Лекция 10: Алгоритмы семейства LZ77

А. М. Шур

Кафедра алгебры и фундаментальной информатики УрФУ

24 апреля 2020 г.

LZ77

LZ77 — семейство словарных алгоритмов сжатия, наследующих основную идею, предложенную в статье

- A. Lempel, J. Ziv. A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 1977

LZ77

LZ77 — семейство словарных алгоритмов сжатия, наследующих основную идею, предложенную в статье

- A. Lempel, J. Ziv. A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 1977
- текст разбивается на подстроки любой длины так, что каждая подстрока
  - либо (i) состоит из одного символа
  - либо (ii) встречается в предшествующем тексте
- подстроки из одного символа передаются “буквально”
- более длинные подстроки заменяются парой чисел (длина, позиция в тексте)
- последовательность, в которую преобразован текст, кодируется с использованием метода Хаффмана, арифметического кодирования или их вариаций

LZ77

LZ77 — семейство словарных алгоритмов сжатия, наследующих основную идею, предложенную в статье

- A. Lempel, J. Ziv. A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 1977

- текст разбивается на подстроки любой длины так, что каждая подстрока
  - либо (i) состоит из одного символа
  - либо (ii) встречается в предшествующем тексте
- подстроки из одного символа передаются “буквально”
- более длинные подстроки заменяются парой чисел (длина, позиция в тексте)
- последовательность, в которую преобразован текст, кодируется с использованием метода Хаффмана, арифметического кодирования или их вариаций

★ Изначально Лемпель и Зив предлагали в случае (ii) передавать, кроме пары чисел, еще и следующий символ, тем самым делая случай (i) подслучаем (ii) с длиной совпадения, равной 0; в практических реализациях от этого предложения отказались

Разбиение Лемпеля–Зива

Разбиением Лемпеля–Зива, или LZ-разбиением называется представление текста в виде

$$T = f_1 \cdot f_2 \ldots \cdot f_r,$$

в котором каждая фраза f_i является либо символом, не входящим в $f_1 \cdots f_{i-1}$, либо самым длинным префиксом $f_i \cdots f_r$, который имеет более раннее вхождение в T, то есть встречается в T начиная с некоторой позиции внутри $f_1 \cdots f_{i-1}$

Разбиение Лемпеля–Зива

Разбиением Лемпеля–Зива, или LZ-разбиением называется представление текста в виде

$$T = f_1 \cdot f_2 \ldots \cdot f_r,$$

в котором каждая фраза f_i является либо символом, не входящим в $f_1 \cdots f_{i-1}$, либо самым длинным префиксом $f_i \cdots f_r$, который имеет более раннее вхождение в T, то есть встречается в T начиная с некоторой позиции внутри $f_1 \cdots f_{i-1}$

Пример

Используем тот же текст $T = \text{БанБананана\#}$, что и в предыдущей лекции.
Его LZ-разбиение: $T = \text{Б} \cdot \text{а} \cdot \text{н} \cdot \text{Бан} \cdot \text{анана} \cdot \#$.

★ Фраза $f_5 = \text{анана}$ имеет более раннее вхождение, которое начинается внутри фразы f_4 и заканчивается уже внутри f_5; это допускается определением

Разбиение Лемпеля–Зива

Разбиением Лемпеля–Зива, или LZ-разбиением называется представление текста в виде

$$T = f_1 \cdot f_2 \ldots \cdot f_r,$$

в котором каждая фраза f_i является либо символом, не входящим в $f_1 \cdots f_{i-1}$, либо самым длинным префиксом $f_i \cdots f_r$, который имеет более раннее вхождение в T, то есть встречается в T начиная с некоторой позиции внутри $f_1 \cdots f_{i-1}$

Пример

Используем тот же текст $T = \text{БанБананана}\#$, что и в предыдущей лекции.
Его LZ-разбиение: $T = \text{Б} \cdot \text{а} \cdot \text{н} \cdot \text{Бан} \cdot \text{анана} \cdot \#$.

