МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

Урок №6

Индексация

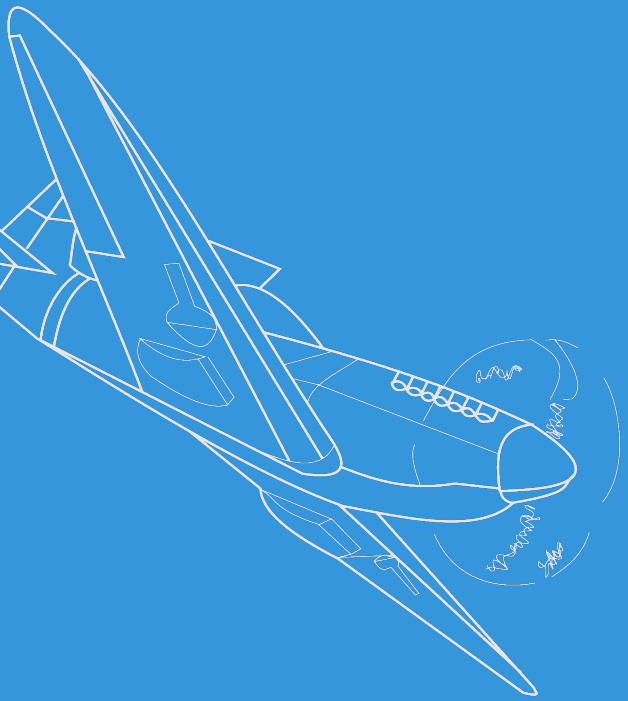
(основано на слайдах Андрея Калинина, Hinrich Schütze, Christina Lioma)

25 марта 2021 года

Антон Кухтичев

Содержание занятия

1. Введение
2. BSBI
3. SPIMI
4. Распределённое индексирование
5. Динамическое индексирование



Введение

Обзор лекции

- Два алгоритма индексирования: BSBI (наивный) и SPIMI (лучше масштабируемый)
- Понятие о распределённой индексации: MapReduce
- Динамическая индексация: как поддерживать индекс в актуальном состоянии при изменении корпуса документов.

Оборудование

- Много архитектурных решений в информационном поиске основаны на ограничениях, накладываемых используемым оборудованием.
- Начнём с обзора общих ограничений, которые нам потребуются в дальнейшем.
- В лекциях про веб-поиск рассмотрим их более подробно.

Оборудование

- Доступ к данным быстрее, если они находятся в памяти, а не на диске (примерно в 10 раз);
- Время поиска дорожки на диске — простаивание: никакие данные не будут передаваться с диска, пока головка не будет правильно установлена;
- Основной принцип оптимизации: чтение одного большого куска данных быстрее, чем большого количества маленьких кусочков.

Оборудование

- Операции ввода-вывода с дисками блочные: приходиться читать блоки целиком, размеры блоков от 8 до 256КБ.
- Используемые сервера: гигабайты или десятки гигабайтов ОЗУ, терабайты или сотни гигбайт дискового пространства.
- Устойчивость к сбоям слишком дорога: дешевле использовать несколько обычных ЭВМ, чем одну, устойчивую к сбоям.

Немного данных (для 2008-го года)

символ	**статистика**	**значение**
s	среднее время поиска	5 ms = 5×10^{-3} s
b	скорость передачи байта	0.02 µs = 2×10^{-8}
	частота процессора	10^{9} s^{-1}
p	время выполнения инструкции (сравнение двух чисел)	0.01 µs = 10^{-8} s
	размер ОЗУ	гигабайты
	размер диска	терабайты

Дисковая подсистема

- Можно поставить несколько дисков:
  - JBOD (just box of disks)
  - RAID (Redundant Array of Independent Disks):
    - RAID 0 (stripe).
    - RAID 1 (mirror)
    - RAID 1+0
    - RAID5
    - RAID6
  - Если RAID, то аппаратный или программный?
- Диски бывают:
  - SCSI, SATA, SAS (Serial Attached SCSI),
  - SSD
  - 5400 RPM, 7200 RPM, 10000 RPM, 15000 RPM.
  - 2”, 3”.

Корпус RCV1

- Пьесы Шекспира недостаточно велики, чтобы продемонстрировать проблемы, возникающие для больших корпусов.
- В качестве примера для использования масштабируемых алгоритмов индексирования будем использовать корпус документов Reuters RCV1.
- Англоязычные новости 1995-го и 1996-го года (целиком один год).

