

信息检索导论

An Introduction to Information Retrieval

第1讲 布尔检索

Boolean Retrieval

授课人：李波

中国科学院信息工程研究所/国科大网络空间安全学院

提纲

- ① 信息检索概述
- ② 倒排索引
- ③ 布尔查询的处理

上一讲回顾

- 什么是信息检索？
- 为什么要学习信息检索？
- 课程情况
- 布尔检索

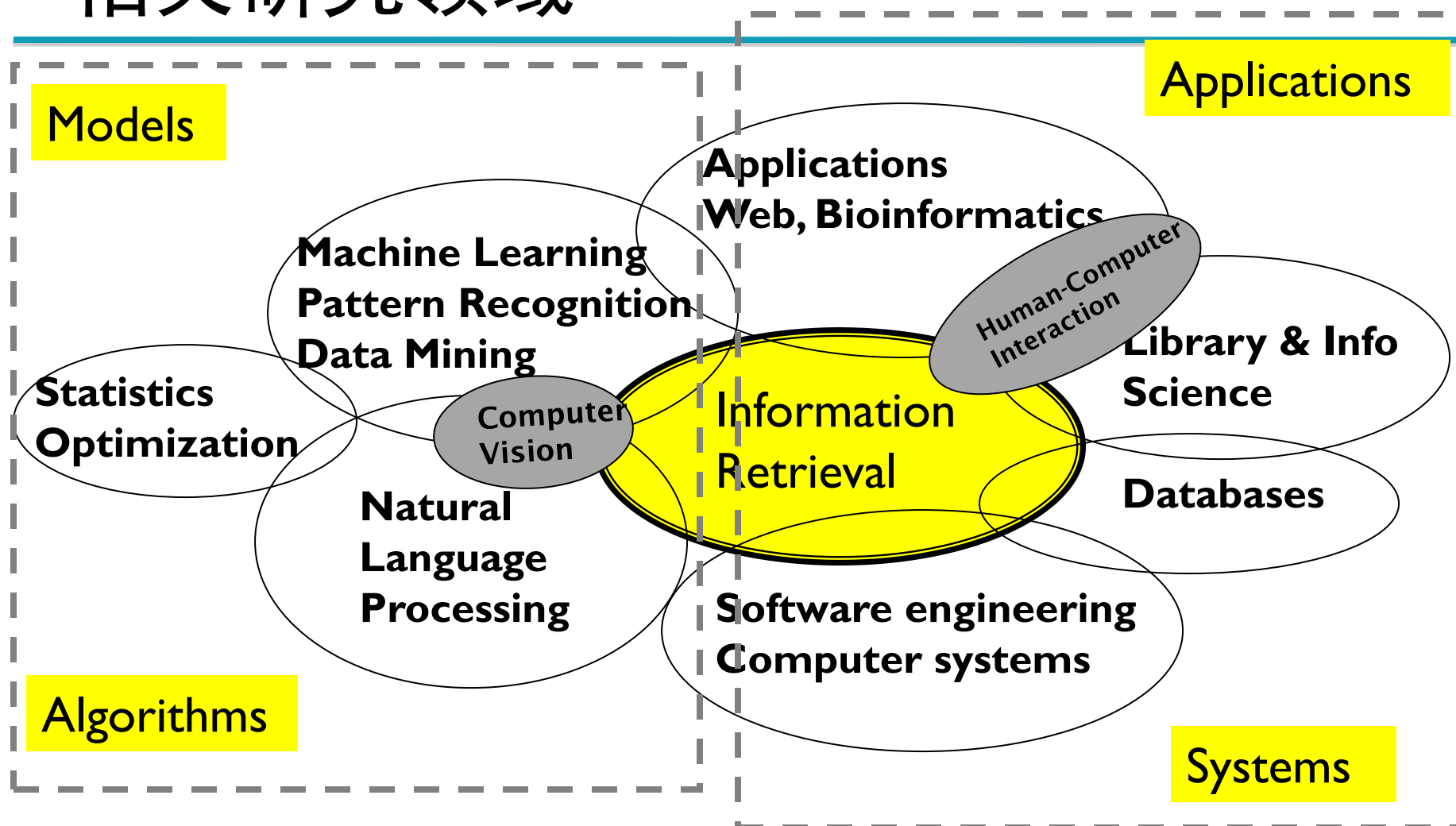
信息检索(Information Retrieval, IR)

- 给定用户需求返回满足该需求信息的一门学科。通常涉及信息的获取、存储、组织和访问。
- 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。
- “找对象”的学科，即定义并计算某种匹配“相似度”的学科。

IR: “rank documents by their relevance to the user’s information need”