★ Фраза $f_5 = \text{анана}$ имеет более раннее вхождение, которое начинается внутри фразы f_4 и заканчивается уже внутри f_5; это допускается определением

★ LZ-разбиение позволяет закодировать текст в виде последовательности символов и ссылок на более ранние вхождения фраз

- Например, $\texttt{enc}(T) = \text{Б}, \text{а}, \text{н}, (3,1), (5,5), \#$
  - пара (i,j) означает "вставь сюда подстроку длины i, начинающуюся в позиции j"
  - если подстрока известна не полностью (как $(5,5)$ в примере), берется ее известный префикс (ан) и размножается до нужной длины i (до анана)

Разбиения типа LZ

Разбиением типа LZ называется любое представление текста в виде $T = f_1 \cdot f_2 \ldots \cdot f_s$, в котором каждая фраза f_i является либо символом, не входящим в $f_1 \cdots f_{i-1}$, либо префиксом $f_i \cdots f_s$, который имеет более раннее вхождение в T, то есть встречается в T начиная с некоторой позиции внутри $f_1 \cdots f_{i-1}$

- ★ По сравнению с определением LZ-разбиения убрано требование к f_i быть самым длинным

Разбиения типа LZ

Разбиением типа LZ называется любое представление текста в виде $T = f_1 \cdot f_2 \ldots \cdot f_s$, в котором каждая фраза f_i является либо символом, не входящим в $f_1 \cdots f_{i-1}$, либо префиксом $f_i \cdots f_s$, который имеет более раннее вхождение в T, то есть встречается в T начиная с некоторой позиции внутри $f_1 \cdots f_{i-1}$

★ По сравнению с определением LZ-разбиения убрано требование к f_i быть самым длинным

Лемма

Для произвольного текста T, LZ-разбиение имеет наименьшее число фраз среди всех разбиений типа LZ.

Доказательство. Пусть $T = f_1 \cdots f_r$ — LZ-разбиение, $T = g_1 \cdots g_s$ — какое-нибудь разбиение типа LZ. Покажем, что $r \leqslant s$, доказав по индукции более сильное утверждение: $|f_1 \cdots f_i| \geqslant |g_1 \cdots g_i|$ для любого $i = 1, \ldots, r$.

- База индукции ($i = 1$) очевидна, так как $f_1 = g_1 = T[1]$
- Пусть $|f_1 \cdots f_{i-1}| \geqslant |g_1 \cdots g_{i-1}|$
  - тогда $T = f_1 \cdots f_{i-1} u = g_1 \cdots g_{i-1} vu$ (возможно, $v = \lambda$)
  - если f_i — ранее не встречавшийся символ, то g_i либо целиком лежит в v, либо, при $v = \lambda$, совпадает с f_i
  - если f_i встречался ранее, а $f_i f_{i+1}[1]$ — нет, то $vf_i f_{i+1}[1]$ ранее не встречался, откуда g_i — префикс vf_i
- Таким образом, $|f_1 \cdots f_i| \geqslant |g_1 \cdots g_i|$ и шаг индукции доказан

Разбиения типа LZ (2)

Несмотря на Лемму, алгоритмы сжатия данных на основе LZ77 строят не LZ-разбиение, а разнообразные разбиения типа LZ. И на это есть причины...

Разбиения типа LZ (2)

Несмотря на Лемму, алгоритмы сжатия данных на основе LZ77 строят не LZ-разбиение, а разнообразные разбиения типа LZ. И на это есть причины...

- Декодеру вообще всё равно, как именно получена последовательность $\mathtt{enc}(T)$ — на декодирование это не влияет
- Простые алгоритмы быстро строят какое-нибудь разбиение типа LZ, разрешая искать предыдущие вхождения только внутри небольшого окна, содержащего недавно прочитанный фрагмент текста
- Продвинутые алгоритмы пытаются строить разбиение типа LZ, которое можно сжать лучше, чем LZ-разбиение
  - в паре (i, j) позицию j обычно задают через смещение, то есть расстояние $n - j$ до текущей позиции: совпадения часто бывают локальными, а маленькие числа занимают меньше места

  $\Rightarrow$ в некоторых случаях можно пожертвовать одним-двумя байтами длины i, если это приводит к очень большому уменьшению смещения; такие трюки применяются, например, в алгоритме LZMA

  ♠ ... Леденящие душу подробности про бит-оптимальные разбиения типа LZ можно прочитать, например, в статье D. Kosolobov. Relations between greedy and bit-optimal LZ77 encodings. STACS 2018

Разбиения типа LZ (2)

Несмотря на Лемму, алгоритмы сжатия данных на основе LZ77 строят не LZ-разбиение, а разнообразные разбиения типа LZ. И на это есть причины...

