信息检索与Web搜索

第2讲 布尔检索

Boolean Retrieval

授课人：高曙明

*改编自“现代信息检索”网上公开课件（http://ir.ict.ac.cn/~wangbin）

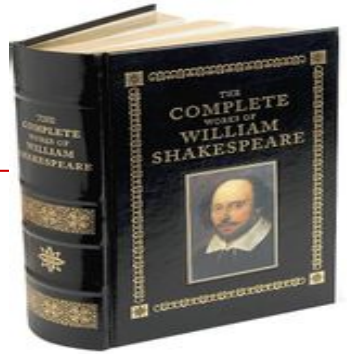
布尔检索

- 针对布尔查询的检索，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支持布尔检索

布尔检索模型

- **检索模型**：查询与文档之间的相关性表示，文档及查询的表示
- **布尔检索模型**
  - 文档采用词项集合表示
  - 查询用词项的布尔表达式表示
  - 相关性：整个查询的相关性通过对词项相关性进行布尔运算得到，对于查询中的每个词项，包含则相关
  - 相关性只有相关和不相关两种

基于扫描的布尔检索



- **信息需求：** 确定莎士比亚的哪些剧本包含Brutus及Caesar但是不包含Calpurnia
- **布尔查询表示：** Brutus AND Caesar AND NOT Calpurnia
- **简单直接的方法：** 从头到尾扫描所有剧本，对每部剧本判断它是否包含Brutus和Caesar，同时又不包含Calpurnia
- **存在的问题**
  - 速度超慢 (特别是对超大型文档集)
  - 处理NOT Calpurnia 并不容易（不到末尾不能停止判断）

词项-文档关联矩阵

- **词项-文档关联矩阵：** 给定一个词项集和一个文档集，由其中的词项和文档之间的关联关系形成的矩阵

	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

若某剧本包含某单词，则该位置上为1，否则为0

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

基于关联向量的布尔检索

- **关联向量：** 指关联矩阵的每一行、每一列对应的向量
- 给定查询**Brutus AND Caesar AND NOT Calpurnia**
- 具体步骤：取出三个相应的行向量，并对**Calpurnia**的行向量求补，最后按位进行与操作

  110100 AND 110111 AND 101111 = 100100

- **检索结果：** **Antony and Cleopatra** 和**Hamlet**

基于关联向量的布尔检索

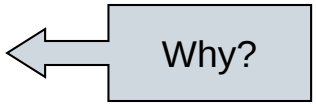
- **Antony and Cleopatra, Act III, Scene ii**
  - Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
  - When Antony found Julius Caesar dead,
  - He cried almost to roaring; and he wept,
  - When at Philippi he found Brutus slain.
- **Hamlet, Act III, Scene ii**
  - Lord Polonius: I did enact Julius Caesar I was killed i' the
  - Capitol; Brutus killed me.

一个更真实的场景

- 假定N = 1 百万篇文档(1M), 每篇有1000个词(1K)
- 假定每个词平均有6个字节(包括空格和标点符号)
  - 那么所有文档将约占6GB 空间.
- 假定词汇表的大小(即词项个数) M = 500K