给定用户的信息需求，按相关度对文档进行排序并返回给用户的一门学科

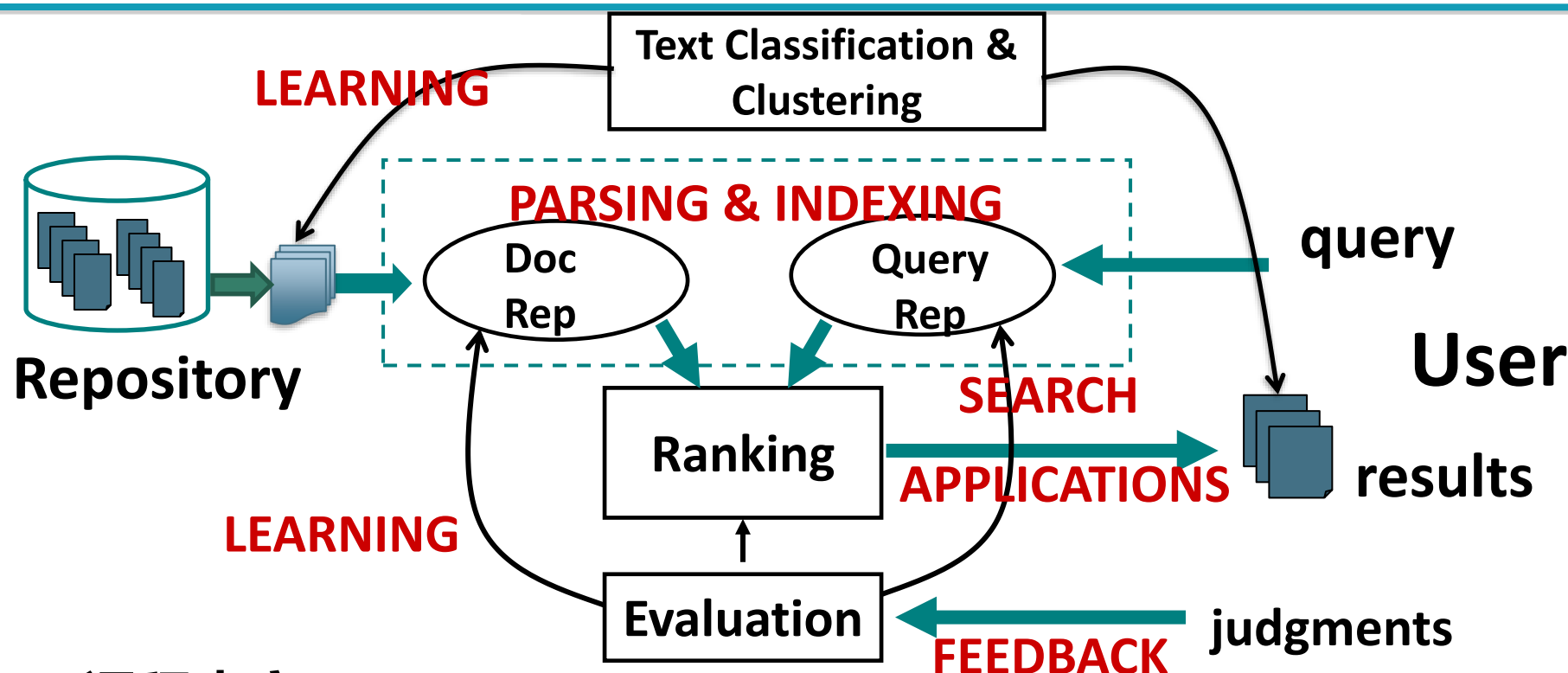
相关研究领域



信息检索发展历程（重要里程碑）

- 早期(late 1950s to 1960s): 学科创建阶段
 - Luhn在自动化索引构建方面的早期工作 Indexing: auto vs. manual
 - Cleverdon发起的Cranfield项目, 早期的评价方法及索引性能实验 Evaluation System
 - Salton在SMART系统和实验上的早期工作
- 1970s-1980s: 提出多种检索模型
 - 向量空间模型(Vector space model) Indexing + Search Theory
 - 概率模型(Probabilistic models)
- 1990s: 检索模型及IR新任务的进一步发展
 - 语言模型 Large-scale evaluation, beyond ad hoc retrieval
 - TREC evaluation
- 2000s至今: 更多的应用场景, Web search及与其它领域的交叉方向
 - Learning to rank Web search
 - Web search Machine learning
 - Scalability (e.g., MapReduce) Scalability
 - Real-time search

如何进行信息检索？



课程内容:

- 1) 搜索引擎基础理论 (Ch1-7章) ; 2) 检索模型 (Ch1/6/11/12) ;
- 3) 信息检索评价方法 (Ch8) ; 4) 相关反馈及查询扩展 (Ch9) ;
- 5) 文本分类聚类 (Ch13-18) ; 6) IR应用: Web搜索 (Ch19-21)

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的高级搜索？

提纲

- ① 信息检索概述
- ② 倒排索引
- ③ 布尔查询的处理

信息检索(Information Retrieval)

- 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

文本检索

本课程主要关注文本检索（Text Retrieval）

- (1) 文本检索是最早的检索应用，也仍然是目前最主要的应用
- (2) 文本检索理论可以用于其他领域
- (3)

IR vs 数据库: 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(free text)
- 允许
 - 关键词加上操作符号的查询
 - 如 奥运会 AND 游泳
 - 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，比如网页就是一种半结构化数据

