



电类工程导论（C类）

张娅

何大治

电子信息与电气工程学院

电子工程系





布尔检索与倒排索引

Boolean Retrieval & Inverted index





信息检索 Information Retrieval

信息检索是从大规模非结构化数据（通常是文本）的集合（通常保存在计算机上）中找出满足用户信息需求的资料（通常是文档）的过程。

- Document – 文档
- Unstructured – 非结构化
- Information need – 信息需求
- Collection—文档集、语料库



结构化 vs 非结构化数据

- 结构化数据即指“表”中的数据

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

数据库常常支持范围或者精确匹配查询。e.g.,
Salary < 60000 AND Manager = Smith.



非结构化数据

通常指自由文本

- “信息检索是指信息按一定的方式组织起来，并根据信息用户的需要找出有关的信息的过程和技术.....”
- “苹果公司在加州库比蒂诺的弗林特艺术中心举行2014年秋季新品发布会，正式发布其新一代产品 iPhone 6.....”

非结构化数据的查询允许：

- 关键词加上操作符号的查询
 - +信息检索， +iphone etc...
- 更复杂的概念性查询，
 - 找出所有的有关药物滥用(drug abuse)的网页

经典的检索模型一般都针对自由文本进行处理



半结构化数据

- 没有数据是完全无结构的
<title>李甲主页</title>
<body>...</body> ...
- 半结构化查询
 - Title contains data AND Bullets contain search
- ... 这里还没有提文本的语言结构



布尔检索

- 布尔查询是指利用 AND, OR 或者 NOT 操作符将词项连接起来的查询
 - 信息 AND 检索
 - 信息 OR 检索
 - 信息 AND 检索 AND NOT 教材



Google Advanced Search

Advanced Search

Find pages with...

To c

all these words:

Type the important words: `tricolor rat terrier`

this exact word or phrase:

Put exact words in quotes: `"rat terrier"`

any of these words:

Type OR between all the words you want:
`miniature OR standard`

none of these words:

Put a minus sign just before words you don't want:
`-rodent, -"Jack Russell"`



一个简单的例子(《莎士比亚全集》)

- ❶ 莎士比亚的哪部剧本包含Brutus及Caesar但是不包含Calpurnia ?
 - 布尔表达式为 Brutus AND Caesar AND NOT Calpurnia
- ❷ 笨方法：从头到尾扫描所有剧本，对每部剧本判断它是否包含Brutus AND Caesar，同时又不包含Calpurnia
- ❸ 笨方法为什么不好？
 - 速度超慢 (特别是大型文档集)
 - 处理NOT Calpurnia 并不容易 (一旦包含即可停止判断)
 - 不太容易支持其他操作 (e.g., find the word Romans near countrymen)
 - 不支持检索结果的排序 (即只返回较好的结果)



词项-文档 (term-doc) 的关联矩阵

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

***Brutus AND Caesar
BUT NOT Calpurnia***

若某剧本包含某单词，则
该位置上为1，否则为0



关联向量 (incidence vectors)

- 关联矩阵的每一列都是 0/1 向量，每个 0/1 都对应一个词项
- 给定查询 Brutus AND Caesar AND NOT Calpurnia
- 取出三个行向量，并对 Calpurnia 的行向量求补，最后按位进行与操作
 - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100.$



检索效果的评价

- ④ **正确率(Precision)**
 - 返回结果文档中正确的比例
 - 如返回80篇文档，其中20篇相关，正确率 $1/4$
- ④ **召回率(Recall)：**
 - 全部相关文档中被返回的比例
 - 如返回80篇文档，其中20篇相关，但是总的应该相关的文档是100篇，召回率 $1/5$
- ④ **正确率和召回率反映检索效果的两个方面，缺一不可。**
 - 全部返回，正确率低，召回率100%
 - 只返回一个非常可靠的结果，正确率100%，召回率低



大文档集的词项-文档矩阵

- 假定
 - $N = 1$ 百万(1M)篇文档, 每篇有1000个词(1K)
 - 每个词平均有6个字节(包括空格和标点符号)
 - →所有文档将约占6GB 空间
- 假定词汇表的大小(即词项个数)为500K
 - →矩阵大小为500K x 1M=500Billion
- 词项-文档矩阵高度稀疏(sparse)
 - 词项-文档关联矩阵中最多有10亿(1Billion)个1

