

Урок №7

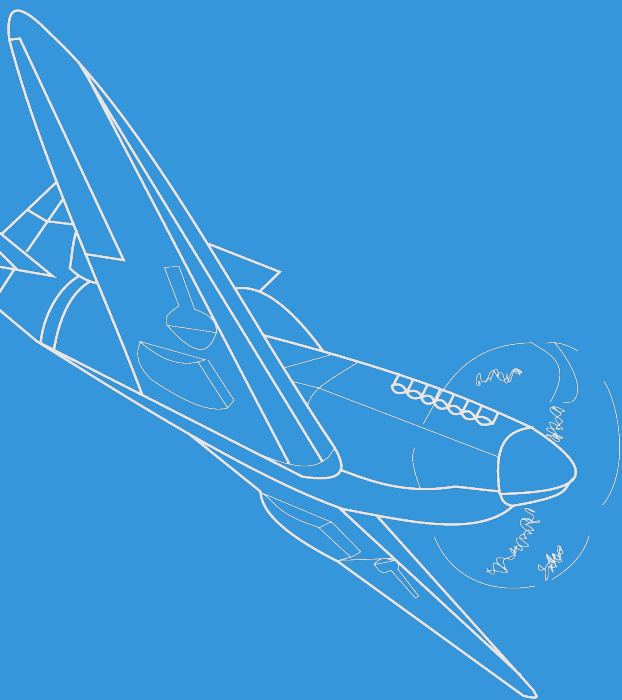
# Сжатие координатных блоков

(основано на слайдах Андрея Калинина, Hinrich Schütze,  
Christina Lioma)

---

# Содержание занятия

1. Введение
2. Сжатие словаря
3. Сжатие координатных блоков



# Введение

# Зачем сжимать?



- Меньше места
  - Экономим деньги
- Больше помещается в память
  - Увеличивается скорость
- Увеличивается скорость передачи данных между диском и оперативной памятью
  - [прочитать сжатое | распаковать ] быстрее, чем [прочитать несжатое]
  - Это выполняется, если алгоритмы распаковки быстрые
    - Поэтому будем рассматривать специальные методы

# Зачем сжимать индекс?



- Словарь
  - Добиться, чтобы словарь поместился в память
  - Уменьшить настолько, чтобы освободить место под координатные блоки
- Координатные блоки
  - Уменьшить объём на диске
  - Уменьшить время чтения с диска
  - Крупные поисковики хранят большую часть КБ в памяти
    - Сжатие позволяет хранить больше.
- Рассмотрим несколько специфичных методов сжатия

# Корпус новостей Reuters RCV1



N	документы	800,000
L	токенов на документ	200
M	термины	400,000
	байтов на токен (с пробелами и пункт.)	6
	байтов на токен (без проблов/пункт.)	4.5
	байтов на термин	7.5
T	постингов без координат	100,000,000

# Связь размера индекса и параметров индексации



Размер ...	терминов			постингов			координат		
	словарь			индекс без координат			координатный индекс		
	Размер (K)	Δ %	всего %	Размер (K)	Δ %	всего %	Размер (K)	Δ %	всего%
Всё	484			109,971			197,879		
Без чисел	474	-2	-2	100,680	-8	-8	179,158	-9	-9
Капитализация	392	-17	-19	96,969	-3	-12	179,158	0	-9
30 стоп-слов	391	-0	-19	83,390	-14	-24	121,858	-31	-38
150 стоп-слов	391	-0	-19	67,002	-30	-39	94,517	-47	-52
Без окончаний	322	-17	-33	63,812	-4	-42	94,517	0	-52

# Сжатие с потерями и без



- Сжатие без потерь: вся информация остаётся как есть.
  - Обычно используем её в ИП.
- Сжатие с потерями: Что-то считаем возможным убрать
- Понижение капитализации, стоп-слова, морф. нормализация – может рассматриваться как сжатие с потерями.
- Ещё – удаление координат для позиций, которые вряд ли будут вверху на ранжировании.