<title>李甲主页</title>

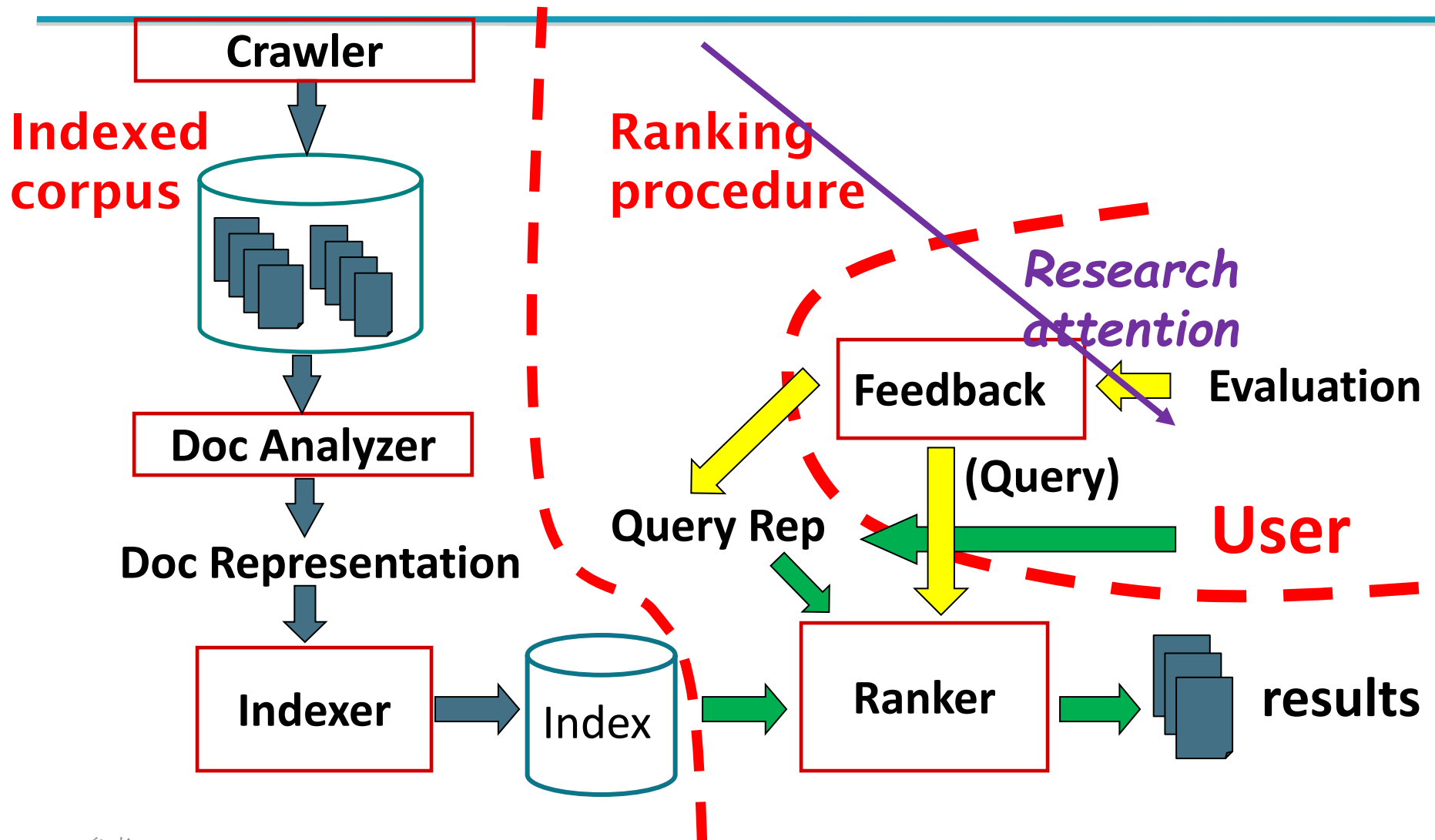
<body>...</body> ...

- 半结构化查询
 - Title contains data AND Bullets contain search
- 严格来说，即使是文本也是有“语言”结构
 - 比如主谓宾结构

传统信息检索 vs. 现代信息检索

- 传统信息检索主要关注非结构化、半结构化数据
- 现代信息检索中也处理结构化数据

Abstraction of search engine architecture



Core IR concepts

- Information need
 - “*an individual or group's desire to locate and obtain information to satisfy a conscious or unconscious need*” – wiki
 - An IR system is to satisfy users' information need
- Query
 - A designed representation of users' information need
 - In natural language, or some managed form

Core IR concepts

- Document

- A representation of information that potentially satisfies users' information need

- Text, ...

- Relevance

- Related to user's information need

- Multiple perspectives: topical, semantic, temporal, spatial, and etc.