Документ из RCV1

REUTERS

World Business Markets Breakingviews Video More

COMMODITIES NEWS MARCH 24, 2021 / 6:22 PM / UPDATED 2 HOURS AGO

Ships carrying commodities stuck after vessel grounding in Suez Canal

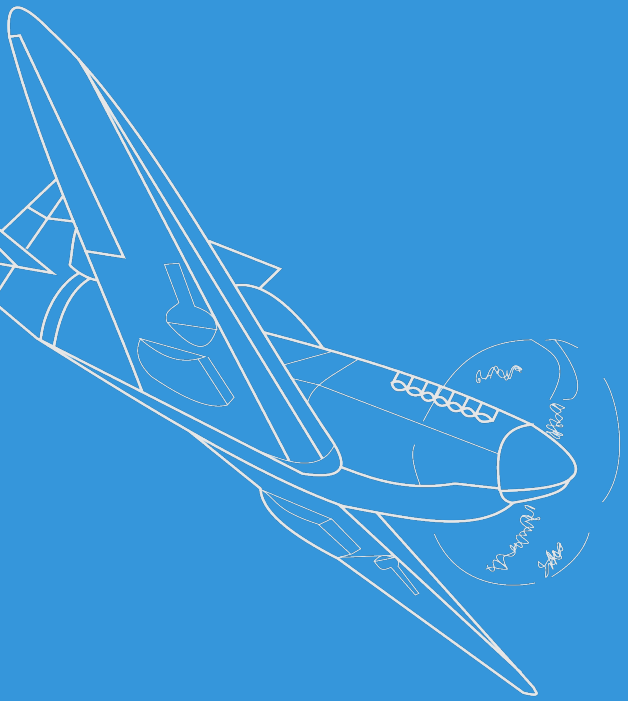
By Jonathan Saul 3 MIN READ

LONDON (Reuters) - Dozens of ships carrying everything from oil to consumer goods have been delayed by the grounding of a vessel in the Suez Canal, and companies may have to re-route cargoes around Africa if the blockage extends beyond 24 hours, shipping sources said.

