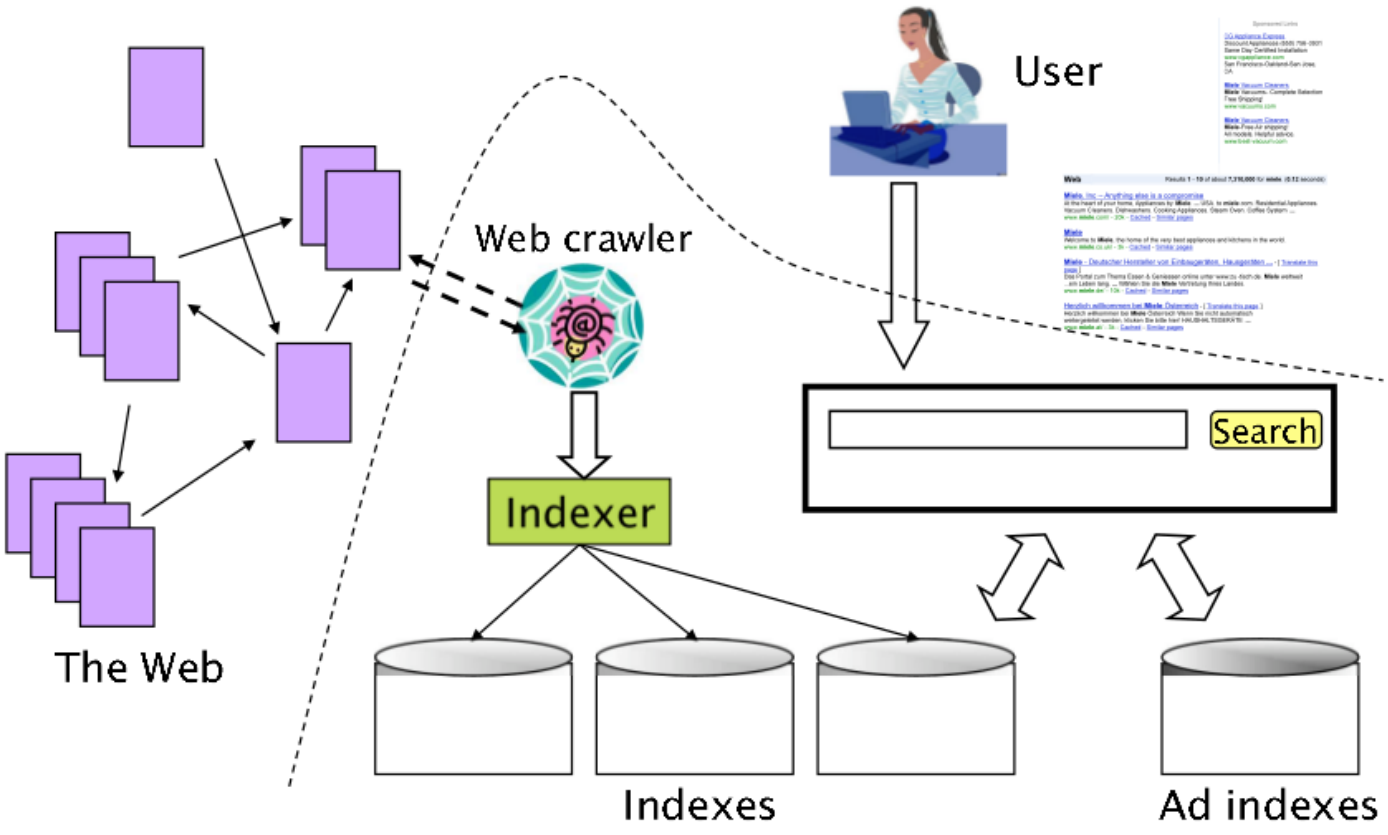
Индексация документов

Лекция 7

БГУ ФПМИ, 2018

План


User
Web crawler
Search
Indexer
The Web
Indexes
Ad indexes

План

Словарь		Словопозиции (postings)
Терм	$\mathbf{N_t}$	
беда	3	3, 10, 11
верста	2	4, 5
друг	7	11, 14, 18, 21, 25, 34, 40
семь	10	1, 3, 4, 11, 15, 23, 37, 45, 51, 56
.....		.....