**One sentence about IR -
“rank documents by their
relevance to the user's
information need”**

Key components in a search engine

- Web crawler
 - An automatic program that systematically browses the web for the purpose of Web content indexing and updating
- Document analyzer & indexer
 - Manage the crawled web content and provide efficient access of web documents

Key components in a search engine

- Query parser
 - Compile user-input keyword queries into managed system representation
- Ranking model
 - Sort candidate documents according to its relevance to the given query
- Result display
 - Present the retrieved results to users for satisfying their information need

Key components in a search engine

- Retrieval evaluation
 - Assess the quality of the returned results
- Relevance feedback
 - Propagate the quality judgment back to the system for search result refinement

Key components in a search engine

- Search query logs
 - Record users' interaction history with search engine
- User modeling
 - Understand users' longitudinal information need
 - Assess users' satisfaction towards search engine output

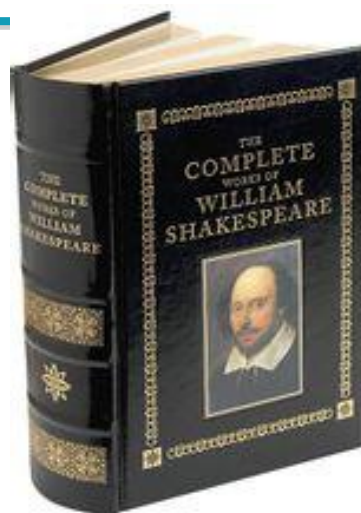
布尔检索

- 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的高级搜索？

提纲

- ① 信息检索概述
- ② 倒排索引
- ③ 布尔查询的处理

一个简单的例子(《莎士比亚全集》)



不到100万单词，假设每个英文单词平均长度为8字节，则整个全集不到10MB

- 需求：莎士比亚的哪部剧本包含Brutus及Caesar但是不包含Calpurnia？
 - 将需求表示为布尔表达式： Brutus AND Caesar AND NOT Calpurnia。

一个简单的例子(《莎士比亚全集》)

- 暴力方法： 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
- 暴力方法有什么不足？
 - 速度超慢 (特别是大型文档集)
 - 处理NOT Calpurnia 并不容易（不到末尾不能停止判断）
 - 不太容易支持其他操作 (e.g., 寻找靠近countrymen的单词Romans)
 - 不支持检索结果的(灵活)排序 (排序时只返回较好的结果)
- 暴力方法的优点？
 - 实现简单
 - 很容易支持文档动态变化

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(对应一篇文档)都是 0/1 向量, 每个0/1都对应一个词项。
- 关联矩阵的每一行(对应一个词项)也可以看成一个0/1向量, 每个0/1代表该词项在相应文档中的出现与否
- 给定查询 Brutus AND Caesar AND NOT Calpurnia
 - 取出三个词项对应的行向量, 并对 Calpurnia 的行向量求反, 最后按位进行与操作
 - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100.$

