第10讲 专题(一)

信息检索
information retrieval
（以搜索引擎为例）

内容

1. 信息检索的概念及基本步骤
2. 匹配
   - Term-document index
   - 倒排索引
3. 排序(ranking)
   - 词频
   - TF-IDF 权重
   - 向量空间模型
   - 文档相似度计算
   - 排序

信息检索(IR)

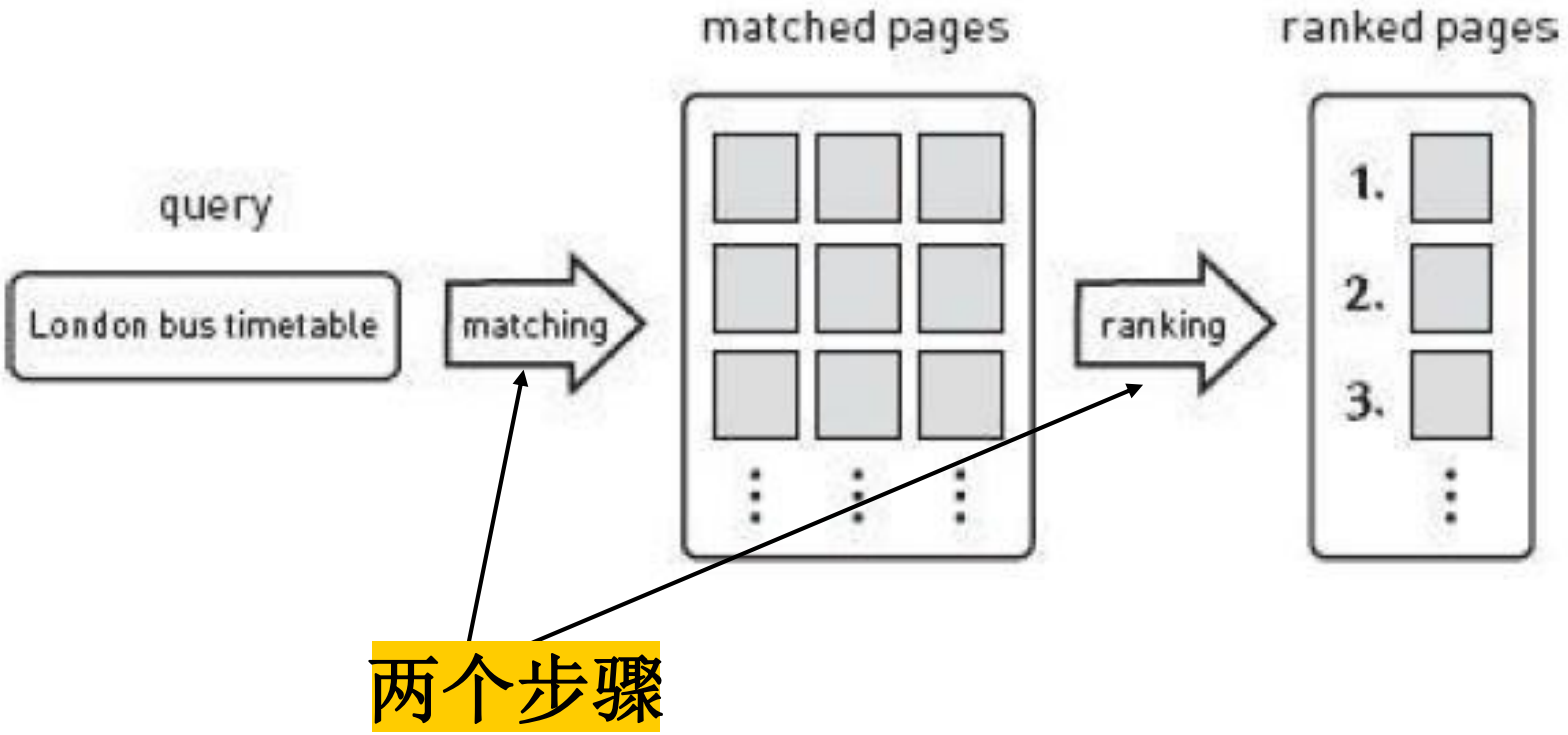
- 信息检索(information retrieval) 用于特定算法或模型从文档中搜索有价值的信息，如搜索引擎，它是NLP的一个重要应用子领域
  - 英文定义：

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

信息检索(IR)