- Декодеру вообще всё равно, как именно получена последовательность $\mathtt{enc}(T)$ — на декодирование это не влияет
- Простые алгоритмы быстро строят какое-нибудь разбиение типа LZ, разрешая искать предыдущие вхождения только внутри небольшого окна, содержащего недавно прочитанный фрагмент текста
- Продвинутые алгоритмы пытаются строить разбиение типа LZ, которое можно сжать лучше, чем LZ-разбиение
  - в паре (i, j) позицию j обычно задают через смещение, то есть расстояние $n - j$ до текущей позиции: совпадения часто бывают локальными, а маленькие числа занимают меньше места

  $\Rightarrow$ в некоторых случаях можно пожертвовать одним-двумя байтами длины i, если это приводит к очень большому уменьшению смещения; такие трюки применяются, например, в алгоритме LZMA

  ♠ ... Леденящие душу подробности про бит-оптимальные разбиения типа LZ можно прочитать, например, в статье D. Kosolobov. Relations between greedy and bit-optimal LZ77 encodings. STACS 2018

Впрочем, LZ-разбиение тоже используется

- при построении сжатых индексов
  - сжатый индекс отличается от сжатого текста тем, что в нем можно решать задачи типа поиска заданной подстроки, не декодируя сжатое представление
- в качестве вычислительного примитива для комбинаторных задач
  - типа поиска повторяющихся структур в тексте

DEFLATE: простейший алгоритм семейства LZ77

Алгоритм DEFLATE придумал в 1980-е американский программист Фил Кац и реализовал его в архиваторе PKZIP для DOS; сейчас DEFLATE используется

- в архиваторах zip, gzip, 7-zip
- в форматах pdf, png, tiff, cab
- в кроссплатформенной библиотеке для сжатия zlib

DEFLATE: простейший алгоритм семейства LZ77

Алгоритм DEFLATE придумал в 1980-е американский программист Фил Кац и реализовал его в архиваторе PKZIP для DOS; сейчас DEFLATE используется

- в архиваторах zip, gzip, 7-zip
- в форматах pdf, png, tiff, cab
- в кроссплатформенной библиотеке для сжатия zlib

★ Важно: DEFLATE, в первую очередь, формат представления сжатых данных

- в построении разбиения типа LZ есть определенная свобода
- ограничение: фраза кодируется парой чисел (длина, смещение), где $3 \leqslant \text{длина} \leqslant 258$ и $1 \leqslant \text{смещение} \leqslant 32768$
- схема кодирования/декодирования построенного разбиения жестко фиксирована

DEFLATE: простейший алгоритм семейства LZ77

Алгоритм DEFLATE придумал в 1980-е американский программист Фил Кац и реализовал его в архиваторе PKZIP для DOS; сейчас DEFLATE используется

- в архиваторах zip, gzip, 7-zip
- в форматах pdf, png, tiff, cab
- в кроссплатформенной библиотеке для сжатия zlib

★ Важно: DEFLATE, в первую очередь, формат представления сжатых данных
- в построении разбиения типа LZ есть определенная свобода
- ограничение: фраза кодируется парой чисел (длина, смещение), где $3 \leqslant \text{длина} \leqslant 258$ и $1 \leqslant \text{смещение} \leqslant 32768$
- схема кодирования/декодирования построенного разбиения жестко фиксирована

- Текст можно разбивать на блоки произвольной длины, границы блоков должны быть выравнены по границам фраз
- Кодирование блока начинается с трехбитового заголовка, указывающего, является блок последним или нет, а также способ кодирования фраз блока:
  - нормальный — статический Хаффман, дерево строится по статистике фраз
  - странный — на фиксированном (известном декодеру) дереве префиксных кодов
  - буквальный — без сжатия (если попытка сжатия увеличит длину текста)
- Нормальный способ описан далее $\Longrightarrow$

★ Полное и подробное описание, включающее все опущенные в лекции подробности, есть тут: https://tools.ietf.org/html/rfc1951

DEFLATE: детали кодирования

Пусть нам нужно закодировать блоки $f_1, \ldots, f_s$, часть из которых — символы (символ=байт), а часть — пары чисел

- По $f_1, \ldots, f_s$ построим статическое дерево Хаффмана (лекция 3) с 288 листьями:
  - символы 0-255
  - символ конца блока 256
  - неиспользуемые символы 286-287 (так бывает при создании форматов данных:)
  - символы 257-285, кодирующие длины фраз по таблице

символ	доп.биты	длины	символ	доп.биты	длины	символ	доп.биты	длины
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

