Лекция 9: Словарные методы. Алгоритм LZW

А. М. Шур

Кафедра алгебры и фундаментальной информатики УрФУ

14 апреля 2020 г.

Словарные алгоритмы

Словарные алгоритмы реализуют следующий подход к сжатию текста:

- текст рассматривается не как последовательность символов, а как последовательность подстрок переменной длины
- поддерживается структура данных (словарь), содержащая некоторое количество строк
- на каждой итерации алгоритм находит некоторый префикс необработанной части текста, который присутствует в словаре, и заменяет этот префикс ссылкой на словарь
- последовательность ссылок, в которую преобразован текст, дополнительно сжимается с использованием стандартных методов кодирования (например, Хаффмана и/или арифметики)

Словарные алгоритмы

Словарные алгоритмы реализуют следующий подход к сжатию текста:

- текст рассматривается не как последовательность символов, а как последовательность подстрок переменной длины
- поддерживается структура данных (словарь), содержащая некоторое количество строк
- на каждой итерации алгоритм находит некоторый префикс необработанной части текста, который присутствует в словаре, и заменяет этот префикс ссылкой на словарь
- последовательность ссылок, в которую преобразован текст, дополнительно сжимается с использованием стандартных методов кодирования (например, Хаффмана и/или арифметики)

★ В узкоспециализированном случае можно использовать статический словарь

- например, мы сжимаем только тексты на английском языке, берем фиксированный (известный декодеру) английский словарь, рассматриваем текст как последовательность слов из словаря (плюс знаки препинания и специальное слово для обработки исключений), сжимаем эту последовательность Хаффманом
- словаря на 65500 слов (2 байта на номер слова) достаточно, чтобы свести исключения к минимуму; при средней длине английского слова в 4.79 буквы (`http://norvig.com/mayzner.html`) это означает более чем двукратное сжатие даже без Хаффмана

Лемпель–Зив

В дальнейшем речь пойдет только о динамических словарях, обновляемых кодером и декодером параллельно с обработкой текста

Лемпель–Зив

В дальнейшем речь пойдет только о динамических словарях, обновляемых кодером и декодером параллельно с обработкой текста

Идея кодирования при помощи отсылок к динамическому словарю была предложена в работе

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

двумя израильскими учеными с сугубо библейскими именами:

Слева—
Иаков Зив,

Справа —
Авраам Лемпель

Аббревиатура LZ объединяет фактически все динамические словарные алгоритмы сжатия

Два подхода: LZ77 и LZ78

Лемпель и Зив предложили два существенно различных алгоритма

- то, что сейчас называют LZ77 — в названной статье 1977 года
- то, что называют LZ78 — в статье Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 1978

Два подхода: LZ77 и LZ78

Лемпель и Зив предложили два существенно различных алгоритма

- то, что сейчас называют LZ77 — в названной статье 1977 года
- то, что называют LZ78 — в статье Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 1978

Принципиальное различие между LZ77 и LZ78:

- в LZ78 словарь строится явно: по ходу чтения текста туда добавляются некоторые его подстроки; при следующем вхождении подстроки она может быть закодирована своим номером в словаре
- в LZ77 роль словаря играет одна длинная строка: прочитанная часть текста или некоторый ее суффикс; если подстрока встречается в словаре, то ее кодируют парой чисел — позиция в словаре и длина

Два подхода: LZ77 и LZ78

Лемпель и Зив предложили два существенно различных алгоритма

- то, что сейчас называют LZ77 — в названной статье 1977 года
- то, что называют LZ78 — в статье Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 1978

Принципиальное различие между LZ77 и LZ78:

- в LZ78 словарь строится явно: по ходу чтения текста туда добавляются некоторые его подстроки; при следующем вхождении подстроки она может быть закодирована своим номером в словаре
- в LZ77 роль словаря играет одна длинная строка: прочитанная часть текста или некоторый ее суффикс; если подстрока встречается в словаре, то ее кодируют парой чисел — позиция в словаре и длина

★ LZ78 проще для понимания, но сжимает хуже;
- Далее в этой лекции рассказывается единственный практический вариант LZ78 (используется в формате GIF, присутствует как опция в TIFF и PDF)

★ LZ77 сложнее эффективно реализовать, но его потенциал гораздо больше
- даже самая простая реализация, называемая DEFLATE, используется в форматах CAB, PNG, PDF, в архиваторах Zip, gzip, WinRaR, 7-Zip и некоторых других
- LZ77 рассматривается в следующей лекции

Алгоритм LZW

Алгоритм LZW, являющийся модификацией LZ78, предложен Уэлшем в 1984 г.