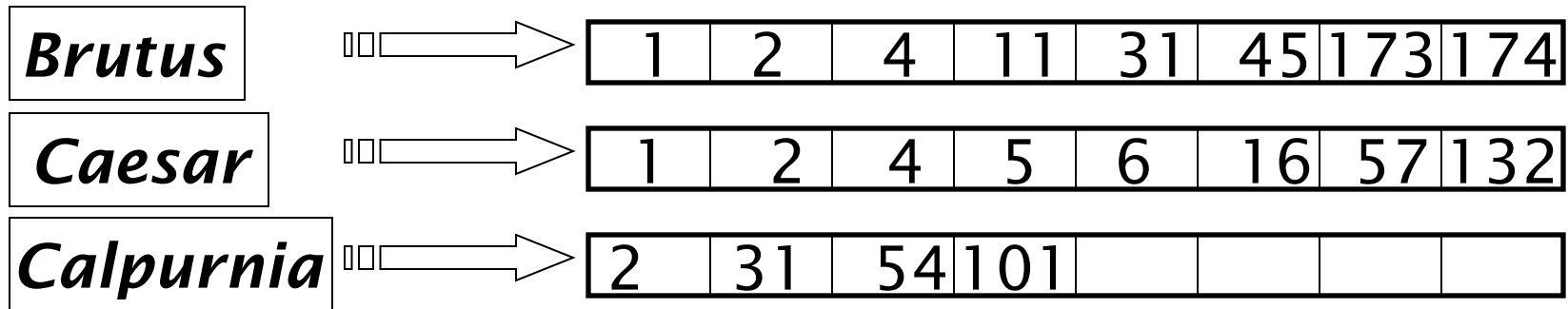
是否有更好的表示方式？

比如：仅仅记录所有1的位置？



倒排索引 (Inverted index)

- 对每个词项t, 记录所有包含t的文档列表.
 - 每篇文档用一个唯一的 docID来表示, 通常是正整数, 如1,2,3...
- 能否采用定长数组的方式来存储docID列表?



文档14中加入单词 **Caesar** 时该如何处理?



倒排索引 (续)

- 通常采用变长表方式
 - 磁盘上，顺序存储方式比较好，便于快速读取
 - 内存中，采用链表或者可变长数组方式
 - 存储空间/易插入之间需要平衡

倒排记录

Posting

Brutus

Caesar

Calpurnia

1 2 4 11 31 45 173 174

1 2 4 5 6 16 57 132

2 31 54 101

Dictionary

词典

Postings 倒排(记录)表

按docID排序 (原因后面再讲)



倒排索引构建

待索引文档



Friends, Romans, countrymen.

⋮

Tokenizer

词条化工具

词条流

*More on
these later.*

Linguistic modules

语言分析工具

Friends

Romans

Countrymen

修改后的词条

friend

roman

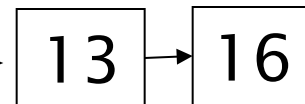
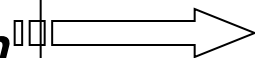
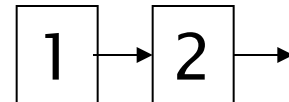
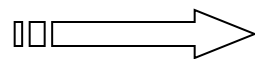
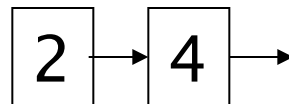
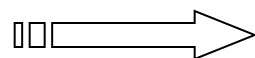
countryman

Indexer

friend

roman

countryman



倒排索引



索引构建过程： 词条序列

• <词条, docID>二元组

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



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



索引构建过程：排序

- 按词项排序
 - 然后每个词项按docID排序

索引构建的核心步骤

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
ambitio	
us	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



索引构建过程：词典 & 倒排记录表

- 某个词项在单篇文档中的多次出现会被合并
- 拆分成词典和倒排记录表两部分
- 每个词项出现的文档数目(doc.

会被加入

为什么加入？后面会讲

Term docID

ambitio
us 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



term doc. freq.

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

→ postings lists

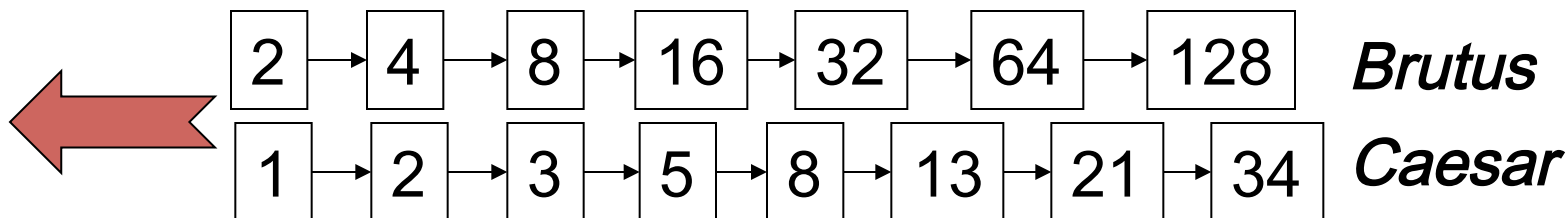
→ 2
→ 2
→ 1 → 2
→ 1
→ 1 → 2
→ 1
→ 1
→ 2
→ 1
→ 1
→ 2
→ 1
→ 2
→ 2
→ 1 → 2
→ 2
→ 2
→ 1 → 2
→ 2