  - если код дает диапазон длин, последующие биты (количество приведено в таблице) дают сдвиг относительно нижней границы диапазона
  - ⋆ получается аналог кодов Элиаса, но код диапазона вычисляется по Хаффману
- Построим еще одно статическое дерево Хаффмана с 32 листьями: символы кодируют смещения от 1 до 32768, используя таблицу, аналогичную приведенной выше (см. `https://tools.ietf.org/html/rfc1951#page-11`)

DEFLATE: детали кодирования (2)

Так как коды Хаффмана нужно передавать декодеру, их нужно закодировать максимально компактно; для этого при построении деревьев Хаффмана добиваются выполнения двух дополнительных свойств:

1. если коды упорядочить лексикографически, то они упорядочатся по длине
   - коды $\{0, 10, 110, 111\}$ обладают этим свойством, а $\{00, 010, 011, 1\}$ — нет
2. из символов с кодами одной длины меньший символ имеет меньший код

DEFLATE: детали кодирования (2)

Так как коды Хаффмана нужно передавать декодеру, их нужно закодировать максимально компактно; для этого при построении деревьев Хаффмана добиваются выполнения двух дополнительных свойств:

1. если коды упорядочить лексикографически, то они упорядочатся по длине
   - коды $\{0, 10, 110, 111\}$ обладают этим свойством, а $\{00, 010, 011, 1\}$ — нет
2. из символов с кодами одной длины меньший символ имеет меньший код

Лемма

Если дерево Хаффмана для текста над алфавитом $[0..\sigma]$ обладает свойствами 1 и 2, его можно восстановить по последовательности $(\ell_0, \ldots, \ell_\sigma)$, где ℓ_i — длина кода Хаффмана символа i (неиспользуемым символам сопоставляется длина 0).

Доказательство: несложное упражнение.

DEFLATE: детали кодирования (2)

Так как коды Хаффмана нужно передавать декодеру, их нужно закодировать максимально компактно; для этого при построении деревьев Хаффмана добиваются выполнения двух дополнительных свойств:

1. если коды упорядочить лексикографически, то они упорядочатся по длине
   - коды $\{0, 10, 110, 111\}$ обладают этим свойством, а $\{00, 010, 011, 1\}$ — нет
2. из символов с кодами одной длины меньший символ имеет меньший код

Лемма

Если дерево Хаффмана для текста над алфавитом $[0..\sigma]$ обладает свойствами 1 и 2, его можно восстановить по последовательности $(\ell_0, \ldots, \ell_\sigma)$, где ℓ_i — длина кода Хаффмана символа i (неиспользуемым символам сопоставляется длина 0).

Доказательство: несложное упражнение.

Блок из фраз $f_1, \ldots, f_s$ кодируется следующим образом:

- кодируются оба построенных дерева
  - используя Лемму, коды Хаффмана передаются как последовательности длин; эти последовательности дополнительно сжаты при помощи кодирования повторов и. . . применения метода Хаффмана к тому, что получилось
- для каждой фразы f_i
  - если f_i — символ, передается его код Хаффмана
  - если $f_i = (\text{длина}, \text{смещение})$, передается код Хаффмана нужного диапазона длин (плюс дополнительные биты для вычисления сдвига, если они нужны), а за ним — код Хаффмана диапазона смещений (плюс биты для вычисления сдвига)

DEFLATE: поиск фраз

★ напомним, что DEFLATE ищет повторы длины не меньшей 3; смысл в том, что
- ⋆ повтор кодируется двумя кодами Хаффмана (+ доп. биты), а символ — одним
- ⇒ кодировать повтор длины 1 точно невыгодно, а повтор длины 2 иногда выгодно, иногда — невыгодно, но выгодность определится только в будущем (когда закончится блок и по нему будет построены деревья Хаффмана)

Рассмотрим поиск очередной фразы f начиная с текущей позиции $T[n]$:

DEFLATE: поиск фраз

★ напомним, что DEFLATE ищет повторы длины не меньшей 3; смысл в том, что
  - ⋆ повтор кодируется двумя кодами Хаффмана (+ доп. биты), а символ — одним
  - ⇒ кодировать повтор длины 1 точно невыгодно, а повтор длины 2 иногда выгодно, иногда — невыгодно, но выгодность определится только в будущем (когда закончится блок и по нему будет построены деревья Хаффмана)

Рассмотрим поиск очередной фразы f начиная с текущей позиции $T[n]$:

- строка $T[n-32768\,..\,n-1]$ хранится в кольцевом буфере
  - после нахождения f самые старые $|f|$ символов заменяются на $T[n..n+|f|-1]$
- для поиска используется хэш-таблица:
  - ключ: строка длины 3
  - значение: односвязный список позиций, в которых начинается эта строка, начиная с ближайшей к n