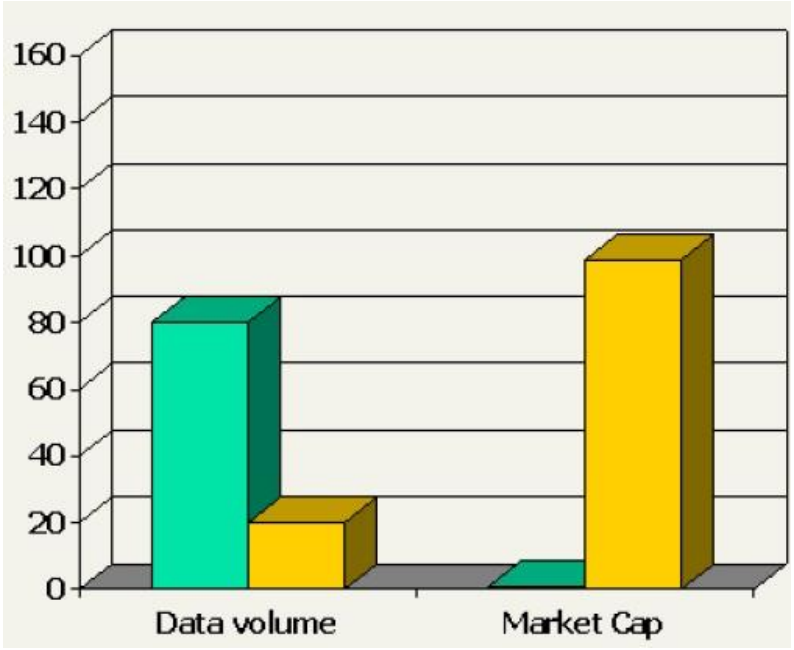
- 通常一个信息检索系统主要包括两块内容：
  - 文档匹配(matching)
    - Finding Needles in the world's Biggest Haystack
  - 文档排序(ranking)
    - Picking out the best few hits in the right order

以搜索引擎为例

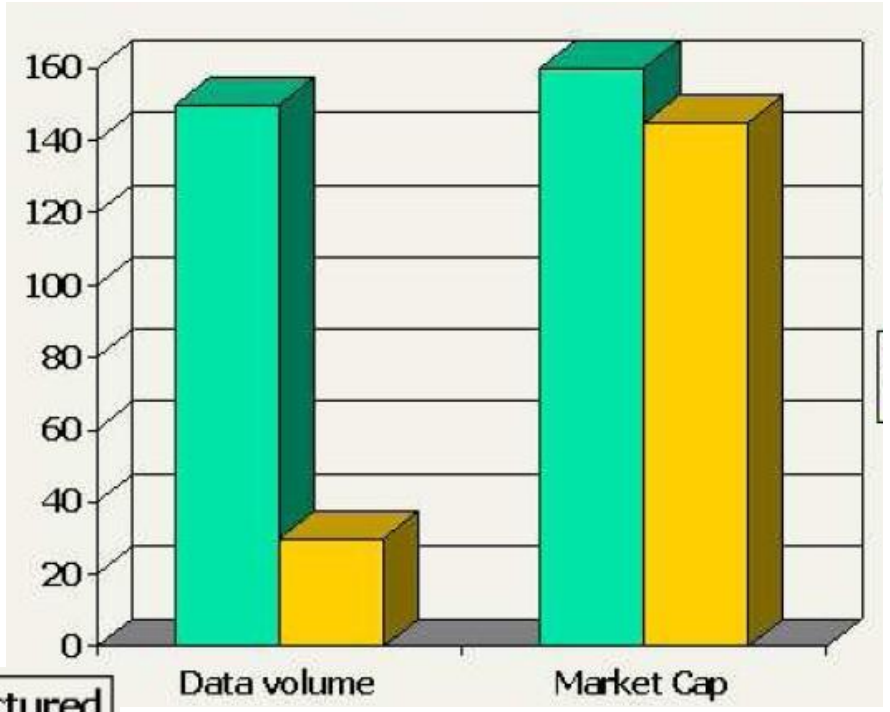

matched pages
ranked pages
query
London bus timetable
matching
ranking
1.
2.
3.
两个步骤

信息检索的数据

- 结构化数据(如数据库中的数据)
- 非结构化数据(如文本数据)



1996年



2006年

匹配
matching

文档匹配

- 文档匹配主要是通过建立索引来实现的
- 索引方法包括：
  - 简单的词-文档索引(term-document indexing)
  - 倒排索引(inverted indexing)

简单词文档(布尔)索引矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

- http://www.rhymezone.com/shakespeare/

Limit to: **All**, Comedies, Tragedies, Histories, Poetry

Keyword search results:

Alas, poor yor<*i*>**i**ck! I knew him, horatio: a fellow → Hamlet: V, i
Yours; for <*i*>**i** will never love that which my friend hates. → Much Ado About Nothing: V, ii
You may th<*i*>**i**nk I love you not: let that appear → Much Ado About Nothing: III, ii
Out, damned spot! out, <*i*>**i** say!--one: two: why, → Macbeth: V, i
When f<*i*>**i**rst I raised the tempest. say, my spirit, → The Tempest: V, i
W<*i*>**i**th love's light wings did I o'er-perch these walls; → Romeo and Juliet: II, ii
<*i*>**i**s this a dagger which I see before me, → Macbeth: II, i
<*i*>**i** love you with so much of my heart that none is → Much Ado About Nothing: IV, i

问题

	文档1	文档2	……	文档n
单词1	1	0	…	1
单词2				
…				

单词是有限的，文档是无限的，导致稀疏矩阵，且效率降低

倒排索引(inverted indexing)

- 倒排索引不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引(inverted index)。带有倒排索引的文件我们称为倒排索引文件，简称倒排文件(inverted file)。

- 倒排索引对象是文档中的单词，用来存储这些单词在一个文档或者一组文档中的存储位置。
- 格式：
  - 单词 文档

简单倒排索引

- 类似于一些书籍中所列出的索引一样，在信息检索及WEB检索中，简单倒排索引方法是类似的
  - 例如：a 3
  - cat 1 3
  - the 1 2 3

例

- 文档1：我 喜欢 电脑
- 文档2：我　说　学习 好
- 文档3：我　讨厌 学习 电脑
- 倒排索引：
  - 我　1 2 3
  - 喜欢　1
  - 电脑　1 3
  - 说　2
  - 好　2
  - 讨厌　3
  - 学习　2 3

此时，若搜索“电脑”，
可以很快检索到相应的文档

此时，若搜索“电脑学习”？

问题

- 文档1：我 喜欢 电脑
- 文档2：我　说　学习 好
- 文档3：我　讨厌 学习 电脑
- 索引：
  - 我　1 2 3
  - 喜欢　1
  - 电脑　1 3
  - 说　2
  - 好　2
  - 讨厌　3
  - 学习　2 3

搜索“电脑学习”这种短语时，
无法知道顺序

增加单词在文档中的位置

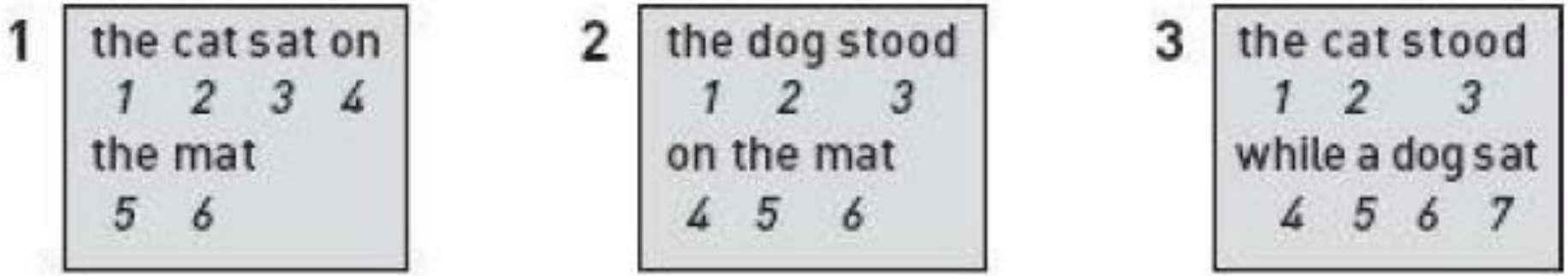
- 词 文档号-在文档中位置
- 例如：喜欢 1-2

例1

- 文档1：我1 喜欢2 电脑3
- 文档2：我1　说2　学习3 好4
- 文档3：我1　讨厌2 学习3 电脑4
- 倒排索引：
  - 我　1-1 2-1 3-1
  - 喜欢　1-2
  - 电脑　1-3 3-4
  - 说　2-2
  - 好　2-4
  - 讨厌　3-2
  - 学习　2-3 3-3

检索“电脑学习”！

例2



1. 已知3个文档，建立完整的带有文档和位置标号的倒排索引；
2. 检索“cat sat”！

例2

1 the cat sat on
1 2 3 4
the mat
5 6

2 the dog stood
1 2 3
on the mat
4 5 6

3 the cat stood
1 2 3
while a dog sat
4 5 6 7

a	3-5				
cat	1-2	3-2			
dog	2-2	3-6			
mat	1-6	2-6			
on	1-4	2-4			
sat	1-3	3-7			
stood	2-3	3-3			
the	1-1	1-5	2-1	2-5	3-1
while	3-4				