- Текст рассматривается как последовательность байт
- Словарь D инициализируется всеми односимвольными строками
  - по умолчанию изначально в нем есть элементы с номерами $0, 1, \ldots, 255$
- Добавляемые элементы получают номера по возрастанию до достижения верхнего ограничения на размер словаря (например, 4096)
  - по достижении максимального размера словарь перестает обновляться
  - вариант: словарь сбрасывается до исходного

Алгоритм LZW

Алгоритм LZW, являющийся модификацией LZ78, предложен Уэлшем в 1984 г.

- Текст рассматривается как последовательность байт
- Словарь D инициализируется всеми односимвольными строками
  - по умолчанию изначально в нем есть элементы с номерами $0, 1, \ldots, 255$
- Добавляемые элементы получают номера по возрастанию до достижения верхнего ограничения на размер словаря (например, 4096)
  - по достижении максимального размера словарь перестает обновляться
  - вариант: словарь сбрасывается до исходного

Рассмотрим итерацию кодера, на которой обрабатывается символ $T[n] = a$

- К началу итерации для некоторого $i < n$ имеем:
  - строка $T[1..i-1]$ закодирована в $\texttt{enc}(T)$ как последовательность элементов D
  - строка $w = T[i..n-1]$ присутствует в D

Итерация:

- если $wa \in D$
  - положить $w = wa$
- иначе
  - добавить к $\texttt{enc}(T)$ номер элемента w в D
  - добавить wa в словарь
  - положить $w = a$

Алгоритм LZW

Алгоритм LZW, являющийся модификацией LZ78, предложен Уэлшем в 1984 г.

- Текст рассматривается как последовательность байт
- Словарь D инициализируется всеми односимвольными строками
  - по умолчанию изначально в нем есть элементы с номерами $0, 1, \ldots, 255$
- Добавляемые элементы получают номера по возрастанию до достижения верхнего ограничения на размер словаря (например, 4096)
  - по достижении максимального размера словарь перестает обновляться
  - вариант: словарь сбрасывается до исходного

Рассмотрим итерацию кодера, на которой обрабатывается символ $T[n] = a$

- К началу итерации для некоторого $i < n$ имеем:
  - строка $T[1..i-1]$ закодирована в $\texttt{enc}(T)$ как последовательность элементов D
  - строка $w = T[i..n-1]$ присутствует в D

Итерация:

- если $wa \in D$
  - положить $w = wa$
- иначе
  - добавить к $\texttt{enc}(T)$ номер элемента w в D
  - добавить wa в словарь
  - положить $w = a$

★ Для завершения работы кодеру нужно либо знание длины текста до начала кодирования, либо спецсимвол конца текста
  - в примере будем рассматривать второй вариант $\Longrightarrow$

Алгоритм LZW — пример кодирования

Пример

Закодируем текст 'БанБананана#' (# в роли символа окончания текста). Пусть начальный словарь содержит статьи 0..255 в соответствии с кодировкой ср1251.
Последовательность итераций:

1. (подготовительная) $\mathtt{enc} = \lambda$, $w = \text{Б}$
2. $\text{Ба} \notin D$: $\mathtt{enc} = [193]$, $D[256] = \text{Ба}$, $w = \text{а}$
3. $\text{ан} \notin D$: $\mathtt{enc} = [193, 224]$, $D[257] = \text{ан}$, $w = \text{н}$
4. $\text{нБ} \notin D$: $\mathtt{enc} = [193, 224, 237]$, $D[258] = \text{нБ}$, $w = \text{Б}$
5. $\text{Ба} \in D$: $w = \text{Ба}$
6. $\text{Бан} \notin D$: $\mathtt{enc} = [193, 224, 237, 256]$, $D[259] = \text{Бан}$, $w = \text{н}$
7. $\text{на} \notin D$: $\mathtt{enc} = [193, 224, 237, 256, 237]$, $D[260] = \text{на}$, $w = \text{а}$
8. $\text{ан} \in D$: $w = \text{ан}$
9. $\text{ана} \notin D$: $\mathtt{enc} = [193, 224, 237, 256, 237, 257]$, $D[261] = \text{ана}$, $w = \text{а}$
10. $\text{ан} \in D$: $w = \text{ан}$
11. $\text{ана} \in D$: $w = \text{ана}$
12. (обработка #): $\mathtt{enc} = [193, 224, 237, 256, 237, 257, 261, 35]$

Представление закодированного текста

Как компактно представить $\mathtt{enc}(T)$, т.е. последовательность номеров словарных статей?

Представление закодированного текста

Как компактно представить $\mathtt{enc}(T)$, т.е. последовательность номеров словарных статей?

