基于倒排索引的检索系统

苏州大学计算机学院

提纲

① 倒排索引

② 布尔查询的处理

一个简单的例子(金庸小说)

- 金庸的哪本小说包含**郭靖**和**黄蓉**但不包含洪七公？
  - 布尔表达式为 郭靖 AND 黄蓉 AND NOT 洪七公
- 笨方法： 从头到尾扫描所有小说，对每本小说判断它是否包含**郭靖**和**黄蓉**但不包含**洪七公**
- 笨方法为什么不好?
  - 速度超慢 (特别是大型文档集)
  - 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

	射雕英雄传	神雕侠侣	天龙八部	倚天屠龙记	鹿鼎记
郭靖	1	1	0	1	0
黄蓉	1	1	0	1	0
洪七公	1	1	0	0	0
张无忌	0	0	0	1	0
韦小宝	0	0	0	0	1

郭靖 AND 黄蓉 *BUT NOT* 洪七公

若某小说包含某单词，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*郭靖 AND* 黄蓉 *BUT NOT* 洪七公

- 取出三个行向量，并对 **洪七公**的行向量求反，最后按位进行与操作

- 11010 AND 11010 AND 00111 = 00010.

上述查询的结果文档

- 倚天屠龙记

IR中的基本假设

- 文档集Collection: 由固定数目的文档组成
- 目标: 返回与用户需求相关的文档并辅助用户来完成某项任务
- 相关性Relevance
  - 主观的概念
  - 反映对象的匹配程度
  - 不同应用相关性不同

检索效果的评价

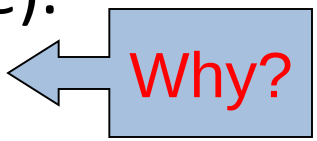
- 正确率(Precision) : 返回结果文档中正确的比例。如返回80篇文档，其中20篇相关，正确率1/4
- 召回率(Recall) : 全部相关文档中被返回的比例，如返回80篇文档，其中20篇相关，但是总的应该相关的文档是100篇，召回率1/5
- 正确率和召回率反映检索效果的两个方面，缺一不可。
  - 全部返回，正确率低，召回率100%
  - 只返回一个非常可靠的结果，正确率100%，召回率低

大文档集

- 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

词项-文档矩阵将非常大

- 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



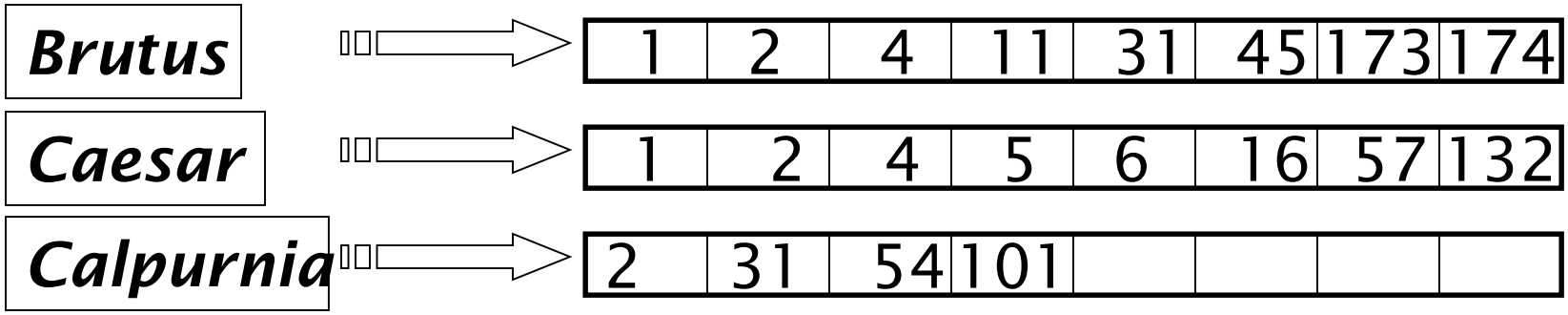
- 应该有更好的表示方式
  - 求方法？

词项-文档矩阵将非常大

- 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)