检索“cat sat”！

实现

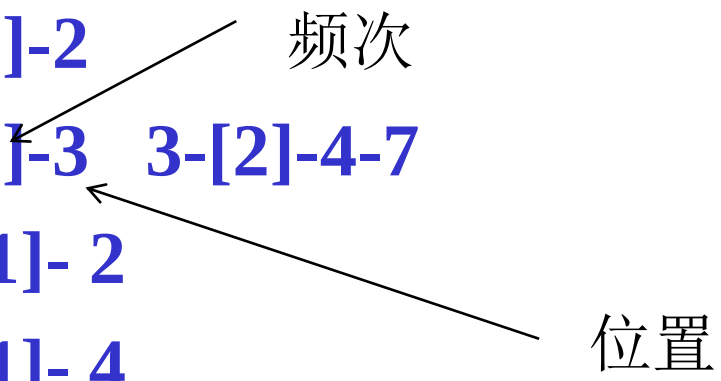
- 假定有3篇文档：
  - 文档1：我喜欢电脑。
  - 文档2：我说学习好。
  - 文档3：我讨厌学习电脑，喜欢玩电脑。
- 步骤1：分词和过滤停用词，标点符号等预处理。
  - 文档1：我 喜欢 电脑
  - 文档2：我 说 学习 好
  - 文档3：我 讨厌 学习 电脑 喜欢 玩 电脑

实现

- **步骤2：建立倒排索引**
  - 最初对应关系是：“文章号”对“文章中所有关键词”
  - 建立倒排索引把这个关系倒过来，变成：“关键词”对“拥有该关键词的所有文章号”。
  - 文档1，2，3经过倒排后变成：
    - 我　1 2 3
    - 喜欢　1 3
    - 电脑　1 3
    - 说　2
    - 好　2
    - 讨厌　3
    - 学习　2 3

实现

- 步骤2可添加位置标记和频次标记
  - 我　　1-[1]-1　2-[1]-1　3-[1]-1
  - 喜欢　1-[1]-2
  - 电脑　1-[1]-3　3-[2]-4-7
  - 说　　2- [1]- 2
  - 好　　2- [1]- 4
  - 讨厌　3- [1]- 2
  - 学习　2- [1]- 3　3- [1]- 3



实现

- 以Java的Lucene全文检索工具包为例，实现过程如下：
  - lucene将上面三列分别以下面三个文件来保存：
    - 词典文件(Term Dictionary)
    - 频率文件(frequencies)
    - 位置文件 (positions)
  - 其中词典文件不仅保存有每个关键词，还保留了指向频率文件和位置文件的指针，通过指针可以找到该关键字的频率信息和位置信息。

英文例 步骤1：文本预处理

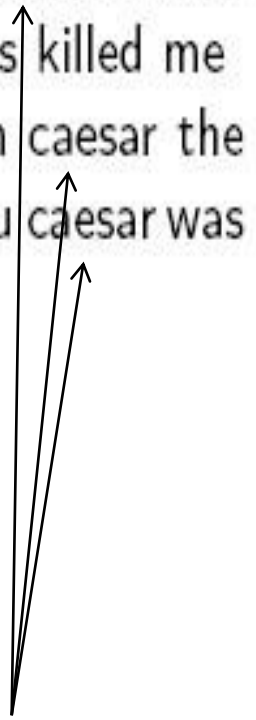
1. Collect the documents to be indexed:
   Friends, Romans, countrymen. | So let it be with Caesar | ...
2. Tokenize the text, turning each document into a list of tokens:
   Friends | Romans | countrymen | So | ...
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend | roman | countryman | so | ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