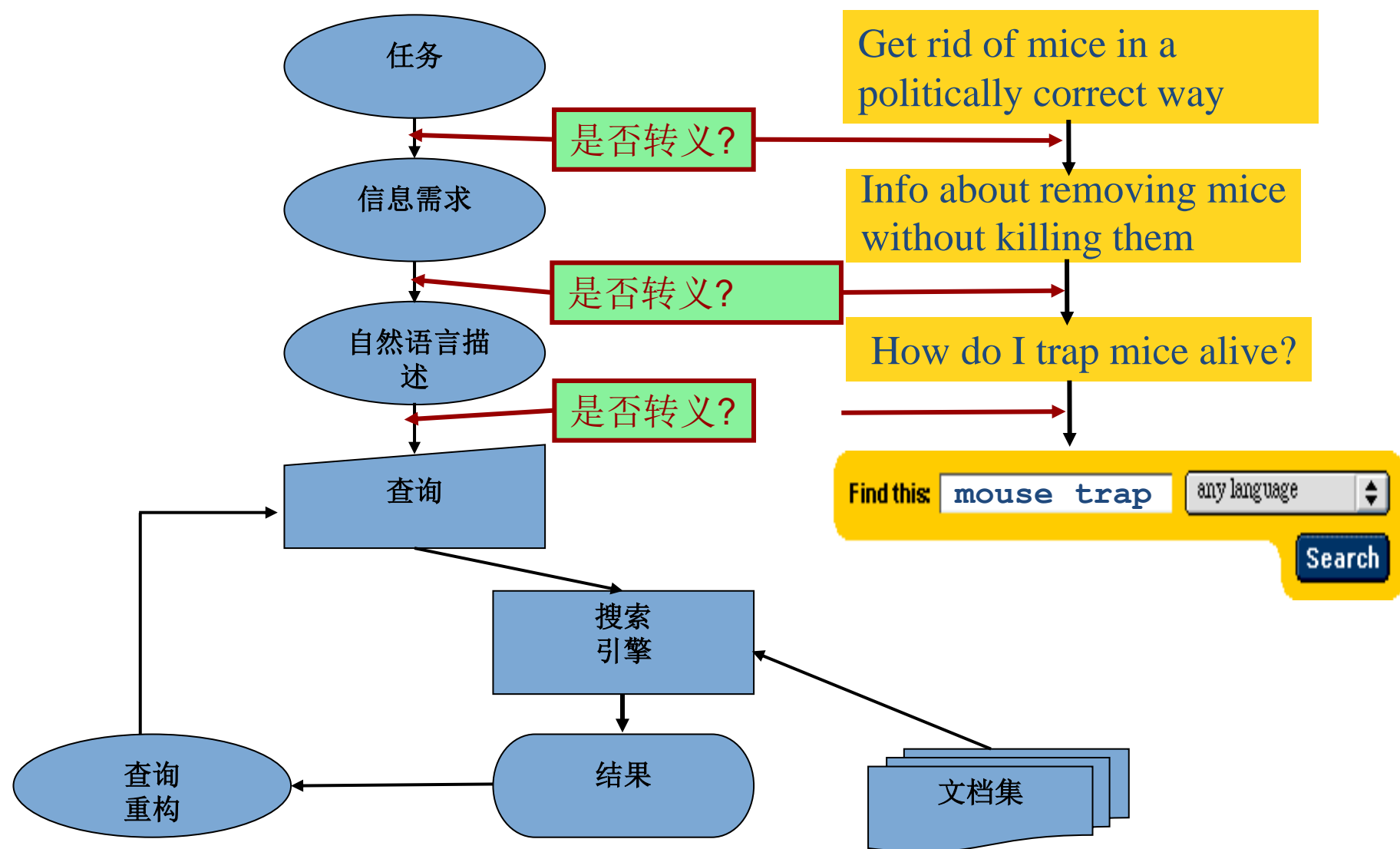
上述查询的结果文档

- 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.

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

典型的搜索过程



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
 - 将在后面介绍(有兴趣的可以先看)

更大的文档集

- 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) $M = 500K$

	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

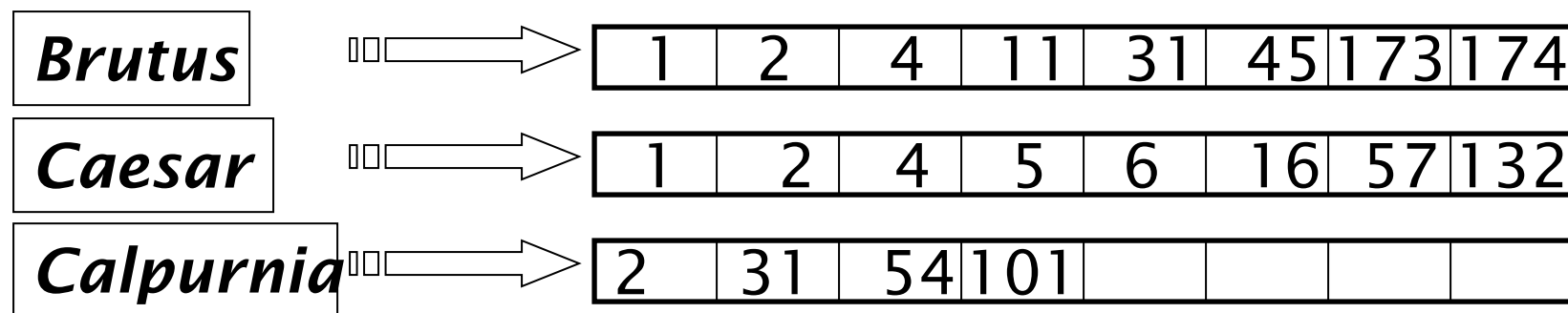
词项-文档矩阵将非常大

- 矩阵大小为 $500K \times 1M = 500G$
- 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
 - 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



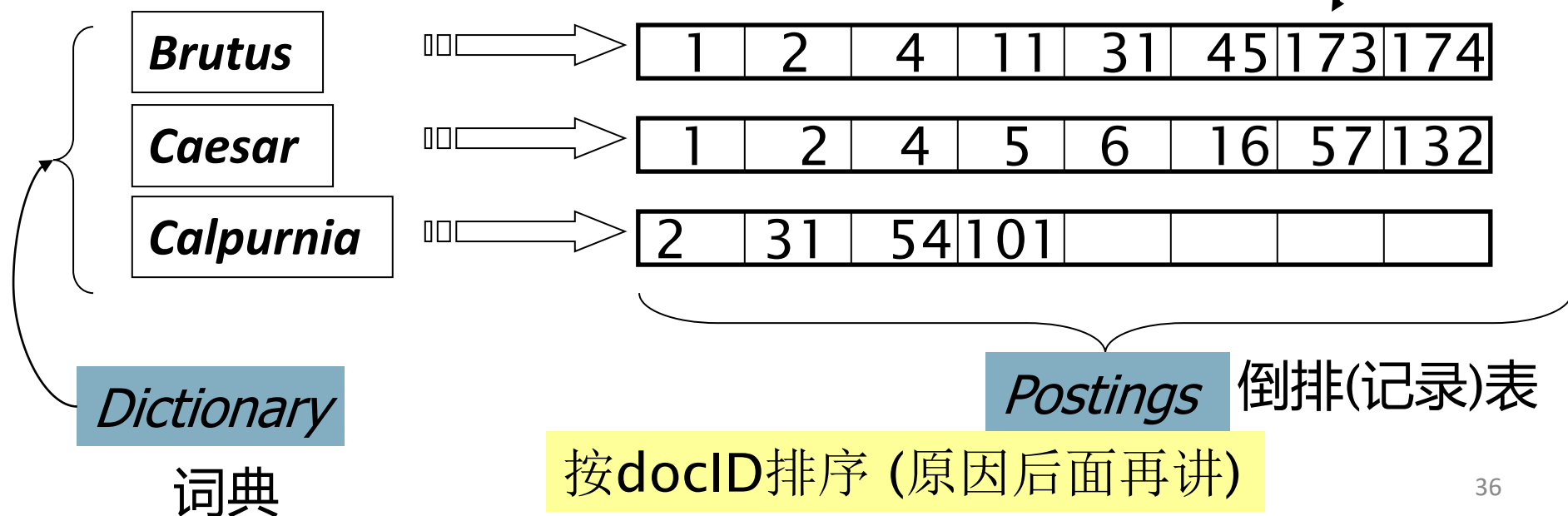
文档14中加入单词**Caesar**时该如何处理?