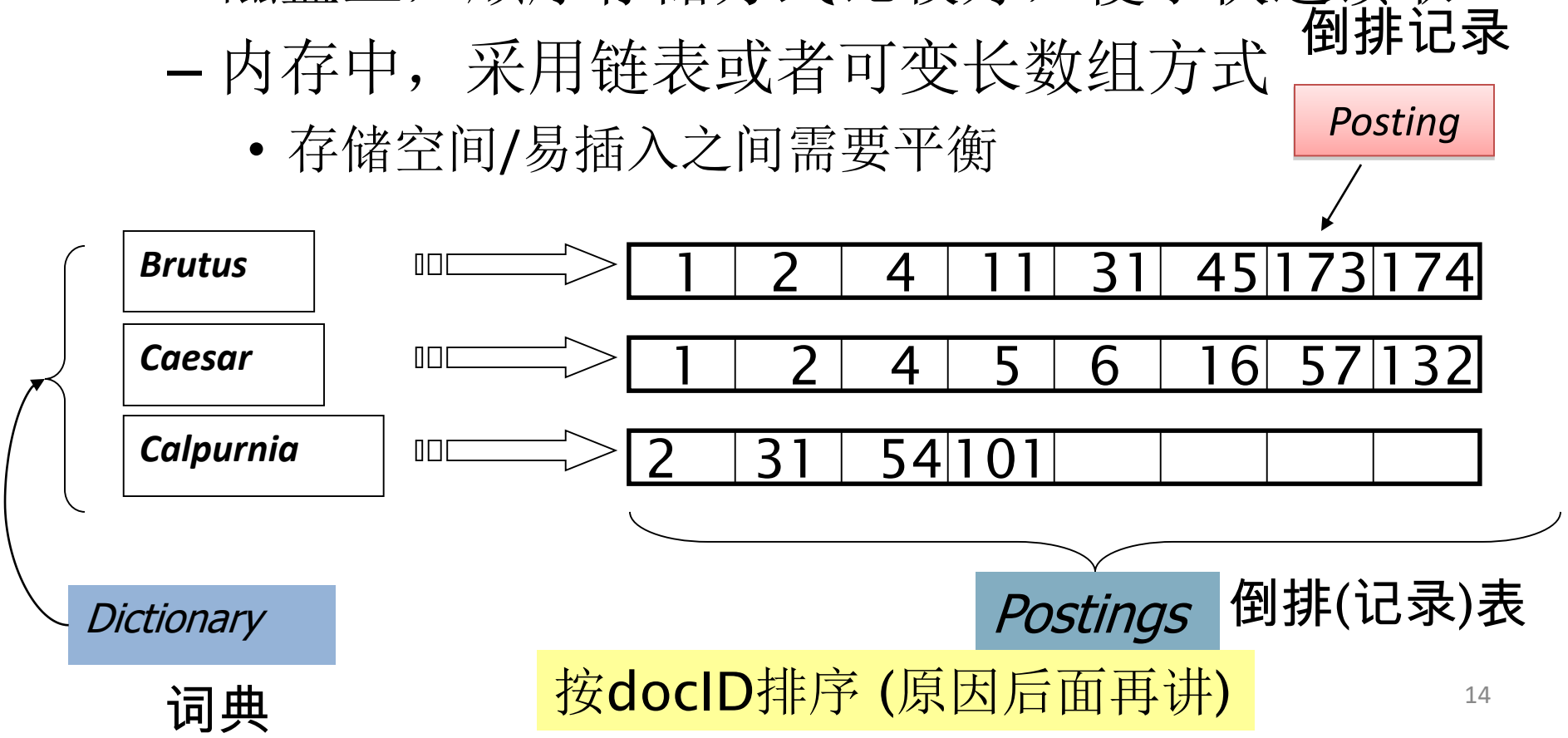
- 对每个词项t, 记录所有包含t的文档列表.
  - 每篇文档用一个唯一的 docID来表示，通常是正整数，如1,2,3…
- 能否采用定长数组的方式来存储docID列表