DEFLATE: поиск фраз

★ напомним, что DEFLATE ищет повторы длины не меньшей 3; смысл в том, что
- ⋆ повтор кодируется двумя кодами Хаффмана (+ доп. биты), а символ — одним
- ⇒ кодировать повтор длины 1 точно невыгодно, а повтор длины 2 иногда выгодно, иногда — невыгодно, но выгодность определится только в будущем (когда закончится блок и по нему будет построены деревья Хаффмана)

Рассмотрим поиск очередной фразы f начиная с текущей позиции $T[n]$:

- строка $T[n{-}32768\,..\,n{-}1]$ хранится в кольцевом буфере
  - после нахождения f самые старые $|f|$ символов заменяются на $T[n..n{+}|f|{-}1]$
- для поиска используется хэш-таблица:
  - ключ: строка длины 3
  - значение: односвязный список позиций, в которых начинается эта строка, начиная с ближайшей к n
- если по ключу $T[n..n{+}2]$ ничего не найдено, положим $f = T[n]$
  - добавим запись по ключу $T[n..n{+}2]$ в хэш-таблицу
- если по ключу найден список, бежим по нему, для каждого элемента сравнивая соответствующий участок буфера с $T[n..n{+}257]$ и выбираем длиннейшее совпадение в качестве f
  - обновим в хэш-таблице записи по ключам $T[n..n{+}2], \ldots, T[n{+}|f|{-}1..n{+}|f|{+}1]$
  - ⋆ из-за возможной коллизии хэшей, сравнивать надо начиная с $T[n]$, а не с $T[n{+}3]$
  - ⋆ для ускорения, удаление из хэш-таблицы не производится: сравнение со списком заканчивается, когда попадается позиция за пределами буфера

DEFLATE: ускорение поиска фраз

Медленная часть поиска: посимвольное сравнение с каждым из элементов цепи

Идея ускорения: использовать упорядоченность алфавита

DEFLATE: ускорение поиска фраз

Медленная часть поиска: посимвольное сравнение с каждым из элементов цепи
Идея ускорения: использовать упорядоченность алфавита

- Когда мы сравниваем $T[n..]$ с $T[i..]$, где i — первая позиция списка в хэш-таблице по ключу $T[n..n+2]$, мы узнаем не только длину совпадения, но и какая из двух строк лексикографически меньше
- Сохраняя эту информацию каждый раз, можно резко сократить число позиций, с которыми нужно будет сравнивать текущий текст:
  - ⋆ если $T[n..] < T[i..]$, то получить бóльшую длину совпадения можно только при сравнении с такой позицией j, что $T[j..] < T[i..]$

DEFLATE: ускорение поиска фраз

Медленная часть поиска: посимвольное сравнение с каждым из элементов цепи
Идея ускорения: использовать упорядоченность алфавита

- Когда мы сравниваем $T[n..]$ с $T[i..]$, где i — первая позиция списка в хэш-таблице по ключу $T[n..n+2]$, мы узнаем не только длину совпадения, но и какая из двух строк лексикографически меньше
- Сохраняя эту информацию каждый раз, можно резко сократить число позиций, с которыми нужно будет сравнивать текущий текст:
  - ⋆ если $T[n..] < T[i..]$, то получить бóльшую длину совпадения можно только при сравнении с такой позицией j, что $T[j..] < T[i..]$

Способ: вместо линейного списка хранить и обновлять бинарное дерево поиска

- если узел помечен позицией i, а его потомок — позицией j, то
  - $i > j$
  - если j находится в левом поддереве i, то $T[i..] > T[j..]$, иначе $T[i..] < T[j..]$

\+ число проверяемых позиций не превосходит глубины дерева

\+ после проверок дерево можно перестроить за время $O(1)$

− после нахождения фразы f, при обновлении хэш-таблицы по ключам $T[n+1..n+3], \ldots, T[n+|f|-1..n+|f|+1]$ невозможно обновить деревья (мы не производим сравнений для $T[n+1..], \ldots, T[n+|f|-1..]$), поэтому нужна “гибридная” схема между списком и периодически обновляемым деревом

DEFLATE: улучшение поиска фраз

Для того, чтобы построить разбиение типа LZ, которое лучше сжимается, в построении разбиения иногда используют следующий трюк:

- Если при обработке $T[n..]$ найден повтор f, то производится поиск повтора, начинающегося с позиции $n+1$
  - если найден более длинный повтор f', то в качестве очередной фразы вместо f записывается символ $T[n]$, после чего аналогично делается поиск с позиции $n+2$ и производится сравнение его длины с длиной f'
  - если более длинный повтор не найден, то записывается фраза f, а следующий поиск производится с позиции $n+|f|$

DEFLATE: улучшение поиска фраз

Для того, чтобы построить разбиение типа LZ, которое лучше сжимается, в построении разбиения иногда используют следующий трюк:

- Если при обработке $T[n..]$ найден повтор f, то производится поиск повтора, начинающегося с позиции $n+1$
  - если найден более длинный повтор f', то в качестве очередной фразы вместо f записывается символ $T[n]$, после чего аналогично делается поиск с позиции $n+2$ и производится сравнение его длины с длиной f'
  - если более длинный повтор не найден, то записывается фраза f, а следующий поиск производится с позиции $n+|f|$

★ Не вполне очевидно, почему этот трюк должен улучшать сжатие, но на практике улучшает

Оценка алгоритма DEFLATE

Плюсы:

- ★ быстрый универсальный алгоритм сжатия, сжимает несколько лучше LZW
- ★ декодер быстрее кодера во много раз:
  - основная трудоемкость — поиск повторов, декодер их не ищет
- ★ быстрота декодера способствует использованию в разных форматах файлов, как это и произошло с DEFLATE
- ★ вариативность — можно совершенствовать кодер при сохранении формата
  - сделано, например, в 7-Zip

Оценка алгоритма DEFLATE

Плюсы:

- ★ быстрый универсальный алгоритм сжатия, сжимает несколько лучше LZW
- ★ декодер быстрее кодера во много раз:
  - основная трудоемкость — поиск повторов, декодер их не ищет
- ★ быстрота декодера способствует использованию в разных форматах файлов, как это и произошло с DEFLATE
- ★ вариативность — можно совершенствовать кодер при сохранении формата
  - сделано, например, в 7-Zip

Минусы:

- ♠ как и у LZW — сжатие слабовато, усилия по его улучшению неизбежно замедляют кодер
  - тем не менее, существуют быстрые реализации (см. `http://mattmahoney.net/dc/text.html`), сжимающие 10^9 байт английской википедии в 3.1 раза

LZMA

- ★ Наиболее мощным алгоритмом сжатия из семейства LZ77 является LZMA авторства И. Павлова, используемый для дефолтного формата 7z архиватора 7-Zip, см. `https://www.7-zip.org`
- 7-Zip в режиме LZMA сжимает тот же датасет из википедии в 4.4 раза

LZMA

★ Наиболее мощным алгоритмом сжатия из семейства LZ77 является LZMA авторства И. Павлова, используемый для дефолтного формата 7z архиватора 7-Zip, см. `https://www.7-zip.org`

- 7-Zip в режиме LZMA сжимает тот же датасет из википедии в 4.4 раза

Основные особенности LZMA:

- Другой способ сжатия
  - последовательность фраз воспринимается кодером как последовательность бит и сжимается как результат работы марковского источника (Лекция 2) с контекстами достаточно большой длины (в битах!) арифметическим методом
- Другой формат представления последовательности фраз
  - при записи последовательности фраз перед символом ставится битовый флаг 0, а перед повтором — 1, за которым следуют еще битовые флаги для указания на то, что смещение совпадает с одним из последних использованных (в этом случае можно сэкономить на передаче смещения)
- Различные ухищрения, зависящие от конкретной реализации, для повышения качества разбиения типа LZ, которое строит алгоритм; в частности, повышение частоты использования недавних смещений

То, что не поместилось в лекции

...Построение LZ-разбиения с помощью суффиксного массива

- Можно почитать статьи, например, эту: J. Kärkkäinen, D. Kempa, S. J. Puglisi. Linear Time Lempel-Ziv Factorization: Simple, Fast, Small. CPM 2013
- Можно посмотреть эту лекцию: `https://www.lektorium.tv/lecture/26227`
- Суффиксные массивы используются в архиваторе ZPAQ: `http://mattmahoney.net/dc/zpaq.html`

...Построение разбиений типа LZ с помощью суффиксного дерева вместо хэш-таблиц

- Можно почитать, как модернизировать алгоритм Укконена, чтобы поддерживать суффиксное дерево для заданного окна в тексте: M. Senft. Suffix tree for a sliding window: an overview. WDS'05