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

倒排记录

Posting



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

后面详细讲.

Linguistic
modules

语言分析工具

修改后的词条

friend

roman

countryman

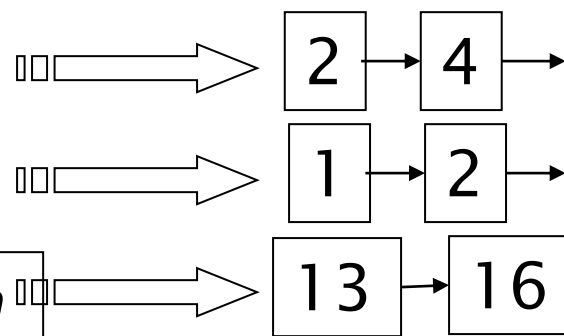
Indexer

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



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
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

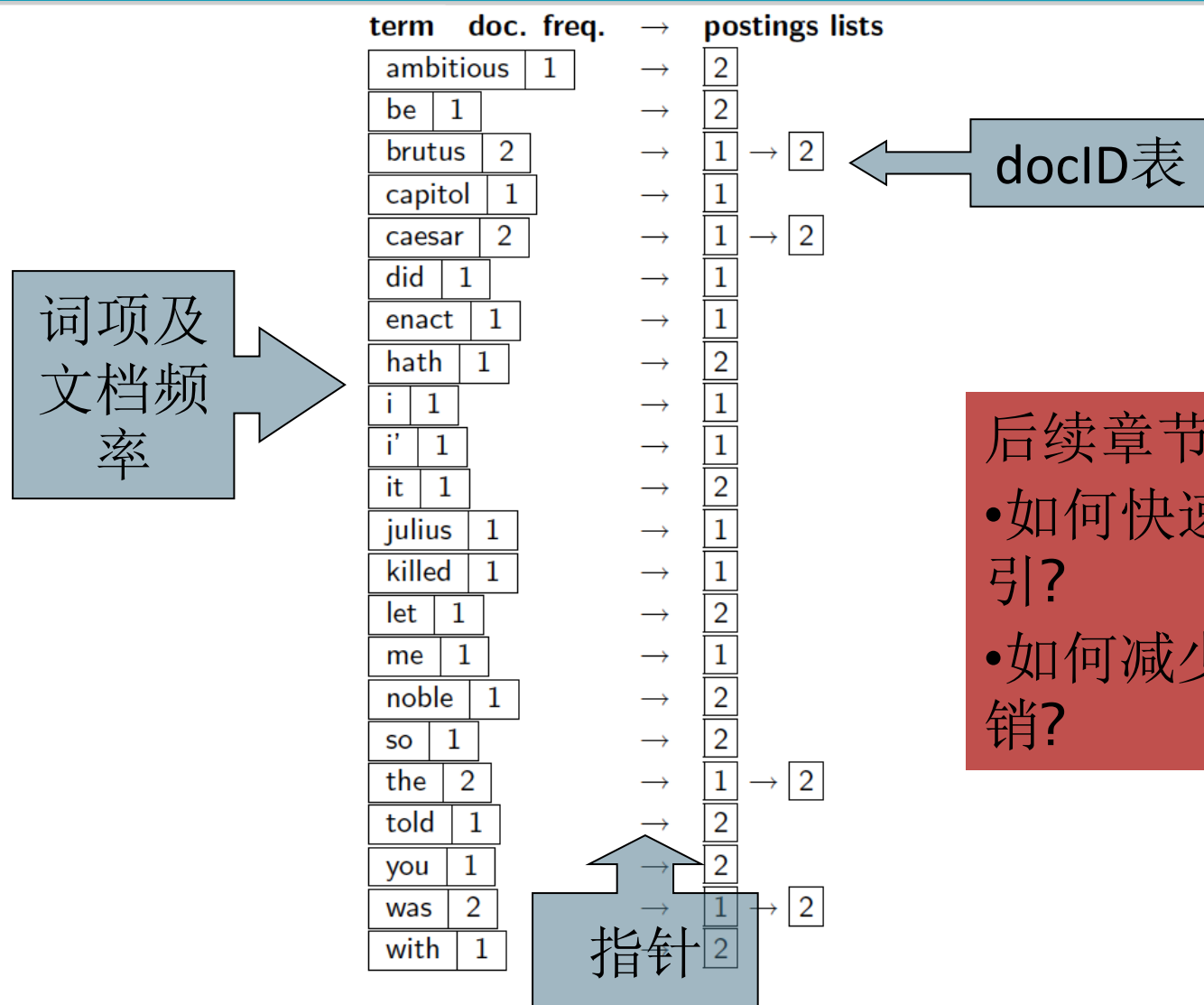
为什么加入? 后面会讲

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



后续章节:

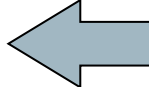
- 如何快速构建索引?
- 如何减少存储开销?