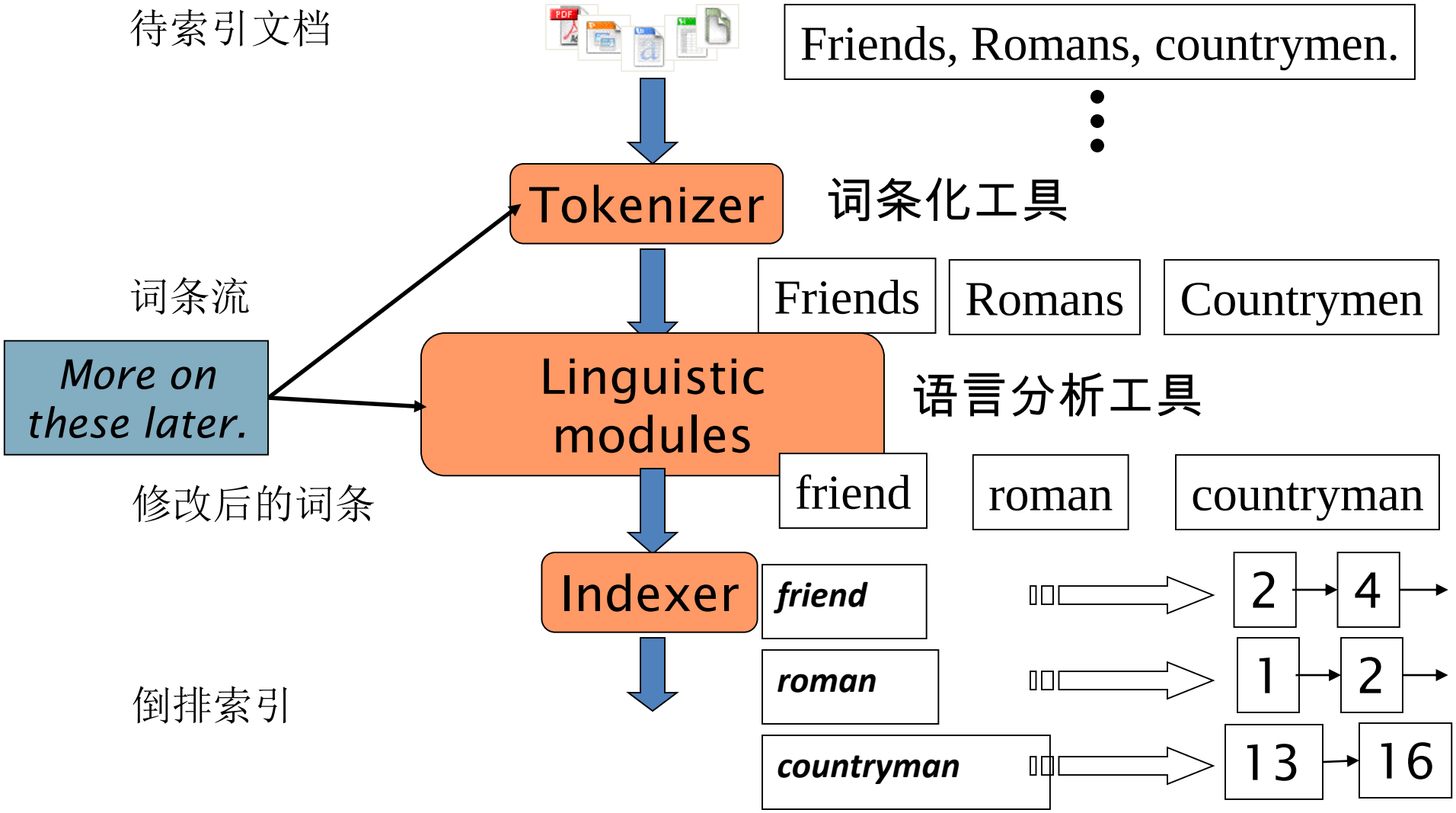
文档14中加入单词***Caesar***时该如何处理?