AND查询的处理

考虑如下查询（从简单的布尔表达式入手）：

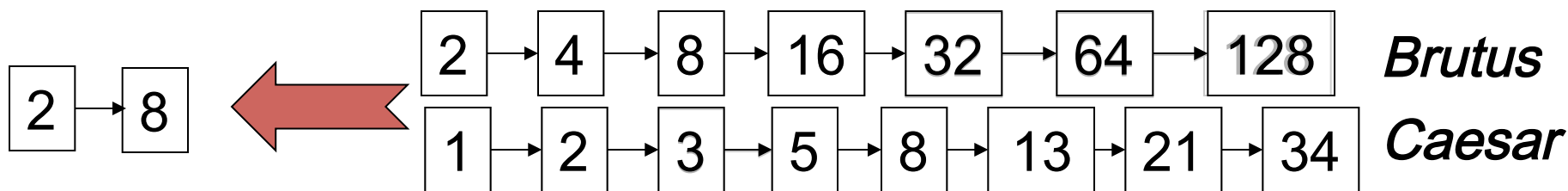
- Brutus AND Caesar
- 在词典中定位 Brutus
 - 返回对应倒排记录表(对应的docID)
- 在词典中定位Caesar
 - 再返回对应倒排记录表
- 合并(Merge)两个倒排记录表，即求交集





合并过程

- 每个倒排记录表都有一个定位指针，两个指针同时从前往后扫描，每次比较当前指针对应倒排记录，然后移动某个或两个指针。合并时间为两个表长之和的线性时间



假定表长分别为 x 和 y , 那么上述合并算法的复杂度为 $O(x+y)$

关键原因: 倒排记录表按照docID排序



上述合并算法的伪代码描述

INTERSECT(p_1, p_2)

```
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8          then  $p_1 \leftarrow \text{next}(p_1)$ 
9          else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return answer
```



其它布尔查询的处理

- OR表达式：Brutus OR Caesar
- 两个倒排记录表的和集
- NOT表达式：Brutus AND NOT Caesar
- 两个倒排记录表的减

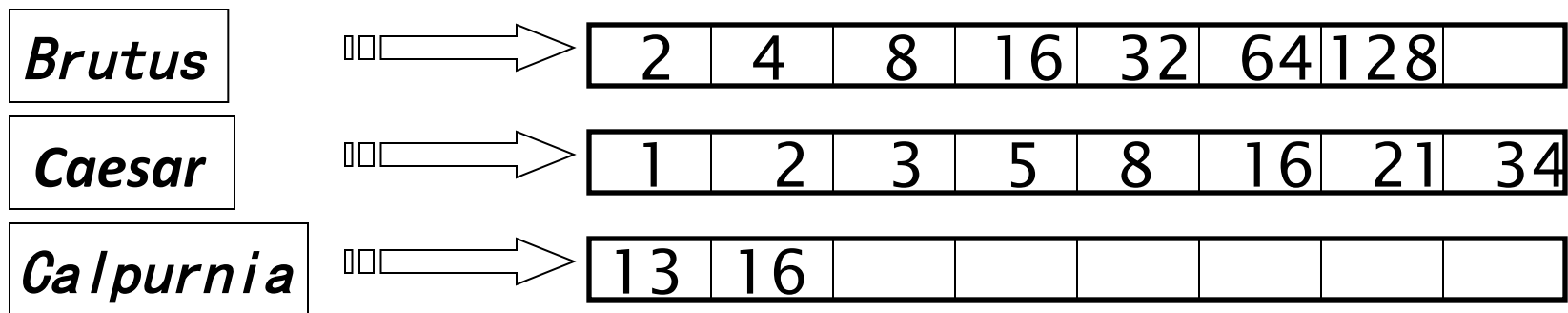
④ 一般的布尔表达式
(Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