英文例

Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

⟹

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



以单词caesar为例

步骤1：文本预处理

英文例

步骤2：建立标记

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

以单词caesar为例

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

英文例

步骤3：排序

以单词caesar为例

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

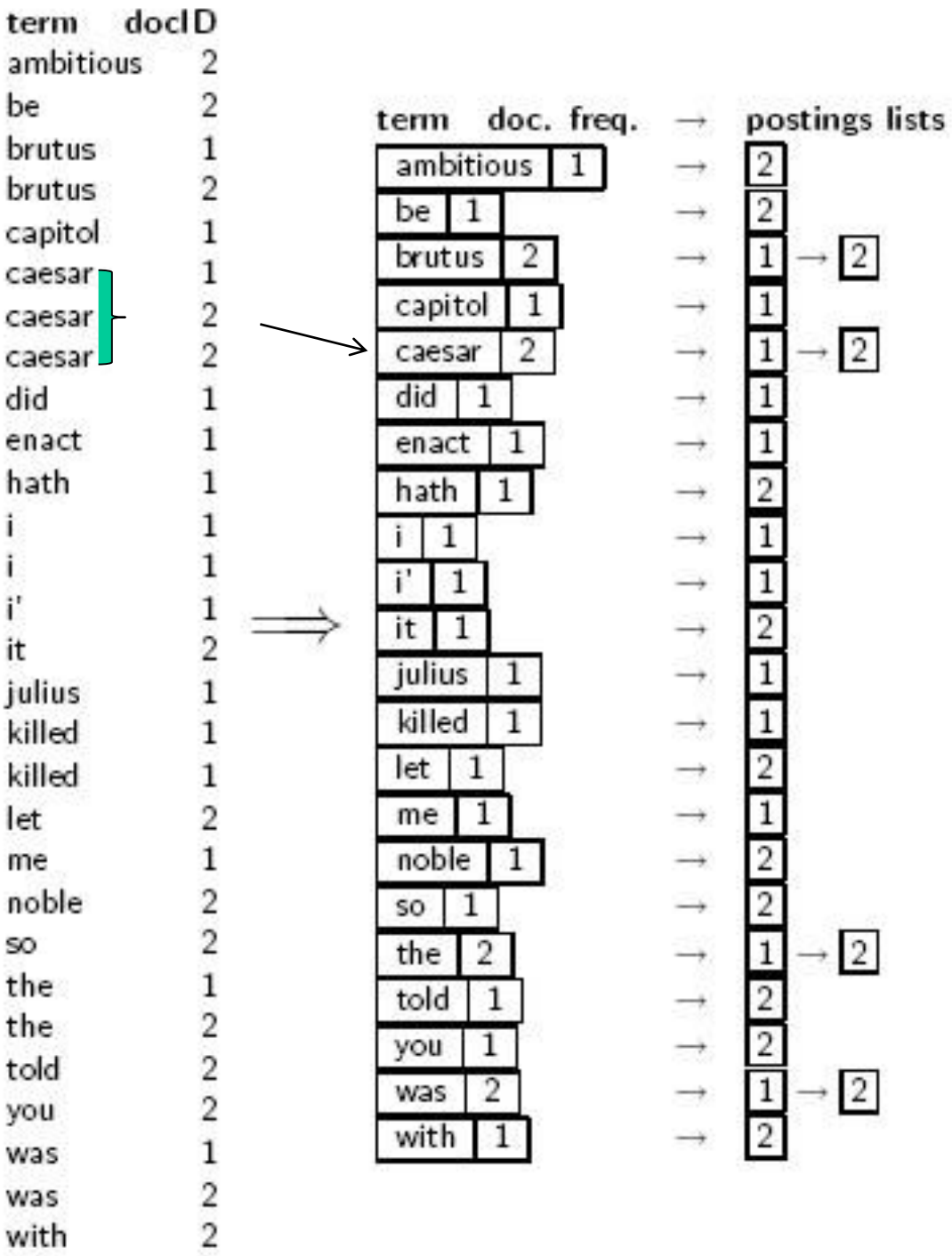
⟹

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

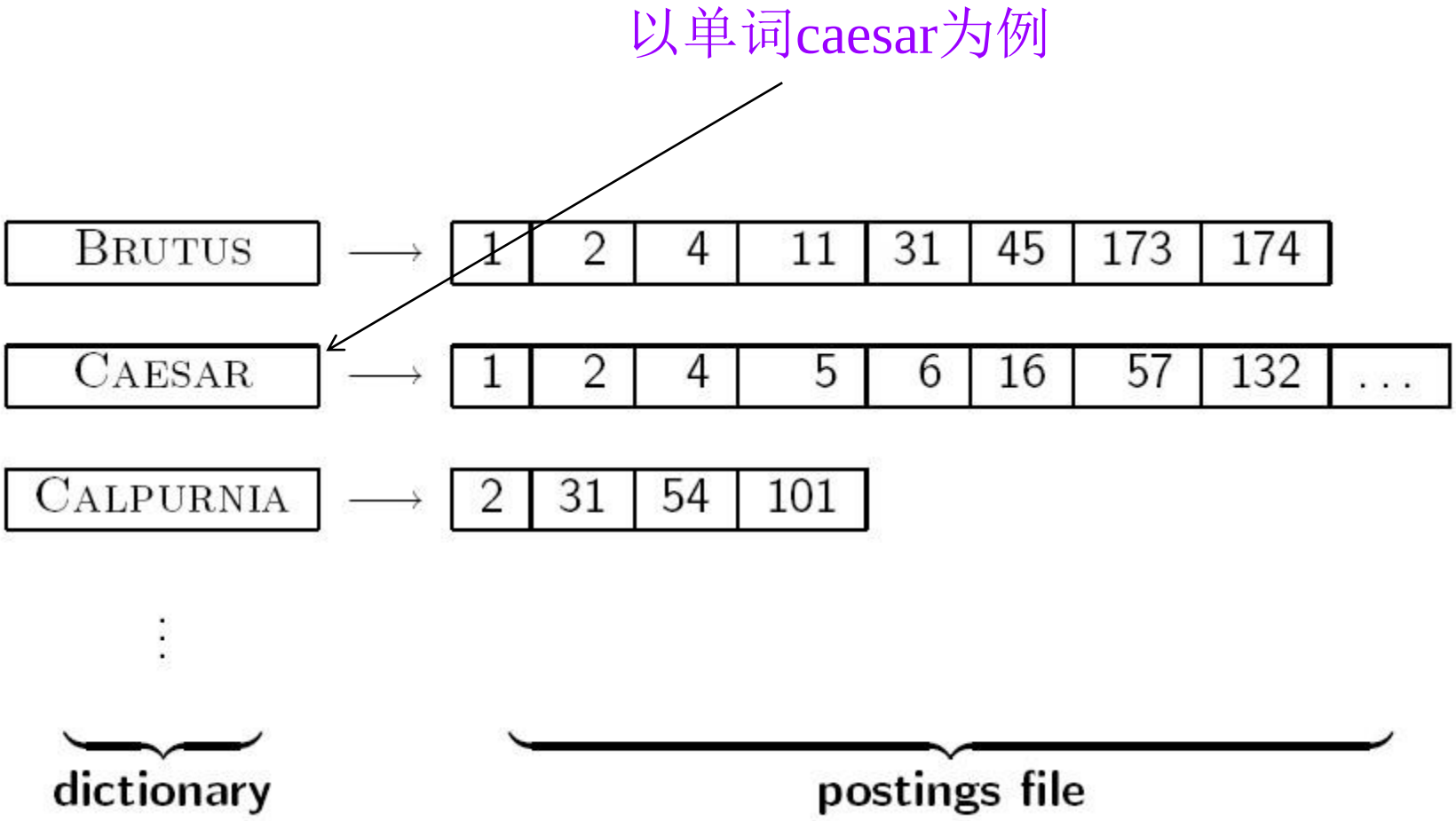
英文例

步骤4：创建标记链表及统计词频

以单词caesar为例



英文例



步骤5：链表分为词典和标记文件

以上为面向文档的检索，接下来我们考虑面向网页的检索

面向网页的检索

- HTML等文件具有特殊的结构，利用这些结构可以检索特殊信息，如检索标题中的信息，检索文件体中的信息，检索超级链接中的信息等。
- 一个HTML：
  - <title>….</title>
  - <body>
  - …..
  - </body>

面向网页的检索

- 将这些标记按照普通文本一样进行倒排索引：