提纲

- ① 信息检索概述
- ② 倒排索引
- ③ 布尔查询的处理

假定索引已经构建好

- 如何利用该索引来处理查询?

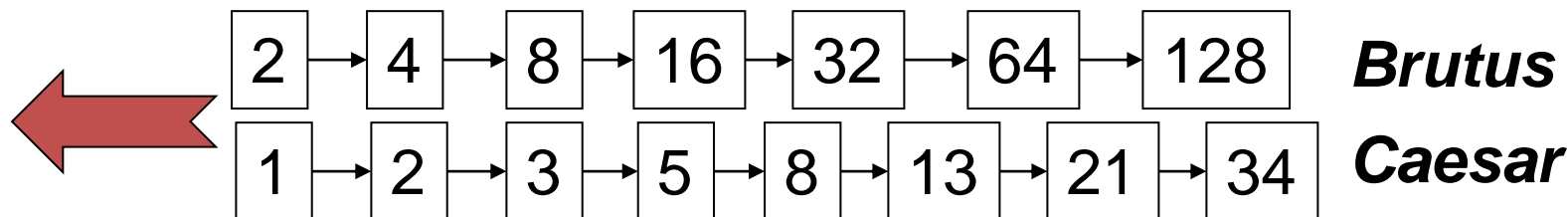


今天主
要内容

- 后面会讲 – 如何处理不同类型的查询? 比如带通配符的查询 “信息*检索”