超大的词项-文档矩阵

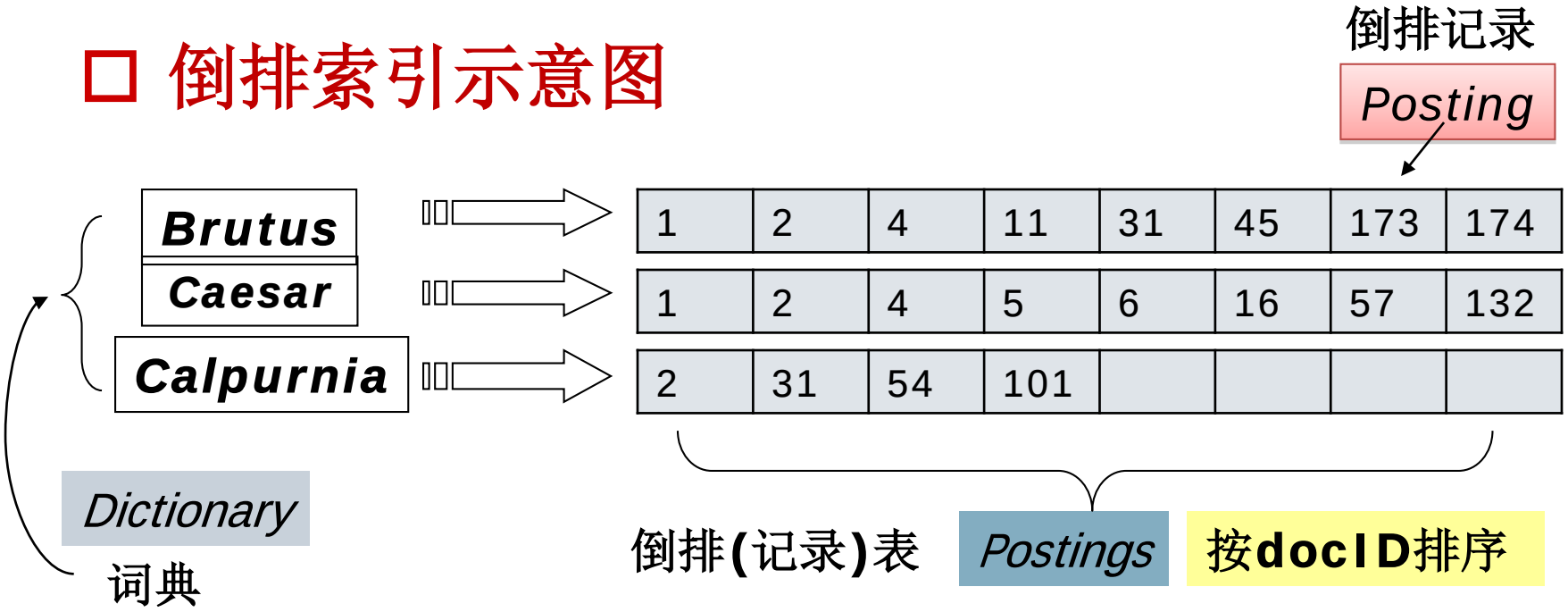
- 矩阵大小为 500K x 1M=500G
- 但是该矩阵中最多有10亿(1G)个1



  - 词项-文档矩阵高度稀疏(sparse).
  - 稀疏矩阵
- 不应简单地建立和存储词项文档关联矩阵
- 应该有更好的表示方式
  - 比如我们仅仅记录所有1的位置

倒排索引(Inverted index)