# Размер словаря относительно размера корпуса (1)



- Насколько велик словарь?
  - Т.е., сколько разных слов?
- Можем ли найти верхнюю границу?
  - Нет:  $70^{20} = 10^{37}$  разных слов длины 20
- На практике, размер словаря растёт с размером корпуса
  - Особенно для Unicode

# Размер словаря относительно размера корпуса (2)



- Закон Хипса:  $M = kT^\beta$
- $M$  – размер словаря,  $T$  – количество токенов
- Обычно:  $30 \leq k \leq 100$  и  $\beta \approx 0.5$
- На логарифмическом графике (log-log)  $M$  от  $T$ , этот закон предсказывает линию под углом  $1/2$ 
  - Простейшая связь в log-log
  - Эмпирический закон (основанный на наблюдениях)

# Закон Хипса



Для RCV1 пунктирная линия

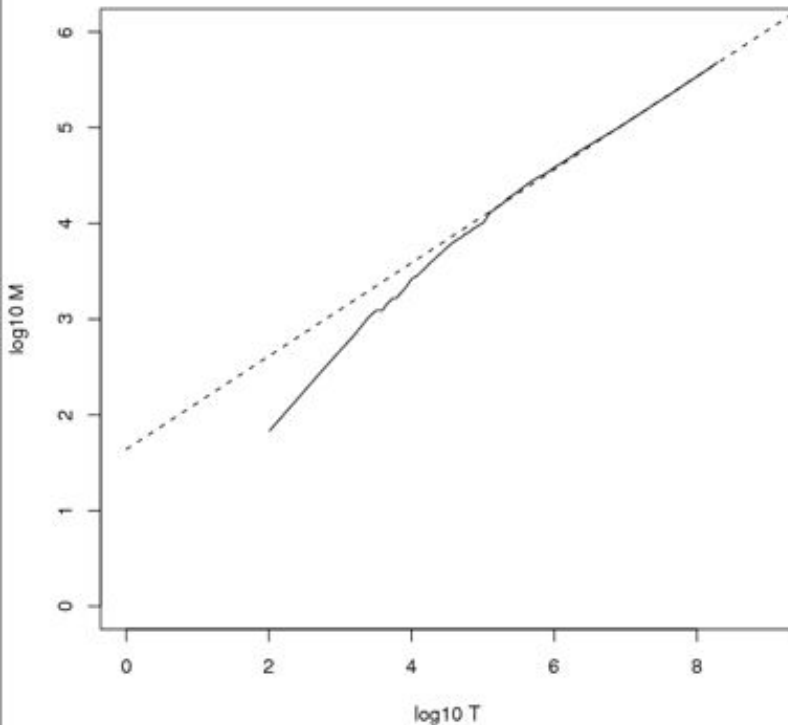
$\log_{10} M = 0.49 \log_{10} T + 1.64$  МНК-оптимизация  
закона.

Так,  $M = 10^{1.64} T^{0.49}$  отсюда  $k = 10^{1.64} \approx 44$  и  $\beta = 0.49$ .

Хорошее предсказание для Reuters RCV1!

Для первых 1,000,020 токенов закон  
предсказывает 38,323 терминов;

На самом деле – 38,365



# Закон Ципфа



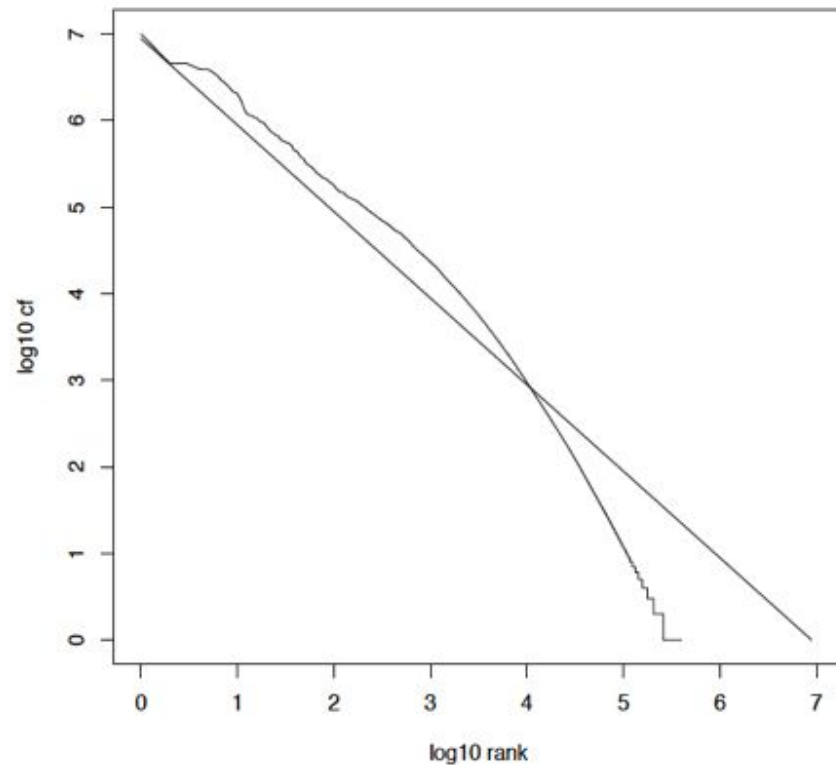
- Нас интересуют относительные частотности терминов.
- В естественном языке немного частых терминов и много редких.
- Закон Ципфа:  $i$ -ый самый частый термин имеет частоту  $\sim 1/i$ .
- $cf_i \propto 1/i = K/I$  где  $K$  – нормализующая константа
- $cf_i$  – корпусная частота, сколько раз  $t_i$  встретился в корпусе