В базовом варианте LZW используется следующая простая идея:

- Каждый номер кодируется минимально необходимым числом бит (логарифмом от длины словаря); более точно
  - на первый номер нужно 8 бит ($|D| = 256$)
  - одновременно с первым номером в $\mathtt{enc}(T)$ появилась новая статья в D, и с этого момента на элемент словаря перестало хватать 8 бит, нужно 9; значит, со следующей итерации все добавляемые номера представляются 9 битами
  - когда число статей превысит 512, далее используется 10 бит на номер, и т.д.

★ Нужно быть внимательным с форматами представления чисел

- является ли левый бит в байте старшим или младшим?

Представление закодированного текста

Как компактно представить $\mathtt{enc}(T)$, т.е. последовательность номеров словарных статей?

В базовом варианте LZW используется следующая простая идея:

- Каждый номер кодируется минимально необходимым числом бит (логарифмом от длины словаря); более точно
  - на первый номер нужно 8 бит ($|D| = 256$)
  - одновременно с первым номером в $\mathtt{enc}(T)$ появилась новая статья в D, и с этого момента на элемент словаря перестало хватать 8 бит, нужно 9; значит, со следующей итерации все добавляемые номера представляются 9 битами
  - когда число статей превысит 512, далее используется 10 бит на номер, и т.д.

★ Нужно быть внимательным с форматами представления чисел

- является ли левый бит в байте старшим или младшим?

Сделать представление $\mathtt{enc}(T)$ более компактным можно, "дожав" его динамическим алгоритмом Хаффмана (лекция 4)

★ для исходного кода впервые встретившегося символа используется минимальное количество бит по правилу, указанному выше

Алгоритм LZW — декодирование

Декодирование почти симметрично кодированию

- ★ разница в том, что кодер одновременно кодирует w и кладет wa в словарь, а декодер может положить wa в словарь только на следующей итерации, когда раскодирует следующую за w подстроку
- ★ имеющаяся асимметрия создает особый случай, который разберем отдельно

Алгоритм LZW — декодирование

Декодирование почти симметрично кодированию

- ★ разница в том, что кодер одновременно кодирует w и кладет wa в словарь, а декодер может положить wa в словарь только на следующей итерации, когда раскодирует следующую за w подстроку
- ★ имеющаяся асимметрия создает особый случай, который разберем отдельно
- Инициализация при декодировании такая же, как при кодировании

Рассмотрим итерацию декодера, на которой обрабатывается символ $\mathtt{enc}(T)[i] = x$

- Пусть w — строка, взятая из словаря на предыдущей итерации

Итерация:

- если строка $D[x]$ существует, положить $v = D[x]$, иначе $v = w \cdot w[1]$
- добавить v к T
- добавить $w \cdot v[1]$ в словарь
- положить $w = v$

Алгоритм LZW — декодирование

Декодирование почти симметрично кодированию

- ★ разница в том, что кодер одновременно кодирует w и кладет wa в словарь, а декодер может положить wa в словарь только на следующей итерации, когда раскодирует следующую за w подстроку
- ★ имеющаяся асимметрия создает особый случай, который разберем отдельно
- Инициализация при декодировании такая же, как при кодировании

Рассмотрим итерацию декодера, на которой обрабатывается символ $\texttt{enc}(T)[i] = x$

- Пусть w — строка, взятая из словаря на предыдущей итерации

Итерация:

- если строка $D[x]$ существует, положить $v = D[x]$, иначе $v = w \cdot w[1]$
- добавить v к T
- добавить $w \cdot v[1]$ в словарь
- положить $w = v$
- Особый случай — это когда $D[x]$ не определено
  - кодер определил $D[x]$ на предыдущем шаге, иначе декодер тоже его определил бы
  - строка, дописанная кодером к $\texttt{enc}(T)$ на том же шаге, равна w (по определению)

  $\Rightarrow$ $D[x] = wb$ для некоторой буквы b
  $\Rightarrow$ $D[x]$ еще и начинается с b
  $\Rightarrow$ $b = w[1]$
  $\Rightarrow$ $D[x] = w \cdot w[1]$, т.е. v выше определено корректно

Алгоритм LZW — пример декодирования

Пример

Раскодируем последовательность $\mathtt{enc} = [193, 224, 237, 256, 237, 257, 261, 35]$, полученную в примере на слайде 6.