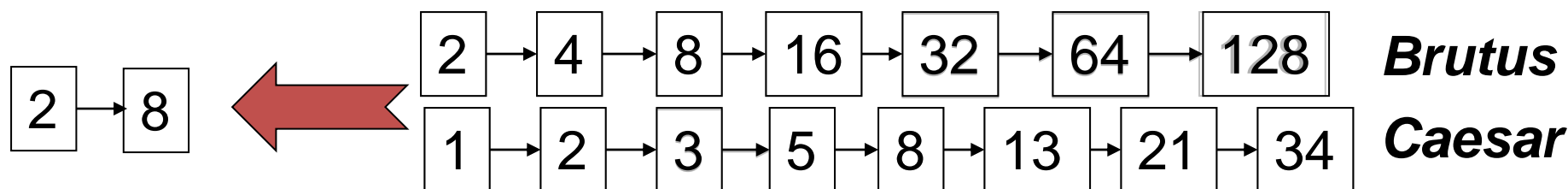
AND查询的处理

- 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

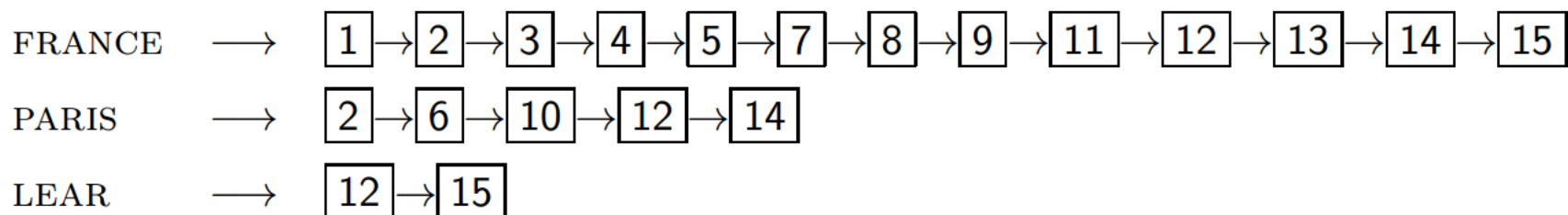
```
INTERSECT( $p_1, p_2$ )  
1   $answer \leftarrow \langle \rangle$   
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$   
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$   
4      then  $\text{ADD}(answer, \text{docID}(p_1))$   
5           $p_1 \leftarrow \text{next}(p_1)$   
6           $p_2 \leftarrow \text{next}(p_2)$   
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
 - 两个倒排记录表的并集
- NOT表达式: Brutus AND NOT Caesar
 - 倒排记录表A减去A和B的交集
- 一般的布尔表达式
- (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

练习

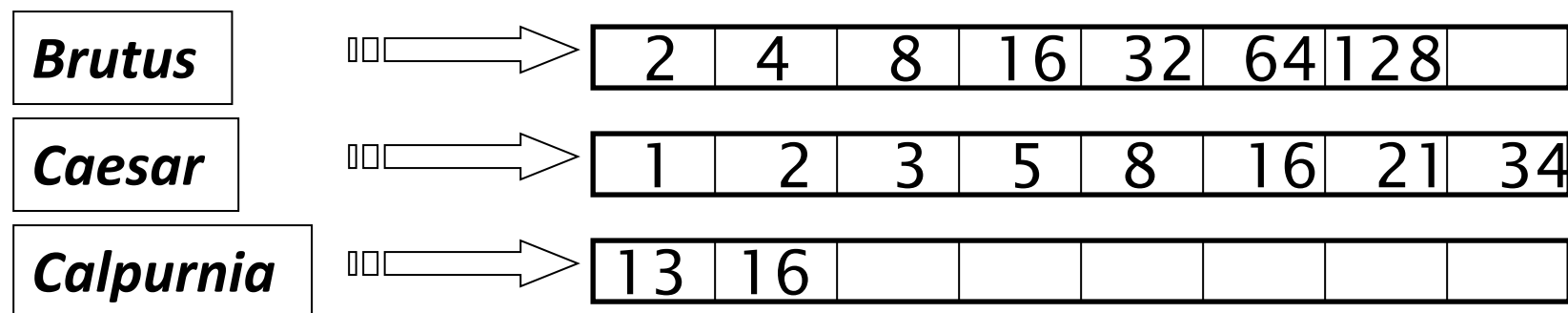
- 计算查询paris AND france AND lear的输出文档集



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

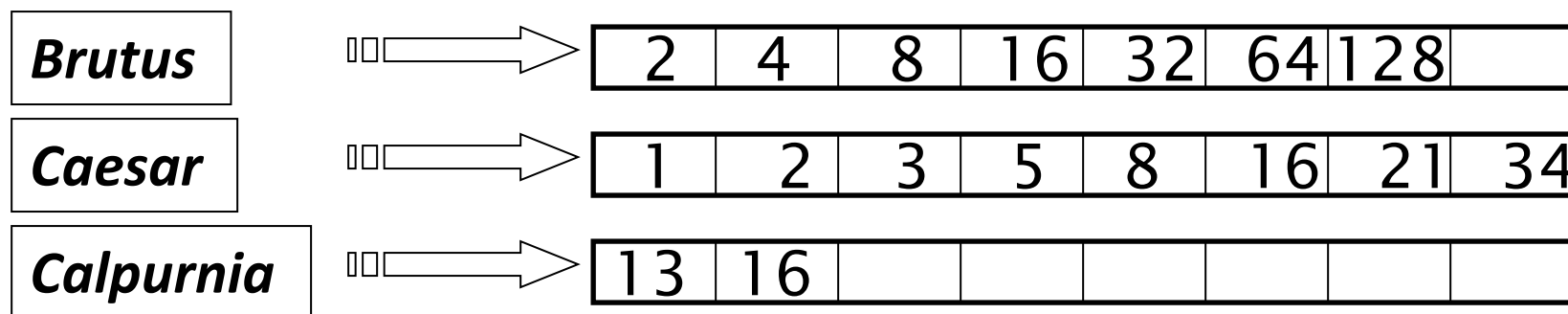


查询: *Brutus AND Calpurnia AND Caesar*

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**.

查询优化（组合查询的实现）

Intersect(<t1, ..., tn>)

- 1 terms \leftarrow SortByIncreasingFrequency(<t1, ..., tn>)
- 2 result \leftarrow postings(first/terms))
- 3 terms \leftarrow rest/terms))
- 4 **while** terms \neq nil and result \neq nil
- 5 **do** result \leftarrow Intersect(result, postings(first/terms)))
- 6 terms \leftarrow rest/terms))
- 7 **return** result

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

布尔检索的成功案例: WestLaw

<http://www.westlaw.com/>

- (付费用户数目)最大的商业化法律搜索服务引擎
 - 几十T数据, 700,000用户
- 1975年开始提供服务; 1992年加入排序功能
- 2005年, 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, /p = within a paragraph, Space is disjunction

布尔检索的成功案例: WestLaw

<http://www.westlaw.com/>

- 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年 OR 9年前) 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

- 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

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

课后阅读

- *Bush, Vannevar. "As we may think." The atlantic monthly 176, no.1 (1945): 101-108.*
- M. Sanderson and W. B. Croft, “The History of Information Retrieval Research,” *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1444–1451, May 2012.
- S. Robertson, “A Brief History of Search Results Ranking,” *IEEE Annals of the History of Computing*, vol. 41, no. 2, pp. 22–28, Apr. 2019.

课后思考

- 1. 列出一个课件中没有给出的应用系统，其本质其实是一个信息检索系统，说明原因。
- 2. 搜索系统和推荐系统都是信息检索系统，给出它们的相同点(至少3条)和不同点(至少3条)。
- 3. 用户信息需求和查询不是一回事，好的查询能较好地反映原始的用户信息需求，利用搜索引擎进行实践，给出某个信息需求，再给出两个查询，其中一个查询的搜索结果较好，而另一个较差。