1 <titleStart> my cat <titleEnd> <bodyStart> the cat sat on the mat <bodyEnd>

2 <titleStart> my dog <titleEnd> <bodyStart> the dog stood on the mat <bodyEnd>

3 <titleStart> my pets <titleEnd> <bodyStart> the cat stood while a dog sat <bodyEnd>

练习，请以这3篇文档建立词典和标记文件，
并利用建好的文件检索标题中的dog！
注意：标记文件也需要做作为索引标志！

解析

1 <titleStart> my cat <titleEnd> <bodyStart> the cat sat on the mat <bodyEnd>

2 <titleStart> my dog <titleEnd> <bodyStart> the dog stood on the mat <bodyEnd>

3 <titleStart> my pets <titleEnd> <bodyStart> the cat stood while a dog sat <bodyEnd>

a	3-10
cat	1-3 1-7 3-7
dog	2-3 2-7 3-11
mat	1-11 2-11
my	1-2 2-2 3-2
on	1-9 2-9
pets	3-3
sat	1-8 3-12
stood	2-8 3-8
the	1-6 1-10 2-6 2-10 3-6
while	3-9
<bodyEnd>	1-12 2-12 3-13
<bodyStart>	1-5 2-5 3-5
<titleEnd>	1-4 2-4 3-4
<titleStart>	1-1 2-1 3-1