Последовательность итераций:

1. $v = D[193] = \text{Б}$, $T = \text{Б}$, $w = \text{Б}$
2. $v = D[224] = \text{а}$, $T = \text{Ба}$, $D[256] = \text{Ба}$, $w = \text{а}$
3. $v = D[237] = \text{н}$, $T = \text{Бан}$, $D[257] = \text{ан}$, $w = \text{н}$
4. $v = D[256] = \text{Ба}$, $T = \text{БанБа}$, $D[258] = \text{нБ}$, $w = \text{Ба}$
5. $v = D[237] = \text{н}$, $T = \text{БанБан}$, $D[259] = \text{Бан}$, $w = \text{н}$
6. $v = D[257] = \text{ан}$, $T = \text{БанБанан}$, $D[260] = \text{на}$, $w = \text{ан}$
7. особый случай: $D[261]$ не определено, $v = w{\cdot}w[1] = \text{ана}$, $T = \text{БанБананана}$, $D[261] = \text{ана}$, $w = \text{ана}$
8. $v = D[35] = \#$, $T = \text{БанБананана}\#$

Эффективная реализация

Хранение словаря

★ По построению, если $w = a_1 \cdots a_{i-1} a_i \in D$, то $a_1, \ldots, a_1 \cdots a_{i-1} \in D$

- значит, словарь можно хранить как префиксное дерево
- (см. бор в лекции 2, но здесь нам нужно само дерево, а не автомат)
- каждый узел обозначает строку, читаемую на пути от корня до него

Эффективная реализация

Хранение словаря

- ★ По построению, если $w = a_1 \cdots a_{i-1} a_i \in D$, то $a_1, \ldots, a_1 \cdots a_{i-1} \in D$
  - значит, словарь можно хранить как префиксное дерево
  - (см. бор в лекции 2, но здесь нам нужно само дерево, а не автомат)
  - каждый узел обозначает строку, читаемую на пути от корня до него
- Для каждого узла храним номер в словаре и переходы к детям
  - все переходы можно хранить в одной хэш-таблице, которая по ключу — паре (номер строки w, символ a) — возвращает номер строки wa
- В начале итерации кодер (см. слайд 5) находится в узле, обозначающем w, и пытается перейти в узел wa
- При попадании в пустую ячейку хэш-таблицы кодер
  - заносит туда номер нового элемента словаря (равный $|D|$)
  - берет код символа a в качестве номера новой строки w
- ★ Декодеру никакие хитрости не нужны:
  - поддерживает словарь как массив строк (индекс в массиве = номер в словаре)
  - хранит вместо текущего w его номер

Эффективная реализация

Хранение словаря

- ★ По построению, если $w = a_1 \cdots a_{i-1} a_i \in D$, то $a_1, \ldots, a_1 \cdots a_{i-1} \in D$
  - значит, словарь можно хранить как префиксное дерево
  - (см. бор в лекции 2, но здесь нам нужно само дерево, а не автомат)
  - каждый узел обозначает строку, читаемую на пути от корня до него
- Для каждого узла храним номер в словаре и переходы к детям
  - все переходы можно хранить в одной хэш-таблице, которая по ключу — паре (номер строки w, символ a) — возвращает номер строки wa
- В начале итерации кодер (см. слайд 5) находится в узле, обозначающем w, и пытается перейти в узел wa
- При попадании в пустую ячейку хэш-таблицы кодер
  - заносит туда номер нового элемента словаря (равный $|D|$)
  - берет код символа a в качестве номера новой строки w
- ★ Декодеру никакие хитрости не нужны:
  - поддерживает словарь как массив строк (индекс в массиве = номер в словаре)
  - хранит вместо текущего w его номер

Оценка времени работы

- И кодер, и декодер совершают константное количество операций на итерацию (включая операции со словарем)
  - если использовать, например, `https://en.wikipedia.org/wiki/Cuckoo_hashing`, поиск по ключу занимает константное время, а вставка — константное в среднем

Оценка алгоритма LZW

Плюсы:

- ★ быстрый универсальный алгоритм сжатия
- ★ декодер быстрее кодера в несколько раз:
  - меньше итераций
  - не нужна хэш-таблица
- ★ сверхбыстрый декодер нужен всегда, когда однократному сжатию соответствует многократное декодирование
  - в случае LZW — создание и просмотр GIF-ки

Оценка алгоритма LZW

Плюсы:

- ★ быстрый универсальный алгоритм сжатия
- ★ декодер быстрее кодера в несколько раз:
  - меньше итераций
  - не нужна хэш-таблица
- ★ сверхбыстрый декодер нужен всегда, когда однократному сжатию соответствует многократное декодирование
  - в случае LZW — создание и просмотр GIF-ки

Минусы:

- ♠ слабое сжатие
  - стандартная unix-утилита compress, использующая LZW, сжимает 10^9 байт из английской википедии менее чем в 2.4 раза (см. `http://mattmahoney.net/dc/text.html`)
- ♠ если несколько улучшить сжатие применением к $\mathrm{enc}(T)$ алгоритма Хаффмана, то ухудшится время работы
  - Хаффман на большом алфавите будет работать медленнее, чем LZW, а декодер у него работает примерно с той же скоростью, что и кодер