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



倒排索引构建


待索引文档
Friends, Romans, countrymen.
Tokenizer
词条化工具
词条流
Friends
Romans
Countrymen
More on these later.
Linguistic modules
语言分析工具
修改后的词条
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
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 & 倒排记录表

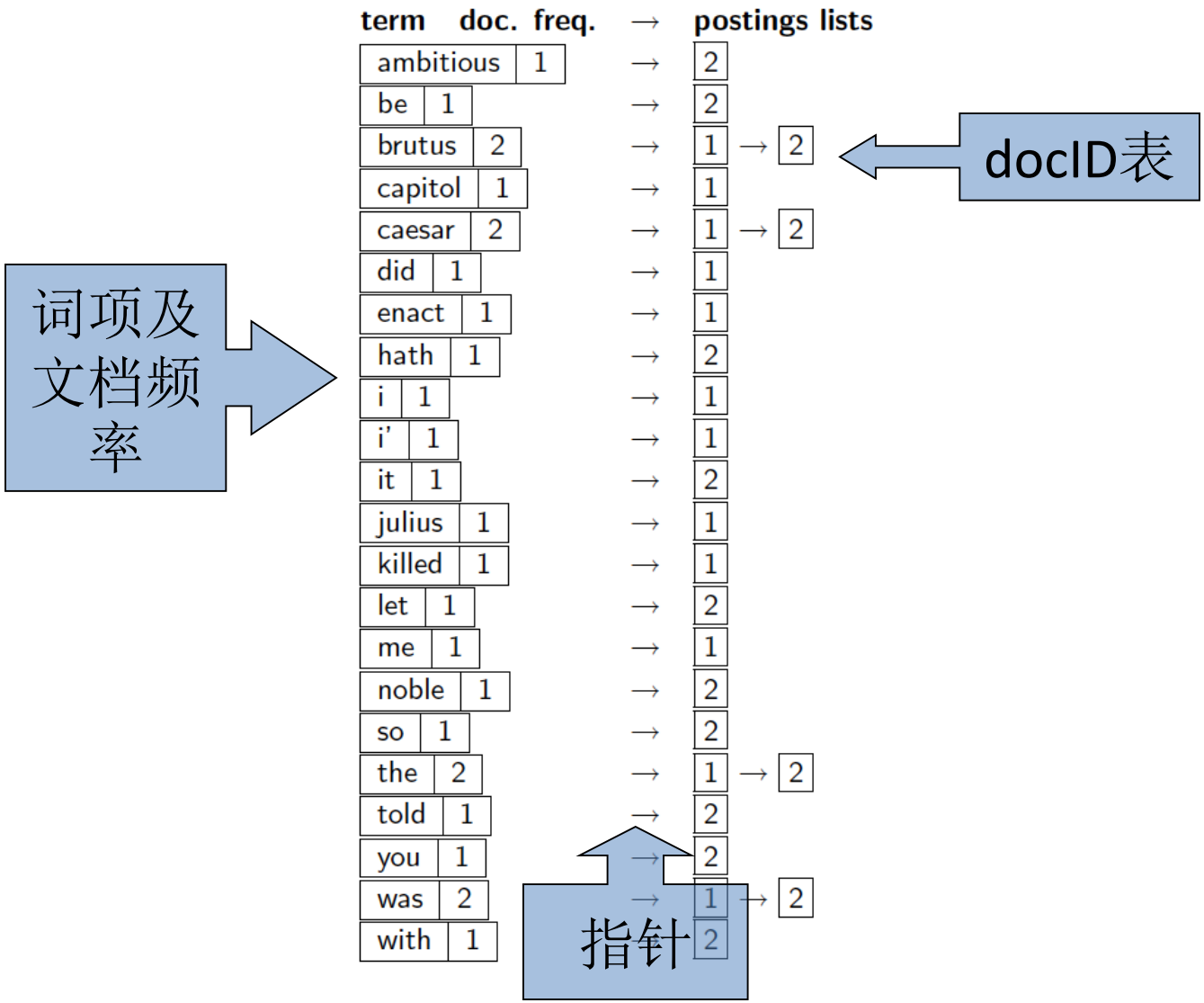
- 某个词项在单篇文档中的多次出现将被合并
- 拆分成词典和倒排记录表两部分
- 每个词项出现的文档数目(doc. freq. 或DF)会被加入

为什么加入？后面会讲

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

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	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
docID表
词项及文档频率
指针