Поиск в индексе

- Термы запроса ищутся в словаре.
- Списки docId для термов запроса пересекаются аналогично алгоритму merge.
- Для ускорения работы обработку списков нужно производить в порядке возрастания их длин.

Операции

Inverted index как структура данных поддерживает

- first(t)
- last(t)
- next(t, current)
- prev(t, current)

Типы индексов

По детализации словопозиции:

- schema-independent
- schema-dependent:
  1. docId индекс
  2. частотный (frequency) индекс
  3. позиционный (positional) индекс

По построению:

- статический
- динамический

Document ID	Document Content
1	Do you quarrel, sir?
2	Quarrel sir! no, sir!
3	If you do, sir, I am for you: I serve as good a man as you.
4	No better.
5	Well, sir.

Term	Docid list	Positional list	Schema-independent
a	1; 3	1; $(3, 1, \langle 13 \rangle)$	1; 21
am	1; 3	1; $(3, 1, \langle 6 \rangle)$	1; 14
as	1; 3	1; $(3, 2, \langle 11, 15 \rangle)$	2; 19, 23
better	1; 4	1; $(4, 1, \langle 2 \rangle)$	1; 26
do	2; 1, 3	2; $(1, 1, \langle 1 \rangle), (3, 1, \langle 3 \rangle)$	2; 1, 11
for	1; 3	1; $(3, 1, \langle 7 \rangle)$	1; 15
good	1; 3	1; $(3, 1, \langle 12 \rangle)$	1; 20
i	1; 3	1; $(3, 2, \langle 5, 9 \rangle)$	2; 13, 17
if	1; 3	1; $(3, 1, \langle 1 \rangle)$	1; 9
man	1; 3	1; $(3, 1, \langle 14 \rangle)$	1; 22
no	2; 2, 4	2; $(2, 1, \langle 3 \rangle), (4, 1, \langle 1 \rangle)$	2; 7, 25
quarrel	2; 1, 2	2; $(1, 1, \langle 3 \rangle), (2, 1, \langle 1 \rangle)$	2; 3, 5
serve	1; 3	1; $(3, 1, \langle 10 \rangle)$	1; 18
sir	4; 1, 2, 3, 5	4; $(1, 1, \langle 4 \rangle), (2, 2, \langle 2, 4 \rangle), (3, 1, \langle 4 \rangle), (5, 1, \langle 2 \rangle)$	5; 4, 6, 8, 12, 28
well	1; 5	1; $(5, 1, \langle 1 \rangle)$	1; 27
you	2; 1, 3	2; $(1, 1, \langle 2 \rangle), (3, 3, \langle 2, 8, 16 \rangle)$	4; 2, 10, 16, 24

Статический индекс

- Индексируем коллекцию один раз, затем ищем
- Индексация:
  - in-memory
  - sort-based
  - merge-based.

План

Реализация

- Хэш-таблица
- Дерево поиска (B-дерево)
- Сортированный массив

Выбор решения:

- сколько ключей требуется?
- изменяется ли словарь?
- какова частота обращения к различным ключам?

План

Posting

$$(d, f_{t,d}, \left\langle p_0, \ldots, p_{f_{t,d}} \right\rangle)$$

- d — docId
- $f_{t,d}$ — tf
- $\left\langle p_0, \ldots, p_{f_{t,d}} \right\rangle$ — offsets, sorted

Skip lists

Во время поиска задействованы списки словопозиций разных длин:

- Это неэффективно
- Хорошо бы перепрыгивать через некоторое количество документов
- Компрессия делает это проблематичным

Skip lists

Skip list — структура данных для поддержки прыжков по словопозициям, представляет собой список пар (posting,position).
Оптимальное количество skip pointers $\sim \sqrt{P}$.