面向网页的检索

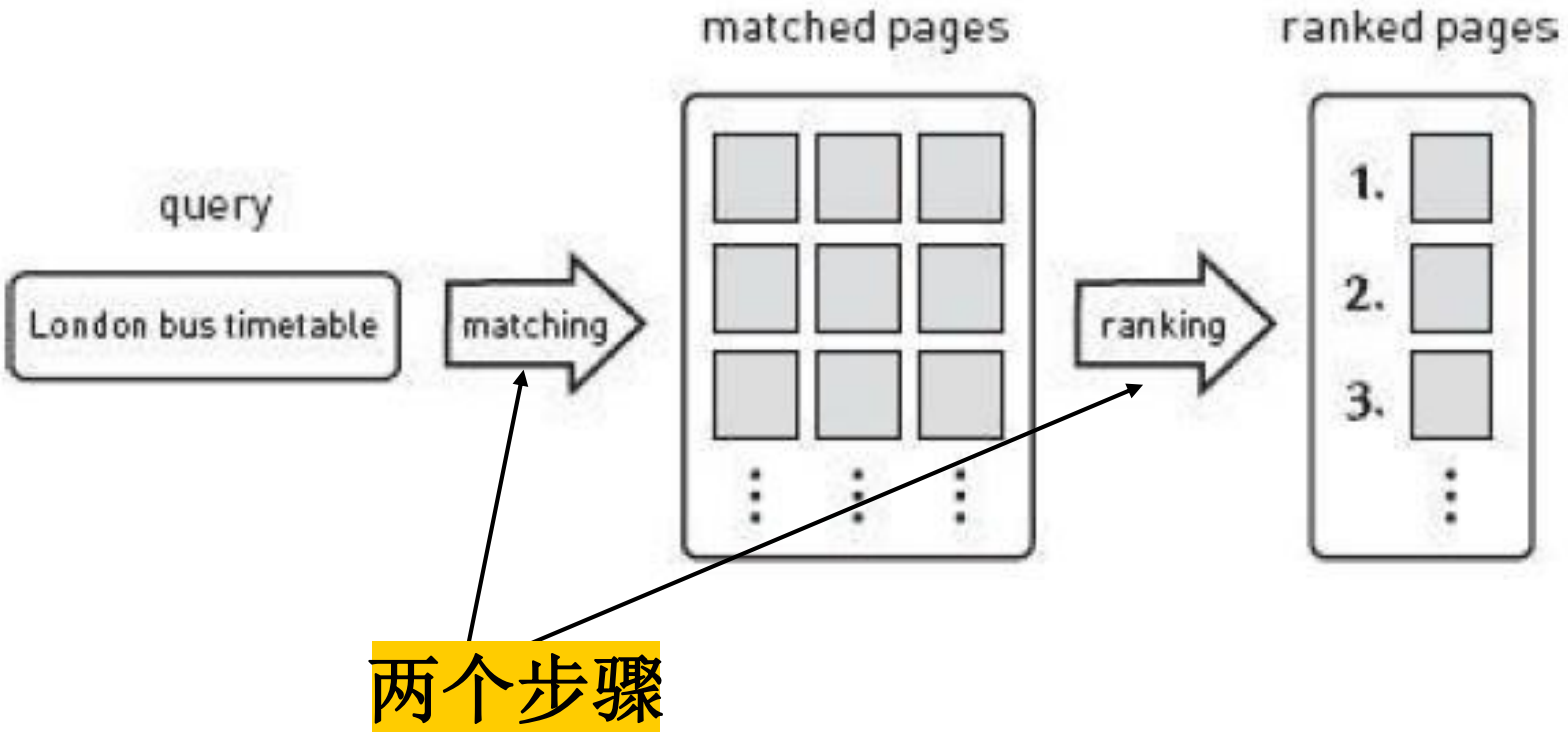
- **JAVA的工具包Lucene使用了field的概念，用于表达信息所在位置（如标题中，文章中，url中），在建索引中，该field信息也记录在词典文件中，每个关键词都有一个field信息(因为每个关键字一定属于一个或多个field)。**

注意

- 可以使用已有的工具包如**java**的**Lucene**来对文档建立倒排索引，从而完成检索的匹配任务；

- 也可以完全自己来写，如：
  - 用哈希表(hashes)或树(binary tree and B-tree)来建立词典，完成排序索引等功能；

以上处理完匹配，接下来处理排序



排序

ranking

为什么要排序

- 有效性
- 时间
- 方法：
  - 对匹配的文档设定一个分值，对分值排序，依据排序的先后显示检索结果

排序用到方法或技术

- **1. TF**
- **2. TF/IDF**
- **3. vector space model**
- **4. PageRank**

- 以单个词查询为例，假定有D1和D2两个文档都匹配了这个查询，那么哪个分值高？
  - 通常来说：这个词在文档中出现次数越高，这个文档分值越高，如果这个词没有出现在这个文档中，则，这个文档分值应为0。
  - 依据这个思路，我们分别有：
    - Jaccard系数打分
    - TF打分
    - TF/IDF打分

Jaccard系数打分

- 对两个集合交集的度量，假定A,B分别为两个集合

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$(A \neq \emptyset \text{ or } B \neq \emptyset)$$

  - **jaccard(*A*,*A*) = 1**
  - **jaccard(*A*,*B*) = 0 if $A \cap B = 0$**
  - **A,B不一定要相同大小，且J系数值在0到1之间**

J系数计算例

- 查询q： *ides of march*
- 文档d1：*caesar died in march*
- 文档d 2: *the long march*
- 则Jaccard(q,d1)= ?
- Jaccard(q,d2)= ?
- 哪个分值更高?