提纲

① 倒排索引

② 布尔查询的处理 （继续）

假定索引已经构建好

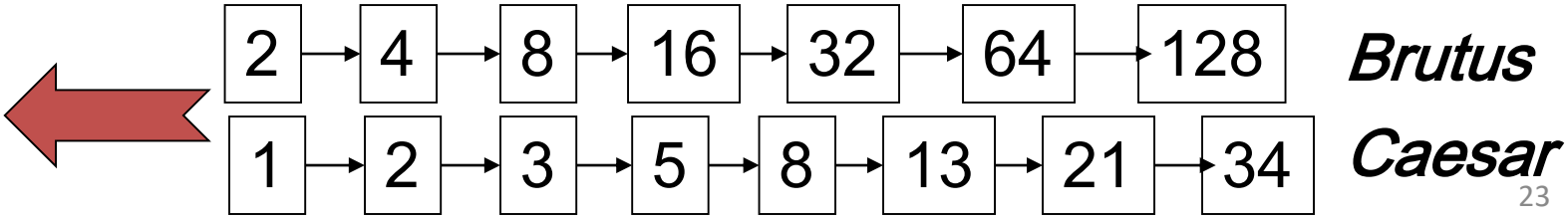
- 如何利用该索引来处理查询?

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

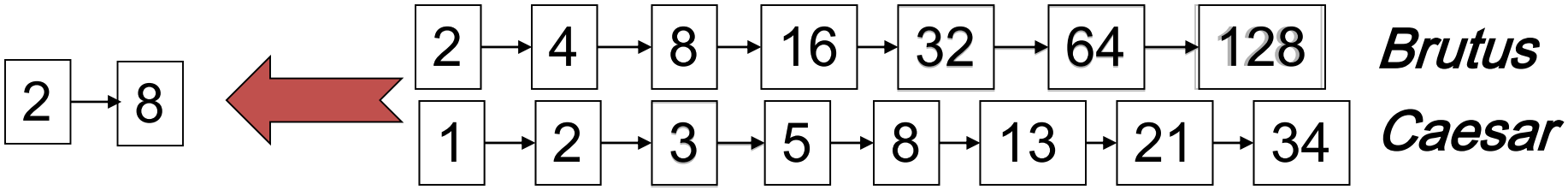
AND查询的处理

- 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

其它布尔查询的处理

- OR表达式： Brutus or Caesar
- 两个倒排记录表的并集

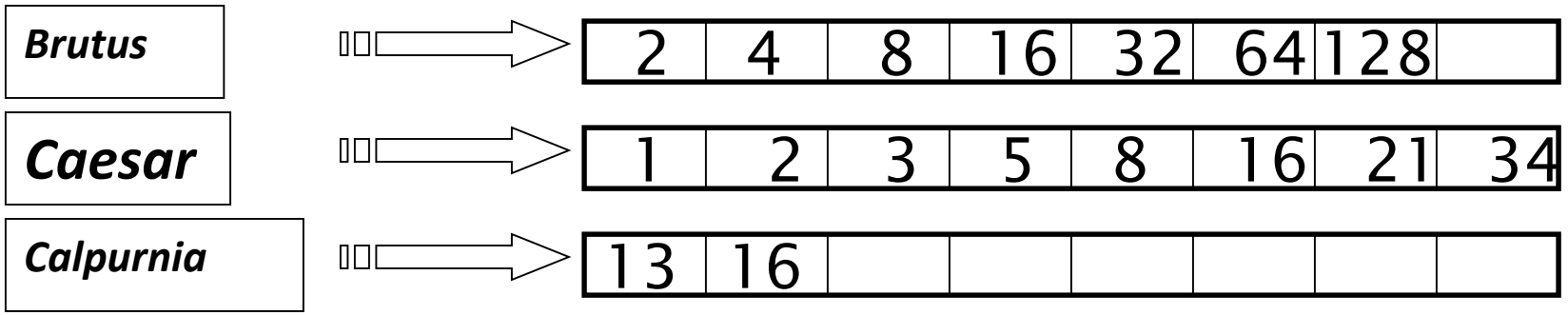
- NOT表达式： Brutus AND NOT Caesar
- 两个倒排记录表的减

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

- 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

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

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

布尔检索的缺点

- 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
- 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
- 不能对检索结果进行排序