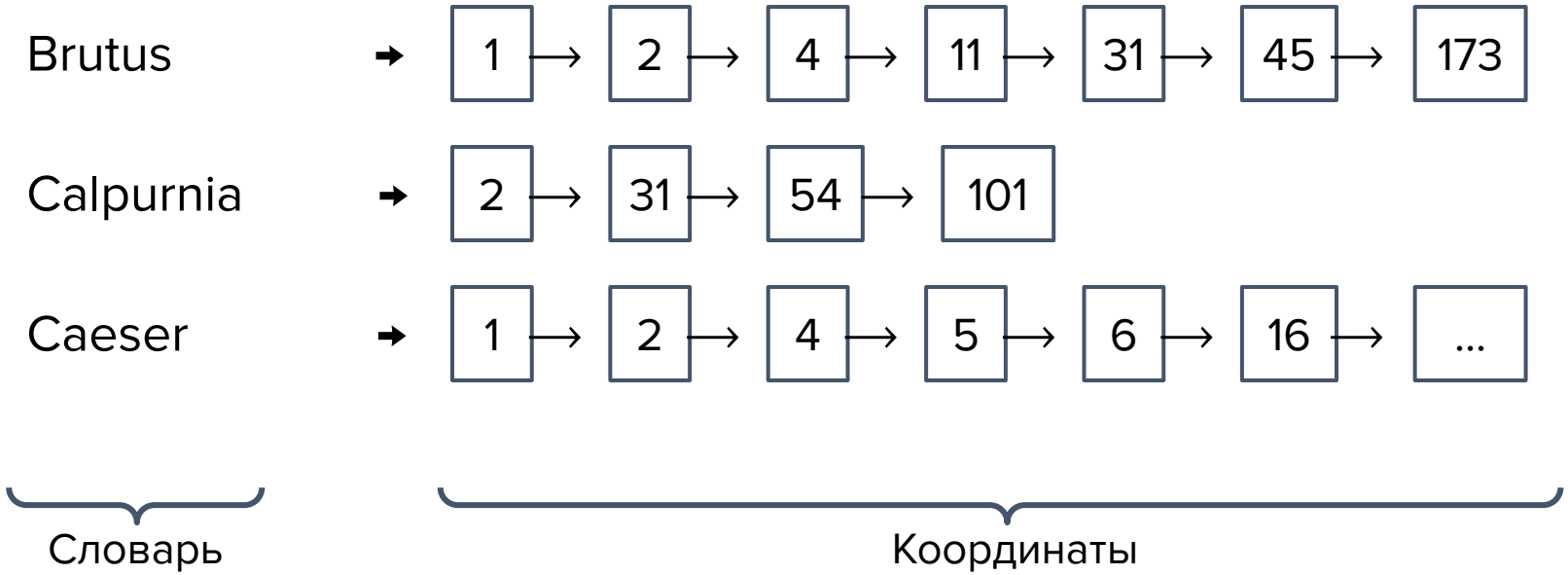
Статистика RCV1

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

BSBI



Задача: построить обратный индекс



В первой лекции: постинги сортировались в памяти

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Индексирование, основанное на сортировке

- Парсим документы по одному
- Постинги для любого термина не завершены до конца работы.
- Можно ли держать все постинги в памяти и отсортировать по завершению?
  - Нет, не для больших корпусов.
- При расходе 10–12 байтов на постинг, потребуется много
- памяти.
- T = 100,000,000 в случае RCV1: мы можем проиндексировать всё в памяти на обычной ЭВМ 2011-го года.
- Но это не масштабируется.
- Следовательно: нужно сохранять промежуточные результаты на диск.

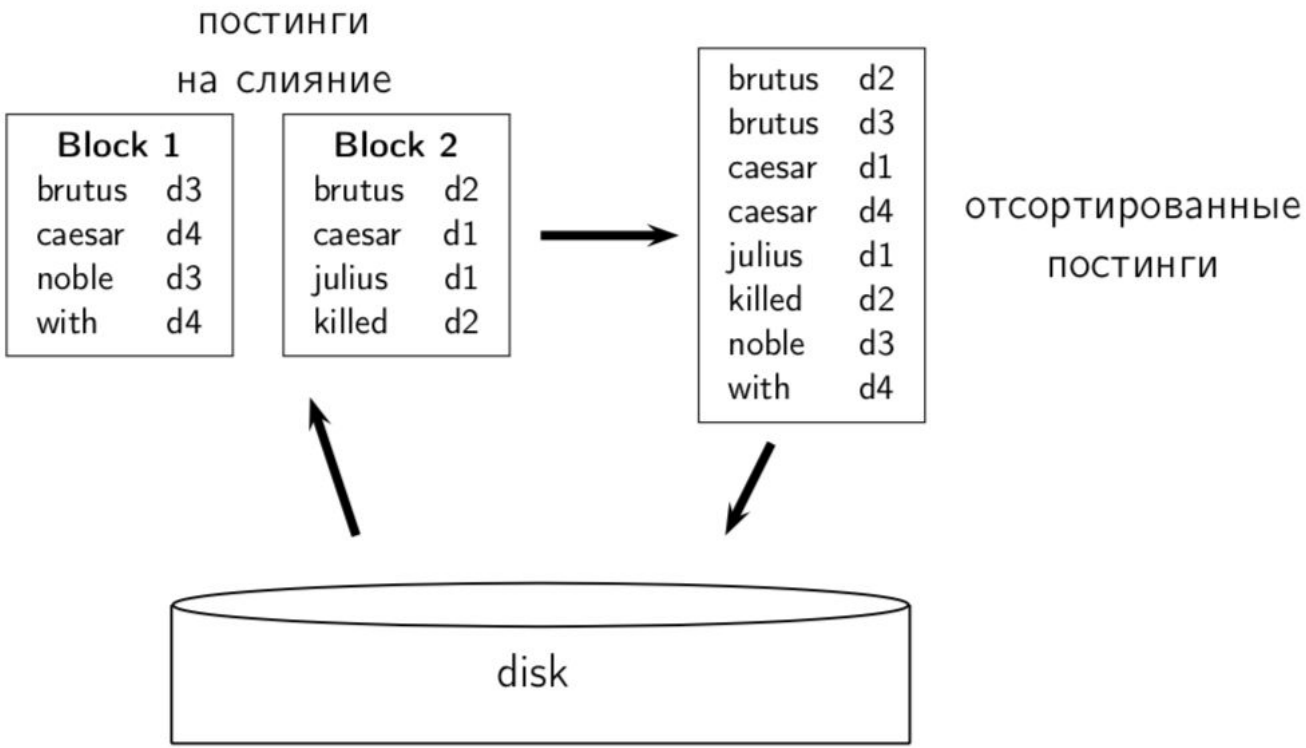
Тот же алгоритм на диске?