Jaccard系数的问题

- 没有考虑词频信息
  - 通常稀有词比常用词更具有信息价值，如“心情压抑”和“常常”，而Jaccard系数没有考虑这点

词频(TF)

- 添加词频前的词-文档矩阵(布尔矩阵)

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

词频(TF)

- 添加词频后的词-文档矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

例如

- 向量表示没有考虑到文档中单词的顺序问题
- 例如，下面两句话具有相同的向量：
  - **John is quicker than Mary**
  - **Mary is quicker than John**
- 这个被称为词袋模型

词频 TF

- 词频$TF_{t,d}$表示词t出现在文档d中的次数
- 通常来说，词频越高的文档具有更高的分值

- 但，太高就不一定了，也就是说，不成比例！
  - 如：I, am,… … 的，我们，你… …

词频 TF

- 为了避免这一情况，采取对数处理
- 词t在文档d中的对数频率权重为

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 这样，$\mathbf{tf_{t,d}=100, w_{t,d}=3}$… …

词频 TF

- 这样，对某个查询**q**, 文档**d**的分值为：

$$\sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$$

- 这样，当没有查询词出现在文档中时，该文档分值为**0**

练习1

- 假定：
  - 查询q: information on cars
  - 文档d1: all you’ve ever wanted to known about cars
  - 文档d2: information on trucks, information on planes, information on trains

- 计算文档d1和d2的Jaccard计算分值以及TF分值

答案

练习2

- 假定：
  - 查询q: red cars and red trucks
  - 文档d: cops stop red cars more ofter

- 计算文档d的Jaccard计算分值以及TF分值

TF的问题

- 虽然利用取对数能够降低一定的高词频词的影响，但是一般而言，高词频词如：是，喜欢，好。。。比低词频词，如语言学等具有更低的信息量

- 但是若是计算TF，结果恰好相反

文档频率document frequency

- 为此，将高词频词的权重降低，而增加低词频词的权重，我们用到了文档频率

- 文档频率df_t是指包含词t的文档的总数
- 它与TF是相反的一个概念，TF高，同时DF也高，说明这个词是高频词，意义不是很大；若TF高，同时DF低，这说明这个词是低频词，说明这个词的信息量更大。

逆文档频率idf

- 逆文档频率定义为：

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

- 其中：N为所有文档的总数
- idf可以度量词t的忠实度(informativeness)

idf练习

- 假定N=1,000,000，请计算以下词的idf

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

结果说明什么？

idf的问题

- idf对一个词的查询没有影响
- 但对2个词以上的查询会有影响
  - 例：query: arachnocentric line
  - idf会增加第一个词的权重，而降低第2个词的权重

tf-idf 权重

- 一个词t的tf-idf权重值为:

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

- 是信息检索中最为著名的权重方法

- 最终某个查询**q**下文档**d**的排序分值：

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

添加词频后的词-文档矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

- 每个文档以每个词的TF-IDF值构成

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

向量空间模型(VSM)

文档作为向量

- 向量的维数为词的个数；
- 文档为空间中的点或向量

- 特点：
  - 高维
  - 稀疏向量

查询作为向量

- 将查询看成是空间中的一个向量
- 依据该查询向量相似性进行排序
- 相似性：
  - 向量间的相似度
  - 距离的逆

余弦相似度

- **cosine(query,document)**

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- *qi* is the tf-idf weight of term *i* in the query
- *di* is the tf-idf weight of term *i* in the document

长度规范化

向量规范化后 $= x_i / \sqrt{\sum_i x_i^2}$

$$\cos(q,d) = q \cdot d = \sum_{i=1}^{|V|} q_i d_i$$

例

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wutherin g	0	0	38

Term frequencies (counts)

例

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈

$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

开源包

- **http://lucene.apache.org/core/**

作业布置

- 以下两个选一个来完成：
  - 编写一个信息检索系统
    - 参考：http://www.rhymezone.com/shakespeare/

  - 编写一个简单的搜索引擎
    - 参考百度或GOOGLE