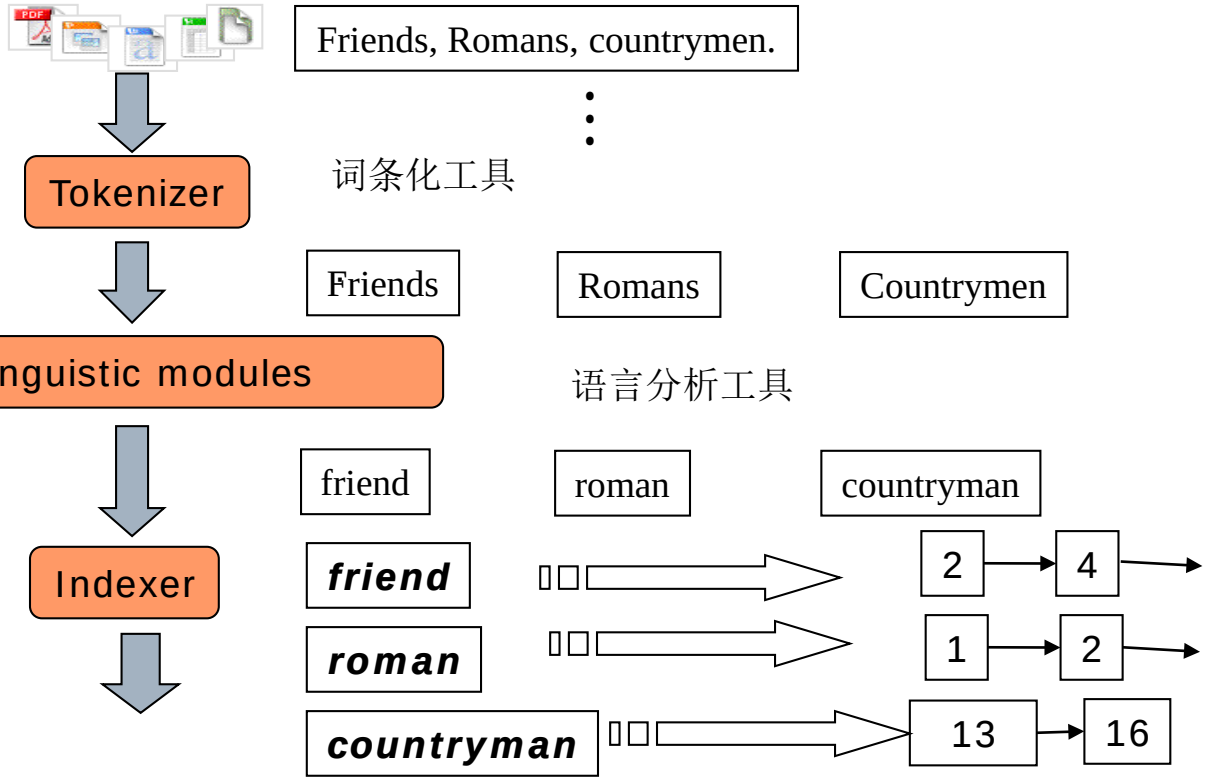
□ 倒排索引示意图



倒排索引

- 核心思想：对每个词项t，记录所有包含t的文档列表，其中每篇文档用docID表示
- 文档列表的数据结构：能否采用定长数组的方式来存储docID列表？
- 通常采用变长表方式存储倒排表
  - 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

倒排索引构建


待索引文档
Friends, Romans, countrymen.
Tokenizer
词条化工具
词条流
Friends
Romans
Countrymen
Linguistic modules
语言分析工具
与词项对应的词条
friend
roman
countryman
Indexer
倒排索引
friend
2
4
roman
1
2
countryman
13
16

索引构建过程：词条表生成

- 基于每篇文档中的词条生成<词条，docID>二元组列表

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

- 首先按词项排序
- 然后对每个相同词项按docID排序

索引构建的核心步骤

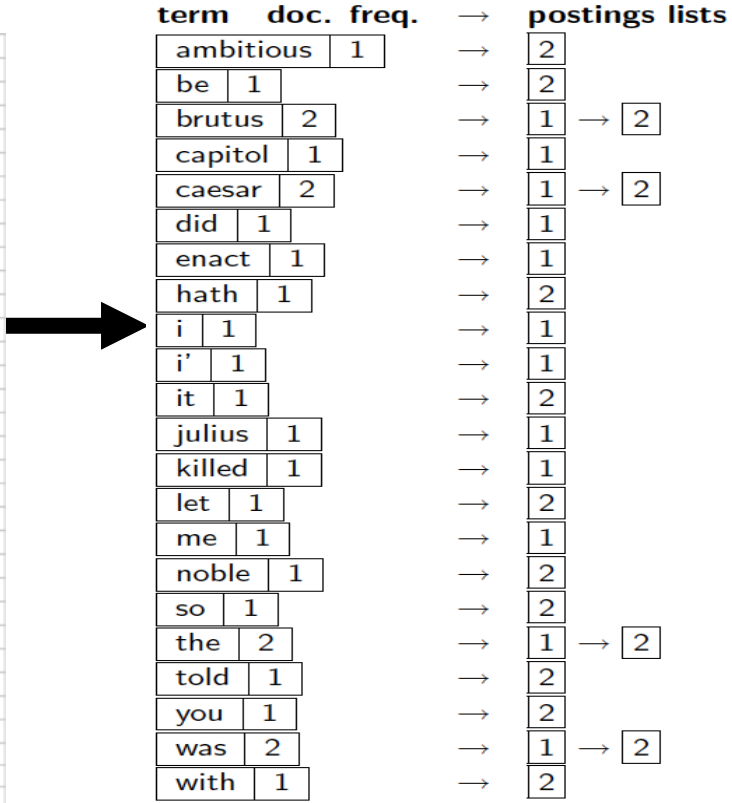
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
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