- Можно ли использовать тот же алгоритм для больших корпусов, работая с диском вместо памяти?
- Нет: сортировка T = 100,000,000 записей на диске слишком медленно, много перемещений головки.
- Нужен алгоритм внешней сортировки.

«Внешняя» сортировка

- Нужно отсортировать T = 100,000,000 постингов.
- Каждый постинг имеет размер 12 байт (4+4+4: termID, docID, частота).
- Возьмём блок, содержащий 10,000,000 таких постингов.
  - Такой блок можно отсортировать в памяти.
  - RCV1 состоит из 10 блоков.
- Основная идея:
  - На каждый блок: (1) собрать постинги, (2) отсортировать в памяти, (3) записать на диск.
  - Слить блоки в один.

Слияние двух блоков

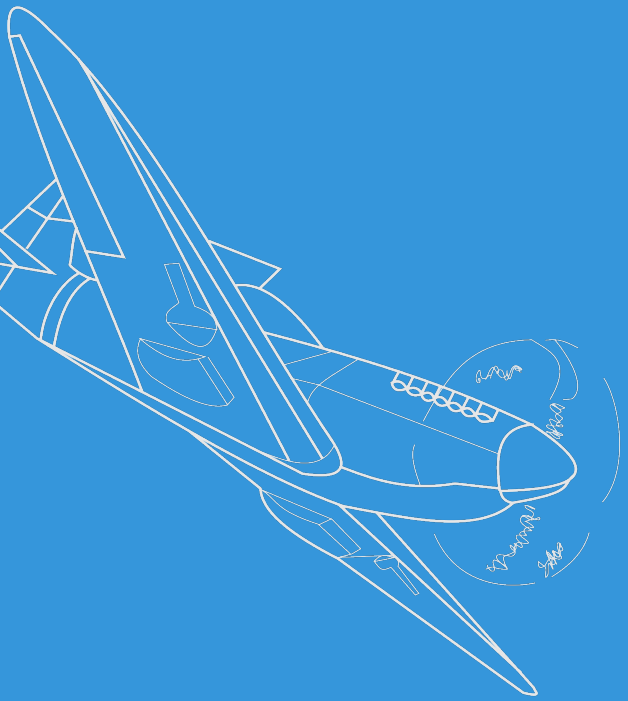


Blocked Sort-Based Indexing


```
BSBIndexConstruction()
1  n ← 0
2  while (не все документы обработаны)
3  do n ← n + 1
4       block ← ParseNextBlock()
5       BSBI-Invert(block)
6       WriteBlockToDisk(block, f_n)
7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Какого размера должен быть блок?

SPIMI



Проблемы с BSBI

- Неявное предположение: словарь находится в памяти.
- Нужен словарь (который постоянно растёт), чтобы отобразить термин в termID.
- В принципе, можно работать в постингами в формате term,docID . . .
- . . . но промежуточные файлы будут очень велики.
- То есть, мы получим масштабируемой, но медленный алгоритм индексирования.

Single-pass in-memory indexing

- Аббревиатура: SPIMI
- Идея 1: создавать отдельные словари дя каждого блока, тогда не требуется поддерживать соответствие term-termID между блоками.
- Идея 2: Не сортировать. Накапливать постинги по мере их появления.
- Тогда мы получим полноценный индекс на каждый блок.
- Эти индексы можно слить в один большой индекс.

SPIMI-Invert

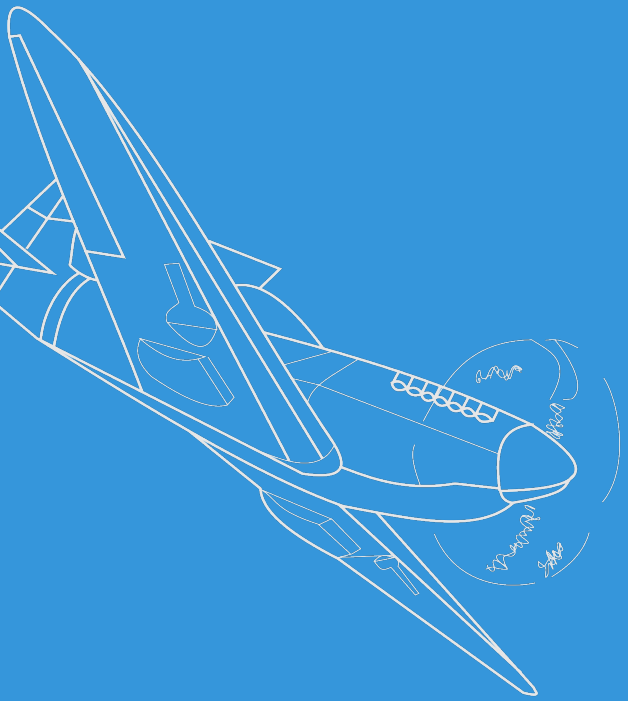
```
SPIMI-INVERT(token_stream)