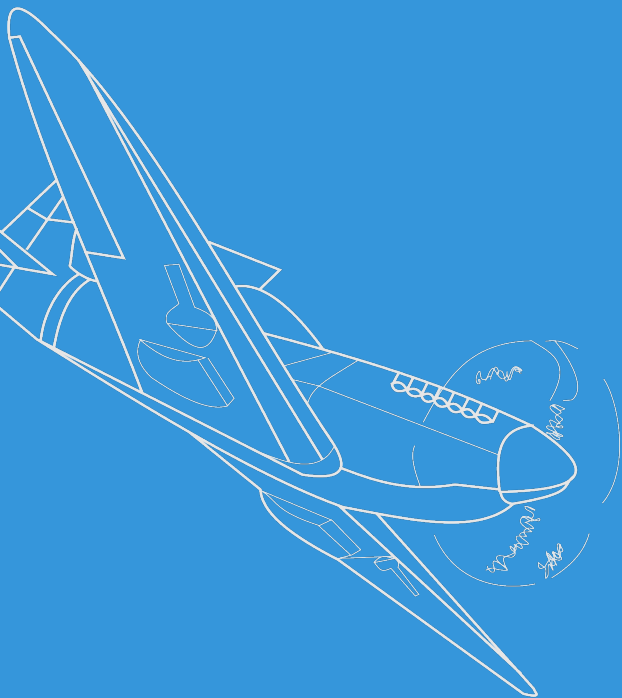
# Следствия из закона Ципфа



- Если самый частый терм (*the*, для английского) встречается  $cf_1$  раз
- Тогда следующий самый частотный термин (*of*) встречается  $cf_1/2$  раз
- Третий (*and*) –  $cf_1/3$  ...
- Т.е.:  $cf_i = K/i$ , где  $K$  – нормализующий коэффициент, тогда
- $\log cf_i = \log K - \log i$
- Линейная зависимость между  $\log cf_i$  и  $\log i$
- Ещё один пример степенного закона

# Закон Ципфа для Reuters RCV1





# Сжатие словаря

# Задача: построить обратный индекс



Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173

Calpurnia → 2 → 31 → 54 → 101

Caeser → 1 → 2 → 4 → 5 → 6 → 16 → ...