④ 查询处理的效率问题！



查询优化

- 查询处理中是否存在处理的顺序问题？
- 考虑n个词项的 AND
- 对每个词项，取出其倒排记录表，然后两两合并



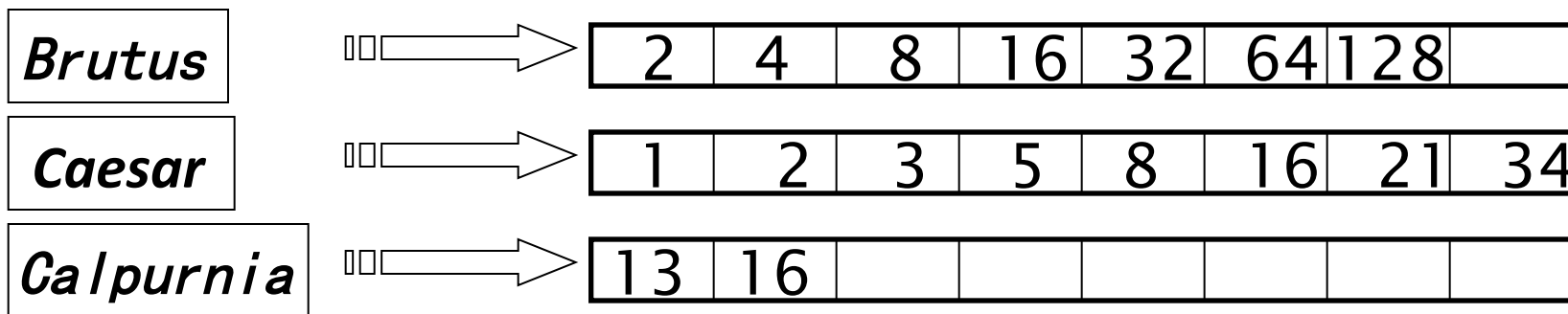
查询: **Brutus AND Calpurnia AND Caesar**



查询优化

- 按照表从小到大(即df从小到大)的顺序进行处理:
 - 每次从最小的开始合并

这是为什么保存
df的原因之一



相当于处理查询 (***Calpurnia AND Brutus***) ***AND Caesar***.



更通用的优化策略

- ⊙ e.g., (madding OR crowd) AND (ignoble OR strife)
 - 每个布尔表达式都能转换成上述形式(合取范式)
- ⊙ 获得每个词项的df
- ⊙ (保守)通过将词项的df相加，估计每个OR表达式对应的倒排记录表的大小
- ⊙ 按照上述估计从小到大依次处理每个OR表达式.



布尔检索的优点

- 构建简单，或许是构建IR系统的一种最简单方式
 - 在30多年中是最主要的检索工具
 - 当前许多搜索系统仍然使用布尔检索模型:
 - 电子邮件、文献编目、Mac OS X Spotlight工具



布尔检索例子: WestLaw

- ⊙ (付费用户数目)最大的商业化法律搜索服务引擎 (1975年开始提供服务; 1992年加入排序功能)
- ⊙ 几十T数据, 700,000用户
- ⊙ 大部分用户仍然使用布尔查询
- ⊙ 查询的例子:
 - 有关对政府侵权行为进行索赔的诉讼时效(What is the statute of limitations in cases involving the federal tort claims act?)
 - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
 - /3 = within 3 words, /S = in same sentence



布尔检索例子: WestLaw

- ⊙ 另一个例子:
 - 残疾人士能够进入工作场所的要求 (Requirements for disabled people to be able to access a workplace)
 - disabl! /p access! /s work-site work-place (employment /3 place)
- ⊙ 扩展的布尔操作符
- ⊙ 很多专业人士喜欢使用布尔搜索
 - 非常清楚想要查什么、能得到什么
- ⊙ 但是这并不意味着布尔搜索其实际效果就很好....



Google支持布尔查询

- 想查关于2011年快女 6进5 比赛的新闻，用布尔表达式怎么构造查询？
- (2011 OR 去年) AND (快乐女声 OR 快女 OR 快乐女生)
AND (6进5 OR 六进五 OR 六 AND 进 AND 五)
- 表达式相当复杂，构造困难！
- 不严格的话结果过多，而且很多不相关；非常严格的话结果会很少，漏掉很多结果。



布尔检索的缺点

- ④ 布尔查询构建复杂，不适合普通用户。构建不当，检索结果过多或者过少
- ④ 没有充分利用词项的频率信息
 - 1 vs. 0 次出现
 - 2 vs. 1次出现
 - 3 vs. 2次出现, ...
 - 通常出现的越多越好，需要利用词项在文档中的词项频率(term frequency, tf)信息
- ④ 不能对检索结果进行排序



参考资料

- Introduction to Information Retrieval
 - [Chap. 1, Boolean retrieval](#)



谢 谢！