 1  output_file ← NEWFILE()
 2  dictionary ← NEWHASH()
 3  while (free memory available)
 4  do token ← next(token_stream)
 5     if term(token) ∉ dictionary
 6        then postings_list ← ADDTODICTIONARY(dictionary, term(token))
 7        else postings_list ← GETPOSTINGSLIST(dictionary, term(token))
 8     if full(postings_list)
 9        then postings_list ← DOUBLEPOSTINGSLIST(dictionary, term(token
10     ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

SPIMI: сжатие

- Сжатие делает использование SPIMI ещё более эффективным.
  - Сжатие терминов.
  - Сжатие координат.
  - Будет на следующей лекции.

Упражнение: время на индексацию всего веба?

- Сжатие делает использование SPIMI ещё более эффективным.
  - Сжатие терминов.
  - Сжатие координат.
  - Будет на следующей лекции.



Распределённое индексирование

Один индекс или несколько?

- Можно строить один индекс, можно — несколько.
- Один индекс должен обязательно помещаться на один
- сервер.
- Несколько индексов — единственный способ разбить большой индекс по нескольким серверам.
- Индекс можно разделить на части:
  - По терминам.
  - По документам.
  - Что лучше?
- Несколько индексов можно разместить и на одном сервере, несколько замедлив поиск (несколько маленьких индексов удобнее в эксплуатации, чем один большой).
- Однако, по разделённому индексу сложно собирать статистику, необходимую для ранжирования.

Ролевое индексирование

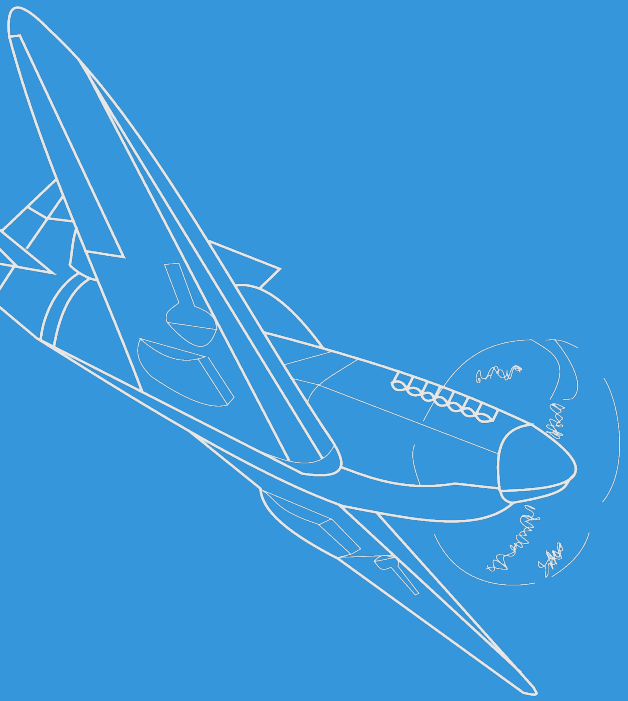
- Делим сервера на роли в рамках алгоритма SPIMI: выкачка, первичная индексация, слияние в один индекс, непосредственный поиск.
- Например: 10 серверов выкачки и первичной индексации, два сервера на слияние индексов, два поисковых фронтенда.
- Работает при небольшом количестве серверов: несколько десятков, сотня уже тяжело.