Словарь

Координаты



# Зачем сжимать словарь?

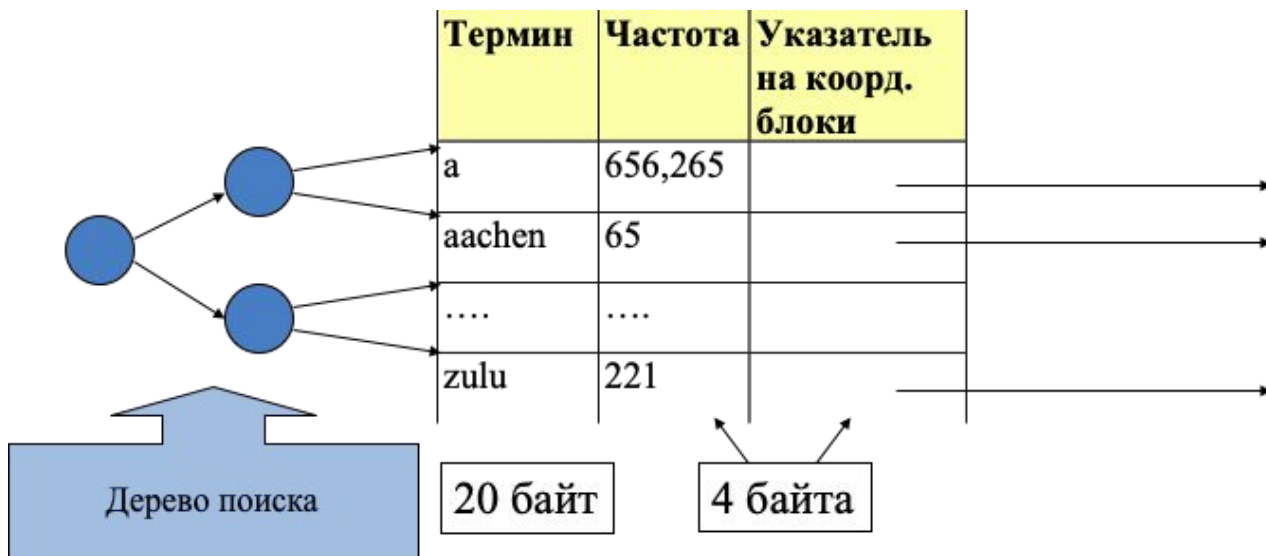


- Поиск начинается со словаря.
- Мы хотим хранить его в памяти.
- При этом мы хотим занять как можно меньше памяти (освободить её для других задач)
- Мобильные устройства могут иметь мало памяти.
- Даже если словарь в память не поместился, мы всё равно хотим быстро его загружать при обращениях.
- Итак, сжатие словаря – важная задача.

# Хранение словаря – первый подход



- Массив структур фиксированного размера
  - ~400,000 терминов; 28 байтов на термин = 11.2 MB.



# Фиксированная длина терминов

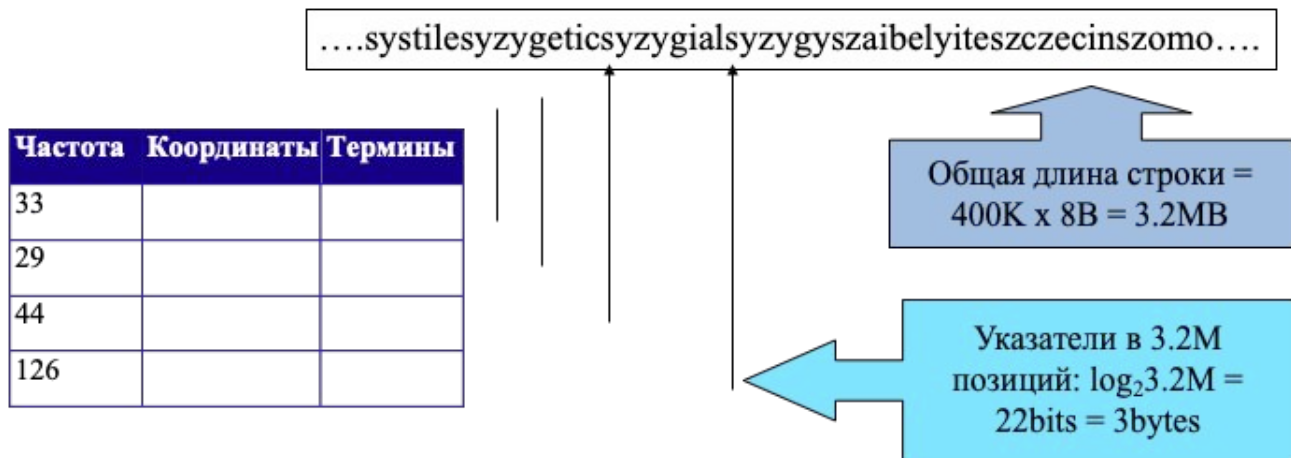


- Большая часть байтов в столбце терминов ни на что не используется – 20 байтов отводится на хранение однобуквенных терминов.
  - И мы всё равно не можем сохранить термины вида *гексакосиойгексеконтагексапараскаведекатриафоб* или *этилоксиэтилпарафенилендиаминсульфат*.
- В письменном английском  $\sim 4.5$  символа на слово.
- Средняя длина слова в английском словаре:  $\sim 8$  символов
- Короткие слова преобладают в тексте, но не в словаре.

# Сжатие списка терминов: представляем словарь строкой



- Словарь – длинная строка:
- Указатель на следующее слово указывает на конец текущего.
- Надеемся на 60%-ное уменьшение размера словаря.



# Место для такого словаря



- 4 байта на частоту.
- 4 байта на указатель на Kb.
- 3 байта на указатель на термин
- 8 байтов на термин в строке
- 400K терминов  $\times 19 \Rightarrow 7.6$  MB (против 11.2 Mb для фиксированной длины)

# Объединяем в блоки



- Храним указатели на каждый k-ый термин
- В примере: k=4.
- Нужно хранить длины терминов (+1 байт)

....7systile9syzygetic8syzygial6syzygy11szaibelyite8szczecin9szomo....

Частота	Коорд.	Термин
33		
29		
44		
126		
7		

} Освободили  
} 9 байтов на 3  
} указателях

← Теряем 4 байта  
на длинах терминов.