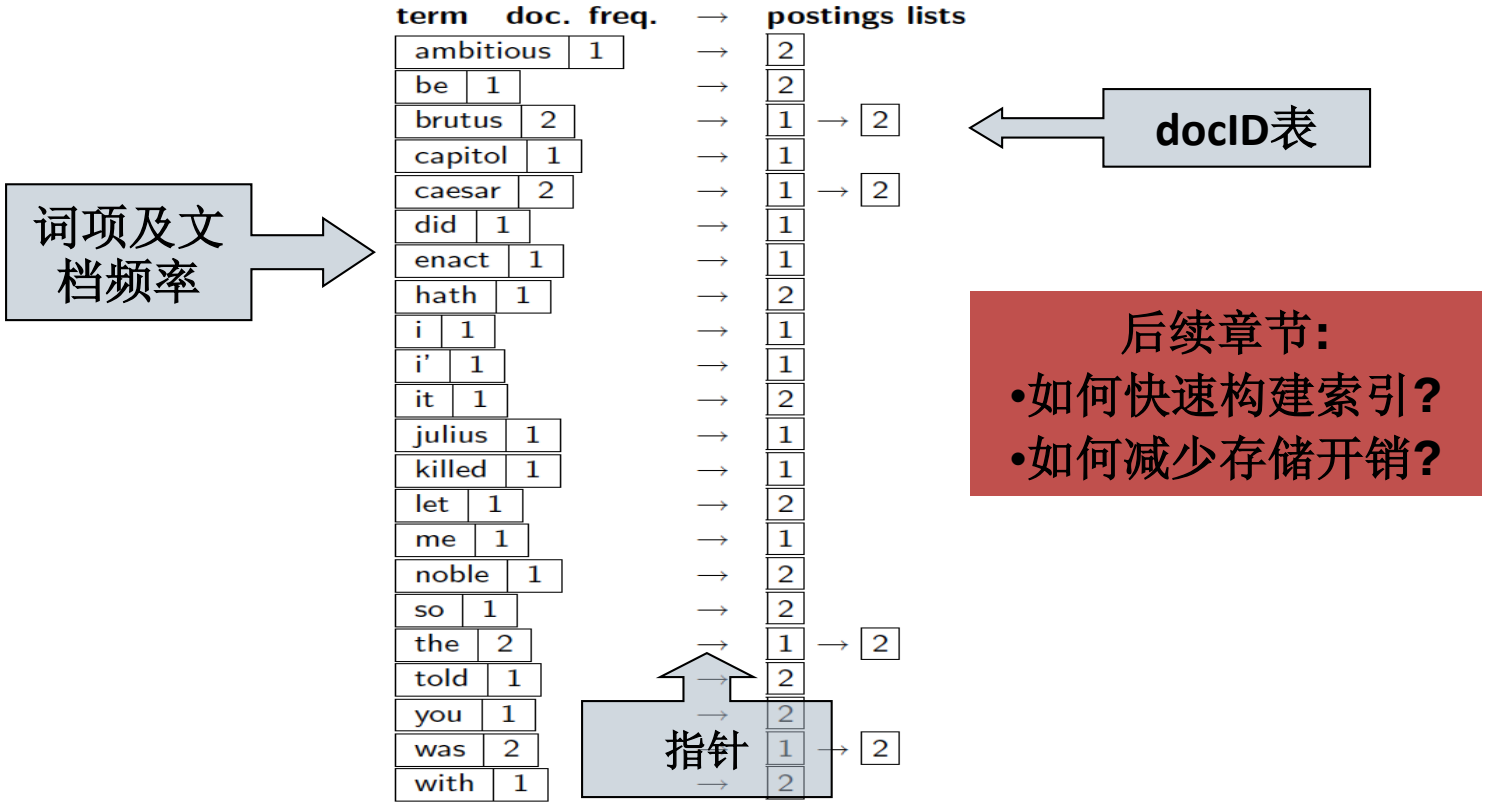
索引构建过程：词典&倒排表生成

- 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. frequency, DF)会被加入

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

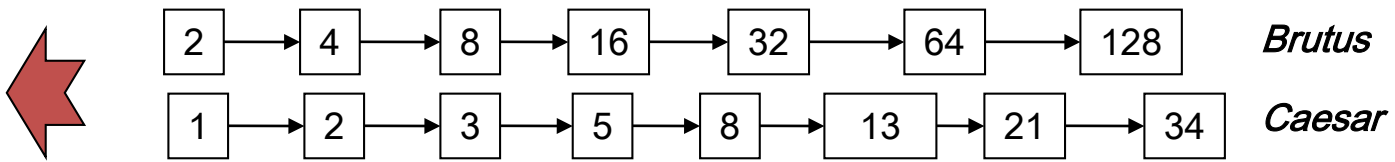


存储开销计算


term doc. freq.
postings lists
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
词项及文档频率
docID表
指针
后续章节:
•如何快速构建索引?
•如何减少存储开销?

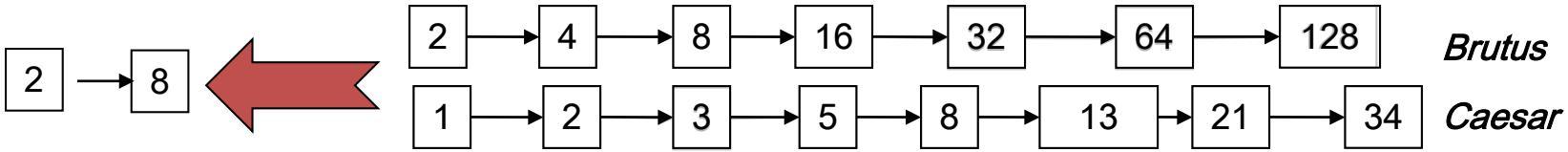
布尔查询的处理：AND

- 给定一个布尔查询：**Brutus AND Caesar**
- 首先在词典中定位 **Brutus**
  - 返回其倒排记录表
- 再在词典中定位**Caesar**
  - 返回其倒排记录表
- 合并(Merge)两个倒排记录表，即求交集