Пример

- Постинг лист:
  5,11,17,21,26,34,36,37,45,48,51,52,57,80,89,91
- Дельта кодирование:
  5,6,6,4,5,8,2,1,8,3,3,1,5,23,9,2
- Skip list:
  (17,3),(34,6),(45,9),(52,12),(89,15)

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

Static index pruning

top k prune(I, k, ϵ)
  for each term t in I
    retrieve the posting list P_t from I
    if $|P_t| > k$
      for each entry $d \in P_t$
        compute $A(t, d)$ (e.g. by Equation 1)
      let z_t be the kth best entry in row t of A
      $\tau_t \leftarrow \epsilon \cdot z_t$
      for each entry $d \in P_t$
        if $A(t, d) \leq \tau_t$
          remove entry d from P_t
    Save (t, P_t) in the pruned inverted file

$$A(t,d) = \frac{\frac{\log(1+f_{t,d})}{\log(1+avgf_d)} \log \frac{N}{N_t}}{|d|}$$

План

Пример

1. ['Семь раз отмерь, один раз отрежь.', 'Один за всех, все за одного', 'Семь бед — один ответ.', 'Семь вёрст до небес и все лесом.']
2. [['семь', 'раз', 'отмерить', 'один', 'раз', 'отрезать'], ['один', 'за', 'все', 'все', 'за', 'один'], ['семь', 'беда', 'один', 'ответ'], ['семь', 'верста', 'до', 'небеса', 'и', 'все', 'лес']]
3. [('беда', 3), ('верста', 4), ('все', 2), ('все', 2), ('все', 4), ('до', 4), ('за', 2), ('за', 2), ('и', 4), ('лес', 4), ('небеса', 4), ('один', 1), ('один', 2), ('один', 2), ('один', 3), ('ответ', 3), ('отмерить', 1), ('отрезать', 1), ('раз', 1), ('раз', 1), ('семь', 1), ('семь', 3), ('семь', 4)]

Терм	Словопозиции
’беда’	[3]
’верста’	[4]
’все’	[2, 4]
’до’	[4]
’за’	[2]
’и’	[4]
’лес’	[4]
’небеса’	[4]
’один’	[1, 2, 3]
’ответ’	[3]
’отмерить’	[1]
’отрезать’	[1]
’раз’	[1]
’семь’	[1, 3, 4]

buildIndex_sortBased (*inputTokenizer*) ≡

1 *position* ← 0
2 **while** *inputTokenizer.hasNext*() **do**
3 $T \leftarrow$ *inputTokenizer.getNext*()
4 obtain dictionary entry for T; create new entry if necessary
5 *termID* ← unique term ID of T
6 write record $R_{position} \equiv (termID, position)$ to disk
7 $position \leftarrow position + 1$
8 $tokenCount \leftarrow position$
9 sort $R_0 \ldots R_{tokenCount-1}$ by first component; break ties by second component
10 perform a sequential scan of $R_0 \ldots R_{tokenCount-1}$, creating the final index
11 **return**

Алгоритм блочного индексирования

1. Сегментируем коллекцию на равные части.
2. Для каждой части строим обратный индекс в памяти одним из способов.
3. Сохраняем inverted index каждой части на диск.
4. Объединяем все промежуточные результаты в окончательный индекс.

```
mergeIndexPartitions (⟨I_0, ..., I_{n-1}⟩) ≡
1   create empty inverted file I_final
2   for k ← 0 to n − 1 do
3       open index partition I_k for sequential processing
4   currentIndex ← 0
5   while currentIndex ≠ nil do
6       currentIndex ← nil
7       for k ← 0 to n − 1 do
8           if I_k still has terms left then
9               if (currentIndex = nil) ∨ (I_k.currentTerm < currentTerm) then
10                  currentIndex ← I_k
11                  currentTerm ← I_k.currentTerm
12      if currentIndex ≠ nil then
13          I_final.addPostings(currentTerm, currentIndex.getPostings(currentTerm))
14          currentIndex.advanceToNextTerm()
15  delete I_0 ... I_{n-1}
16  return
```