# В итоге



- Для размера блока  $k = 4$
- Раньше тратили 3 байта на указатель
- $3 \times 4 = 12$  байтов,
- теперь тратим  $3 + 4 = 7$  байтов.

Убрали  $\sim 0.5$  Mb и уменьшили словарь с 7.6 Mb до 7.1 Mb.

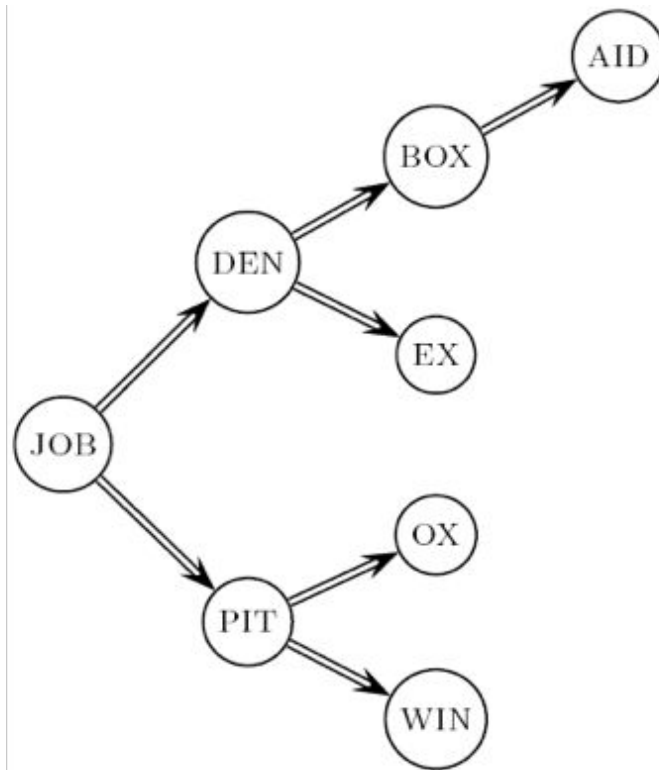
Можем попробовать взять большие  $k$ .

Почему нет?

# Поиск по дереву без блоков



- Если количество доступ к каждому термину равновероятен (на практике это не так), то среднее количество сравнений будет  $= (1+2 \cdot 2+4 \cdot 3+4)/8$   
 $\sim 2.6$
- Если поиск терминов не равновероятен, то как могла бы выглядеть структура данных, учитывающая эту информацию?



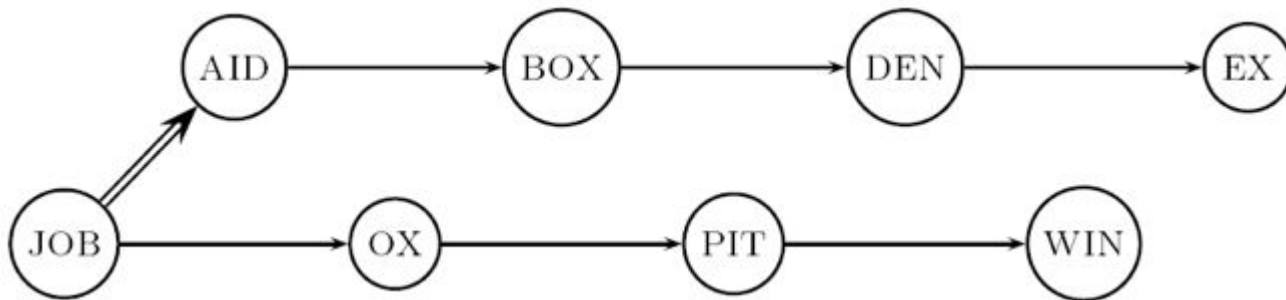


# Поиск по дереву с блоками



- Поиск в дереве доходит до блока с 4-я терминами;
- Далее – линейный поиск внутри блока.
- Теперь среднее =

$$(1+2 \cdot 2+2 \cdot 3+2 \cdot 4+5)/8 = 3 \text{ сравнения}$$

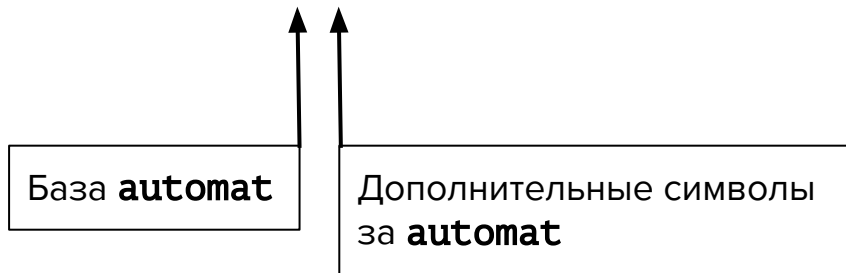


# Префиксное сжатие



- Префиксное сжатие:
- Рядом стоящие слова в отсортированной последовательности обычно имеют длинный общий префикс - будем хранить его отдельно
- 8automata8automate9automatic10automation