倒排表的合并

- 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的倒排记录，然后移动某个或两个指针。



- 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

合并算法的伪代码

```
INTERSECT(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 p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

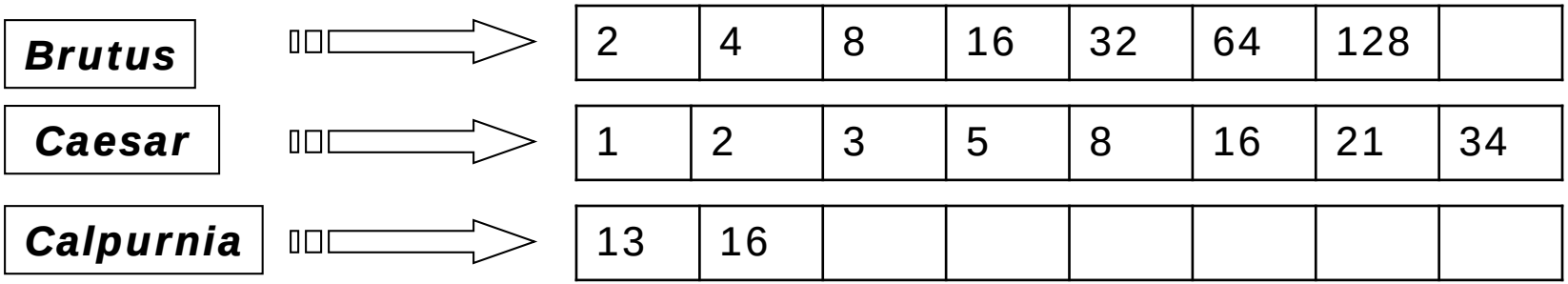
布尔查询的处理：OR/NOT

- **OR表达式：Brutus OR Caesar**
  - 两个倒排记录表的并集
- **NOT表达式：Brutus AND NOT Caesar**
  - 两个倒排记录表的差集
- 一般的布尔表达式

  (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)
- 查询处理依序而行？

查询处理的效率问题

- 考虑n 个词项的 AND查询
- 对每个词项，取出其倒排记录表，然后两两合并

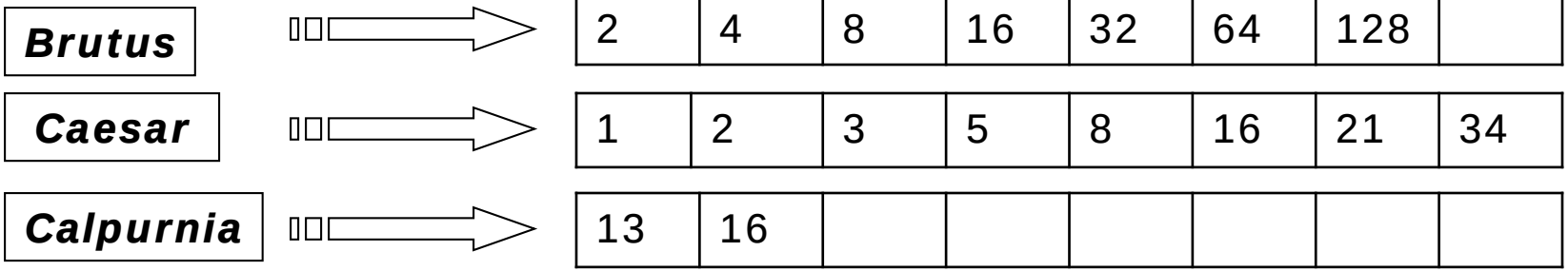


查询：***Brutus*** *AND* ***Caesar*** *AND* ***Calpurnia***

查询处理顺序的优化

- **优化策略：**按照表从小到大的顺序进行处理

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*

一般化的优化处理

- 针对一般的布尔表达式，首先进行形式转化，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)
- 然后获取每个词项的df，并通过将词项的df相加估计每个OR表达式对应的倒排记录表的大小
- 最后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，是构建IR系统的一种最简单方式
- 查询表达方式直观清晰，易于理解
- 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、文献检索系统、Mac OS X Spotlight工具

布尔检索系统：WestLaw

- 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付费用户
- 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

布尔检索系统：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....

布尔检索的不足

- 难以准确表达复杂的用户信息需求
  - 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 二零壹壹) 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

布尔检索的不足

- 不能对检索结果进行排序
- 检索输出量无法合理控制
- 难以支持相关反馈

参考资料

- 《信息检索导论》，第一章
- 莎士比亚全集:
  - http://www.rhymezone.com/shakespeare/
- Managing Gigabytes(深入搜索引擎), 3.2节
- 《现代信息检索》，8.2节

课后作业

- 见课程网页:

http://10.76.3.31