещения не может быть больше 32 Кб. Данные в компактном представлении состоят из кодов элементов двух типов:

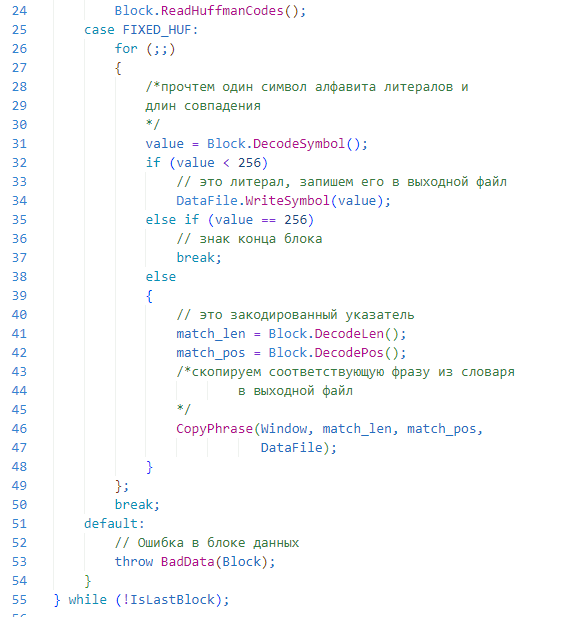
* литералов (одиночных символов);
* указателей имеющихся в словаре фраз; указатели состоят из пары <длина совпадения, смещение>

Длина совпавшей строки не может превышать 258 байт, а смещение фразы - 32 Кб. Литералы и длины совпадения кодируются с помощью одной таблицы кодов Хаффмана, а для смещений используется другая таблица; иначе говоря, литералы и длины совпадения образуют один алфавит. Именно эти таблицы кодов и передаются в начале блока третьего типа.

**Алгоритм декодирования Deflate**

Сжатые данные декодируются по следующему алгоритму:

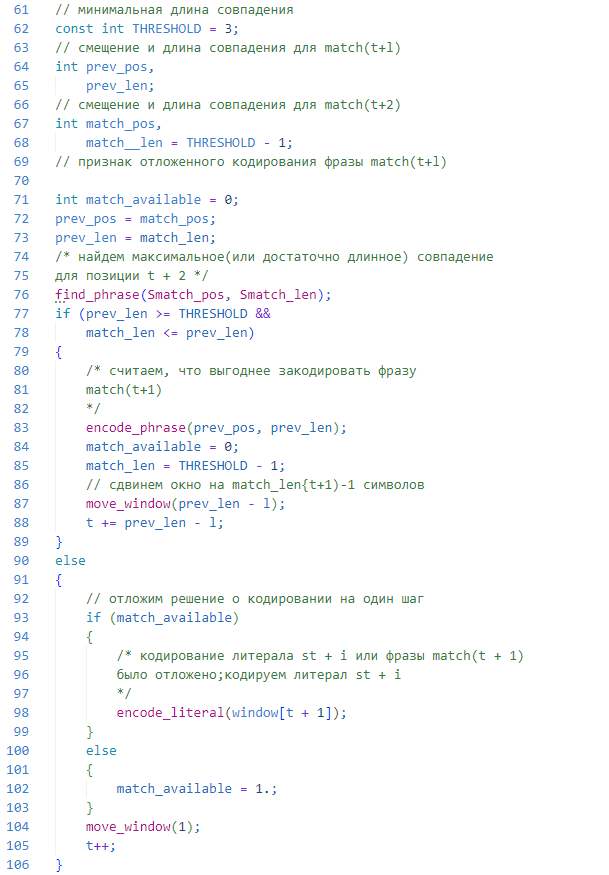




**Алгоритм словарного сжатия для DEFLATE**

Как уже указывалось, формат Deflate не имеет четкой спецификации алгоритма словарного сжатия. Разработчики могут использовать какие-то свои алгоритмы, подходящие для решения специфических задач.

В качестве примера приведу свободный от патентов алгоритм сжатия для Deflate, используемый в разрабатываемой Info-ZIP group утилите Zip.



**Формат сжатия Gzip (GNU zip)**

Это утилита для сжатия и распаковки файлов, которая широко используется в UNIX-системах.

Формат файла gzip состоит из 3 основных частей:

1. Заголовок: Содержит информацию о типе файла, имени оригинального файла, времени создания, уровне сжатия и других параметрах.
2. Тело: Содержит сжатые данные, выполненные с помощью алгоритма DEFLATE.
3. Контрольная сумма (CRC-32) и Размер оригинала: Эти данные предоставляют возможность проверки целостности и правильности распаковки данных.

Заголовок файла содержит следующие ключевые поля:

* - ID1 и ID2: Идентификаторы, указывающие на формат gzip (значения - 0x1F и 0x8B).
* - CM: Метод сжатия (в gzip используется значение 8 для DEFLATE). **\***
* - FLG: Биты флагов, которые указывают наличие дополнительных полей и информации.
* - MTIME: Время последней модификации оригинального файла.
* - XFL: Дополнительная информация о методе компрессии.
* - OS: Платформа, на которой был создан/сжат файл (gzip).

**\*** Другие значения для этого поля теоретически могут быть использованы для обозначения различных методов сжатия, но сам формат gzip и его стандартная реализация подразумевают использование только DEFLATE. На практике, если gzip файл содержит метод сжатия, отличный от DEFLATE, стандартные утилиты для работы с gzip файлами могут его не поддерживать и не распознать.

## Описание Brotli

В разработке…

## За счёт чего можно добиться разных степеней сжатия?

Т.к формат Deflate не имеет четкой спецификации алгоритма словарного сжатия, разработчики могут использовать различные модификации LZH, подбирая параметры для обеспечения желаемого соотношения скорости и коэффициента сжатия.

В приведённом примере алгоритма сжатия DEFLATE, можно менять match\_len(t+1) = L

## Тратиться ли больше времени на декодирование при высокой степени сжатия?

Согласно [замерам производительности](https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/), скорость декодирование для алгоритма gzip не зависит от степени сжатия, что является характерной особенностью семейства алгоритмов LZ77.

На самом деле, в реальных условиях более высокий уровень сжатия позволяет сократить время распаковки (что в первую очередь может быть связано с тем, что вам нужно обрабатывать меньше данных в постоянном хранилище и меньше обращаться к оперативной памяти).

## Сравнение алгоритмов на собственных данных

В разработке…

## Вывод для применения на практике

## Поскольку скорость декодирования не зависит от степени сжатия, разумно использовать максимальный уровень сжатия (CL) для статических файлов, таких как JS, CSS и HTML.

## Либо вообще не сжимать файлы, чтобы сэкономить вычислительные ресурсы компьютера. (Пока что не измерено. В разработке…)

## Источники

* Д.Ватолин, А.Ратушняк, М.Смирнов, В.Юкин - Методы сжатия данных
* <https://datatracker.ietf.org/doc/html/rfc1951> - RFC (Request for comments) deflate
* <https://datatracker.ietf.org/doc/html/rfc1952> - RFC gzip
* <https://datatracker.ietf.org/doc/html/rfc7932> - RFC brotley
* <https://www.npmjs.com/package/lz4>. – Библиотека js с реализованным алгоритмом LZ4 семейства LZ77

Сравнения алгоритмов по скорости компрессии декомпрессии, степени сжатия:

* <https://gist.github.com/Busyrev/6c2eb1651a8df961c1e6552116235a27>
* <https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/>
* <https://stackoverflow.com/questions/28452429/does-gzip-compression-level-have-any-impact-on-decompression>