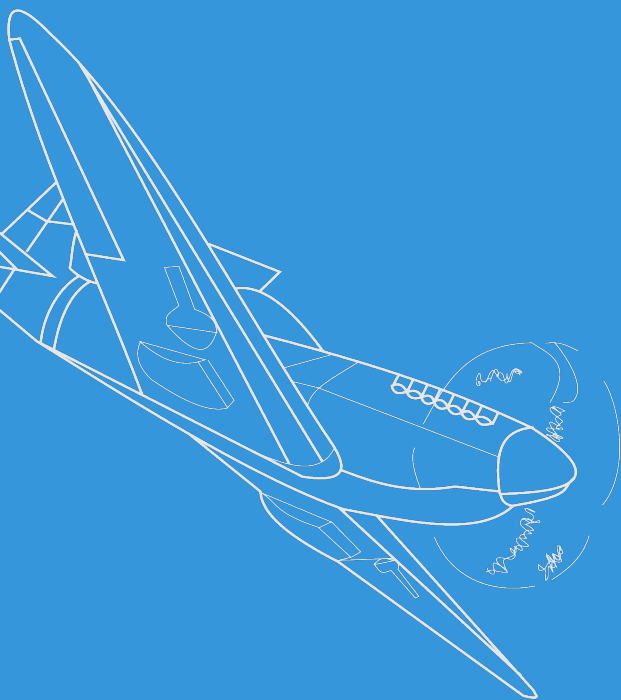
→ 8automat\*a1e2ic3ion



# Сжатие словаря для RCV1



Метод	Размер (МБ)
Фиксированная ширина	11.2
Словарь как строка	7.6
+ блоки $k = 4$	7.1
+ префиксное сжатие	5.9



# Сжатие координатных блоков

# Сжатие координатных блоков



- Координатные блоки значительно больше словаря (для булевого поиска – в 10 раз)
- Будем сжимать каждый постинг.
- В булевском индексе – это docID.
- Для Reuters (800,000 документов), мы можем использовать 32 бита на docID – 4-х байтовые целые.
- Или мы можем взять  $\log_2 800,000 \approx 20$  битов на docID.
- Наша задача: значительно меньше, чем 20 битов.

# Два противоположных случая



- Термин параскаведекатриафобия встречается в одном документе на миллион – потребуется  $\log_2 1M \sim 20$  бит и это нам подходит.
- Термин же и встречается практически везде, 20 битов слишком много.
  - Лучше битмап.
  - А может быть и RLE.



- Список документов хранится в порядке возрастания DocID.
  - computer: 33,47,154,159,202 ...
- **Следовательно:** можем хранить промежутки.
  - 33,14,107,5,43 ...
- **Надеемся:** промежутки потребуют меньше, чем 20 бит.

# Примеры координатных блоков



	encoding	postings list					
THE	docIDs	...	283042	283043	283044	283045	...
	gaps		1	1	1		...
COMPUTER	docIDs	...	283047	283154	283159	283202	...
	gaps		107	5	43		...
ARACHNOCENTRIC	docIDs	252000	500100				
	gaps	252000	248100				



# Кодирование с переменной длиной



- Цель:
  - Для **параскаведекатриафобия**, использовать  $\sim 20$  бит на промежуток.
  - Для **и**, использовать  $\sim 1$  бит на промежуток.
- Если средний промежуток размера  $G$ , мы хотим использовать  $\sim \log_2 G$  битов на промежуток.
- **Главное:** кодировать каждое целое число минимальным количеством битов.
- Требуется код с переменной длиной.
- Будем достигать желаемого тем, что будем назначать короткие коды небольшим промежуткам.

# Код Variable Byte (VB)



- Для  $G$ , мы хотим тратить как можно меньше **байт**, достаточных для хранения  $\log_2 G$  бит
- Выделим в каждом байте один бит под служебные цели,  $s$  (есть продолжение или нет)
- Если  $G \leq 127$ , кодируем его в 7 бит и  $s = 1$
- Иначе кодируем младшие 7 бит и используем дополнительные байты для хранения остатка по тому же алгоритму
- У последнего байта  $s = 1$ , всех остальных –  $s = 0$ .