Распределённое индексирование

- Для задач индексирования больших корпусов (веб-поиск) требуется использование сотен и тысяч серверов.
- При этом каждый сервер ненадёжен.
  - Может непредсказуемо замедлиться или «упасть».
  - То есть, нельзя требовать устойчивости отдельных узлов, но при этом требуется выполнить задачу.
- Как можно использовать много таких серверов для индексации?

Датацентры Google (оценка; Gartner, 2007)

- Основная масса — обычные компьютеры.
- Датацентры распределены по всему миру.
- 1 миллион серверов, 3 миллиона процессоров/ядер.
- Устанавливается 100,000 серверов в каждый квартал.
- Оценка основана на тратах в 200–250 миллионов долларов в год.
- Это 10% вычислительной мощности всего мира.
- Если в системе с 1000 узами каждый узел имеет 99.9% рабочего времени, сколько рабочего времени будет иметь вся система?
- Ответ: 63%
- Предположим, что сервер ломается раз в три года. Для кластера из миллина сервреров каков средний интервал между падениями двух серверов?
- Ответ: меньше двух минут.



Динамическое индексирование

Динамическое индексирование

- До сих пор предполагалось, что корпус — статичный.
- Это редкость: документы добавляются, удаляются и изменяются.
- То есть, словарь и координатные блоки должны динамически изменяться.

Простейший подход

- Держим большой основной индекс на диске
- Новые документы добавлятся в маленький дополнительный индекс в памяти.
- Ищем по обоим, объединяем результаты.
- Периодически вливаем маленький индекс в большой.
- Удаления:
  - Специальный битовый массив для удалённых документов.
  - Фильтруем результаты по этому вектору

Проблемы с дополнительным индексом

- Частые слияния
- Медленный поиск во время слияния.
- Теоретически:
  - Слияние двух индексов не настолько затратно, если выделен отдельный файл для каждого координатного блока.
  - Тогда слияние — простая конкатенация.
  - Но тогда понадобится огромное количество файлов —
  - неэффективно.
- Считаем дальше, что индекс это один большой файл.

Логарифмическое слияние

- Логарифмическое слияние уменьшает стоимость слияния индексов.
  - ➔ слияние меньше сказывается на скорости ответа.
- Используется несколько индексов, каждый вдвое больше
- предыдущего.
- Самый маленький (Z_0) остаётся в памяти.
- Большие индексы (I_0, I_1,...) — хранятся на диске.
- Если Z_0 становится слишком большим (> n), он записывается на диск как I_0
- . . . или сливается с I_0 (если I_0 существовал), а результат добавляется в I1 и т. д.

Проблемы с дополнительным индексом

```
LMergeAddToken(indexes, Z_0, token)
 1  Z_0 ← Merge(Z_0, {token})
 2  if |Z_0| = n
 3    then for i ← 0 to ∞
 4         do if I_i ∈ indexes
 5              then Z_{i+1} ← Merge(I_i, Z_i)
 6                   (Z_{i+1 — временный индекс на диске)
 7                   indexes ← indexes − {I_i}
 8              else I_i ← Z_i     (Z_i становится постоянным индексом I_i.)
 9                   indexes ← indexes ∪ {I_i}
10                   Break
11         Z_0 ← ∅
```

```
LogarithmicMerge()
1  Z_0 ← ∅     (Z_0 — индекс в памяти)
2  indexes ← ∅
3  while true
4  do LMergeAddToken(indexes, Z_0, getNextToken())
```

Проблемы с дополнительным индексом

- Частые слияния
- Медленный поиск во время слияния.
- Теоретически:
  - Слияние двух индексов не настолько затратно, если выделен отдельный файл для каждого координатного блока.
  - Тогда слияние — простая конкатенация.
  - Но тогда понадобится огромное количество файлов —
  - неэффективно.
- Считаем дальше, что индекс это один большой файл.

Рекомендуемая литература

[Введение в информационный поиск | Маннинг Кристофер Д., Шютце Хайнрих]

Для саморазвития (опционально)
[Чтобы не набирать двумя пальчиками]

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@mail.ru

@toshunster