# Пример



docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Координаты хранятся конкатенированной строкой

000001101011100010000101000011010000110010110001

Ключевое свойство: VB-код является префиксным

Для маленького промежутка (5), VB тратит целый байт.

# Другие коды переменной длины



- Вместо байта можно использовать другие границы выравнивания: 32, 16, 4 бита.
- Меньшая граница выравнивания позволяет не терять биты на мелких промежутках – полезно, если много мелких промежутков.
- Коды переменной длины:
- Часто используется
- Хорошо ложатся на архитектуру ЭВМ.
- Можно так же упаковывать несколько промежутков в одно слово.

- 37



- Можем сжимать лучше на уровне битов
  - Гамма-код – самый известный среди них.
- Представим промежуток  $G$  в виде пары длина и смещение
- смещение это  $G$  в бинарном коде, без ведущей единицы
  - Например,  $13 \rightarrow 1101 \rightarrow 101$
- длина – количество битов в смещении
  - Для 13 (смещение 101), это 3.
- Кодируем длину унарным кодом: 1110.
- Гамма код числа 13 – конкатенация длины и смещения: 1110101

# Примеры гамма-кодов



число	длина	смещение	γ-код
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	11111110	1111111	11111110,1111111
1025	1111111110	0000000001	1111111110,0000000001



- $G$  кодируется  $2 \log G + 1$  битами
- Размер смещения:  $\log G$  битов
- Размер длины:  $\log G + 1$  битов
- Все гамма коды имеют нечётное количество битов
- В два раза хуже лучшего результата,  $\log_2 G$
- Гамма коды – префиксные коды, как и VB
- Могут использоваться для любого распределения чисел.
- Не требует параметров.



# Редко используется на практике



- Нужно учитывать границы машинных слов – 8, 16, 32, 64 бит
  - Операции, затрагивающие границы машинных слов, значительно медленнее.
- Работа с битами может быть медленной.
- VB кодировка выровнена по границам машинных слов и потенциально более быстрая.
- VB значительно проще в реализации.

# Сжатие RCV1



Структура данных	Размер в МБ
словарь, фиксированная ширина	11.2
словарь, термины в одной строке	7.6
блоки, $k = 4$	7.1
блоки и префиксное сжатие	5.9
корпус (текст, разметка и т.п.)	3,600.0
корпус (текст)	960.0
матрица термины-документы	40,000.0
координатные блоки, несжатые (32 битные слова)	400.0
КБ, несжатые (20 битов)	250.0
КБ, VB-код	116.0
КБ, g-код	101.0

# Подводим итоги



- Теперь мы можем создать небольшой индекс для булевского поиска
- Всего 4% от общего размера корпуса
- 10-15% от размера текста корпуса
- Но мы не хранили координатную информацию
- Т.е. в реальности индексы больше размером
  - Но методы сжатия похожи на рассмотренные.
- Рассмотрим ещё несколько алгоритмов



- Рассмотрим среднее кодируемых чисел,  $g$
- Округлим  $g$  до ближайшей степени 2
- Каждое число  $x$  будем представлять как
  - $(x-1)/b$  в унарном коде
  - $(x-1) \bmod b$  в бинарном коде

# Пример:



- DocID: 34, 178, 291, 453
- Промежутки: 34, 144, 113, 162
- Среднее:  $g = (34+144+113+162)/4 = 113,33$
- Округляем:  $b = 64$  (6 бит)
- Теперь:
  - $33 = 0 \cdot 64 + 33 = 0 \quad 100001$
  - $143 = 2 \cdot 64 + 15 = 110 \quad 001111$
  - $112 = 1 \cdot 64 + 48 = 10 \quad 110000$
  - $161 = 2 \cdot 64 + 33 = 110 \quad 100001$



- Похож на Rice
- $b$  подбирается по распределению
- Часто подходит  $b = 0,69 * g$  (не степень 2-ки!)
- Мотивация – в Managing Gigabytes
- Кодирование аналогично Rice
- Но нужно использовать разное количество бит в представлении  $(x-1) \bmod b$
- Хорошо подходит для случайных промежутков

# Пример:



- DocID: 34, 178, 291, 453
- Промежутки: 34, 144, 113, 162
- Среднее:  $g = (34+144+113+162)/4 = 113,33$
- Округляем:  $b = 0,69 * g = 78$  (6 или 7 бит)
- $(x-1) \bmod b < 50$ : 6 бит, иначе – 7 бит
  - $(78-50)/2 = 14$ ,  $50+14 = 64$ , т.е. значения между 50 и 64 указывают на дополнительный бит
- Теперь:
  - $33 = 0 * 78 + 33 = 0 \quad 100001$
  - $50 = 0 * 78 + 50 = 0 \quad 1100100$
  - $64 = 0 * 78 + 64 = 0 \quad 1100101$
  - $143 = 1 * 78 + 65 = 10 \quad 1100111$
  - $112 = 1 * 78 + 34 = 10 \quad 100010$
  - $161 = 2 * 78 + 5 = 110 \quad 000101$

# Свойства кодов Rice и Golomb



- Параметризуются  $b$ 
  - Глобально – для всего корпуса
  - Локально – для термина
    - Можно не хранить:  $g = (N - f_t) / (f_t + 1)$
- Однако не могут настроиться на кластеры вхождений в координатных блоках
- Можно рассчитывать  $b$  не для термина, а для блока рядом стоящих документов
- Лучше сжимают, но медленнее VB





- Нужен код, выровненный по словам
  - 32 бита
- Разобьём слово на две части: 4 бита под управление и 28 бит под данные
- Что можно сохранить в 28-и битах?
  - 1 28-и битное число
  - 2 14-и битных числа
  - 3 9-и битных числа (и теряем 1 бит)
  - 4 7-и битных числа
  - 5 5-и битных чисел (и теряем 3 бита)
  - 7 4-х битных чисел
  - 9 3-х битных чисел (и теряем 1 бит)
  - 14 2-х битных чисел
  - 28 1 битных чисел
- И запишем в 4 управляющих бита используемую схему
  - И ещё могут быть исключения, число больше 28 бит

# Свойства Simple9



- Просто упаковать
  - Следующие 28 чисел помещаются в 1 бит?
    - Если нет, то 14 в два бита?
    - И т.д.
- Быстро распаковать (один if на слово)
- Хорошо сжимает
- Есть варианты, например, Simple16

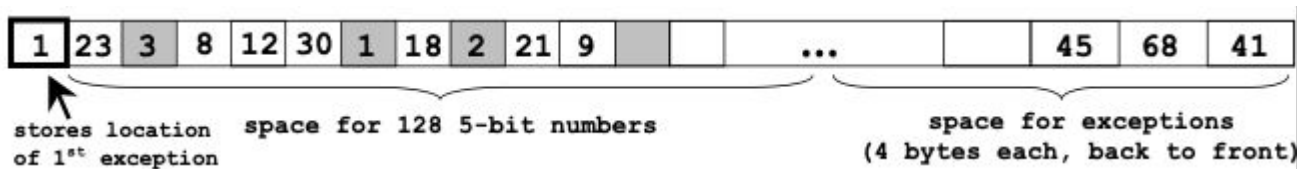


- Будем сжимать сразу много значений, например, 128
- Сколько выделить бит?
  - Выбирать под каждое значение? – Много ветвлений, медленно.
  - Выбрать одно для всех? – Плохо сжимается.
- Идём на компромисс: подберём одно значение для 90% чисел, остальные – исключения

# Пример



- 90% чисел  $< 32$ , возьмём  $b = 5$
- Размер блока =  $5 \cdot 128$  бит + место под исключения
- Исключения храним в хвосте, числами без кода (4 байта на каждое)
- 23, 41, 8, 12, 30, 68, 18, 45, 21, 9, ...





- Могут быть псевдо-исключения
  - Если встретилось подряд больше чем  $2b$  чисел  $< 2b$
- Простая реализация и быстрая декомпрессия
- Всегда распаковываем 128 чисел во временный массив
- Можно немного улучшить за счёт отдельной схемы сжатия под исключения
  - Просто за счёт уменьшения 32 бит до максимальной битности можно получить 10-20% улучшения сжатия

# Что ещё можно почитать



- IIR 5
- MG 3.3, 3.4.
- F. Scholer, H.E. Williams and J. Zobel. 2002. Compression of Inverted Indexes For Fast Query Evaluation. Proc. ACM-SIGIR 2002.
  - Variable byte codes
- V. N. Anh and A. Moffat. 2005. Inverted Index Compression Using Word-Aligned Binary Codes. Information Retrieval 8: 151–166.
  - Word aligned codes

## Рекомендуемая литература

Введение в информационный поиск  
I Маннинг Кристофер Д., Шютце  
Хайнрих



Для саморазвития (опционально)  
Чтобы не набирать двумя  
пальчиками

Спасибо за  
внимание!

**Антон Кухтичев**



[a.kukhtichev@mail.ru](mailto:a.kukhtichev@mail.ru)



[@toshunster](https://t.me/toshunster)