信息检索与数据挖掘

第2章 布尔检索和倒排索引

课程内容

- 第1章 绪论
- 第2章 布尔检索及倒排索引
- 第3章 词典查找及扩展的倒排索引
- 第4章 索引构建和索引压缩
- 第5章 向量模型及检索系统
- 第6章 检索的评价
- 第7章 相关反馈和查询扩展
- 第8章 概率模型
- 第9章 基于语言建模的检索模型
- 第10章 文本分类
- 第11章 文本聚类
- 第12章 Web搜索
- 第13章 多媒体信息检索
- 第14章 其他应用简介

1.1.2 信息检索

- Information Retrieval这个术语产生于 Calvin Mooers 1948年在MIT的硕士论文。
  - Information Retrieval(IR)：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

Mooers Law： An information retrieval system will tend not to be used whenever it is more painful and troublesome for a customer to have information than for him not to have it. Where an information retrieval system tends not to be used, a more capable information retrieval system may tend to be used even less.

穆尔斯定律：当拥有信息比不拥有信息会使用户付出更大的努力或给用户造成更大的麻烦时。用户会倾向于不使用信息检索系统。

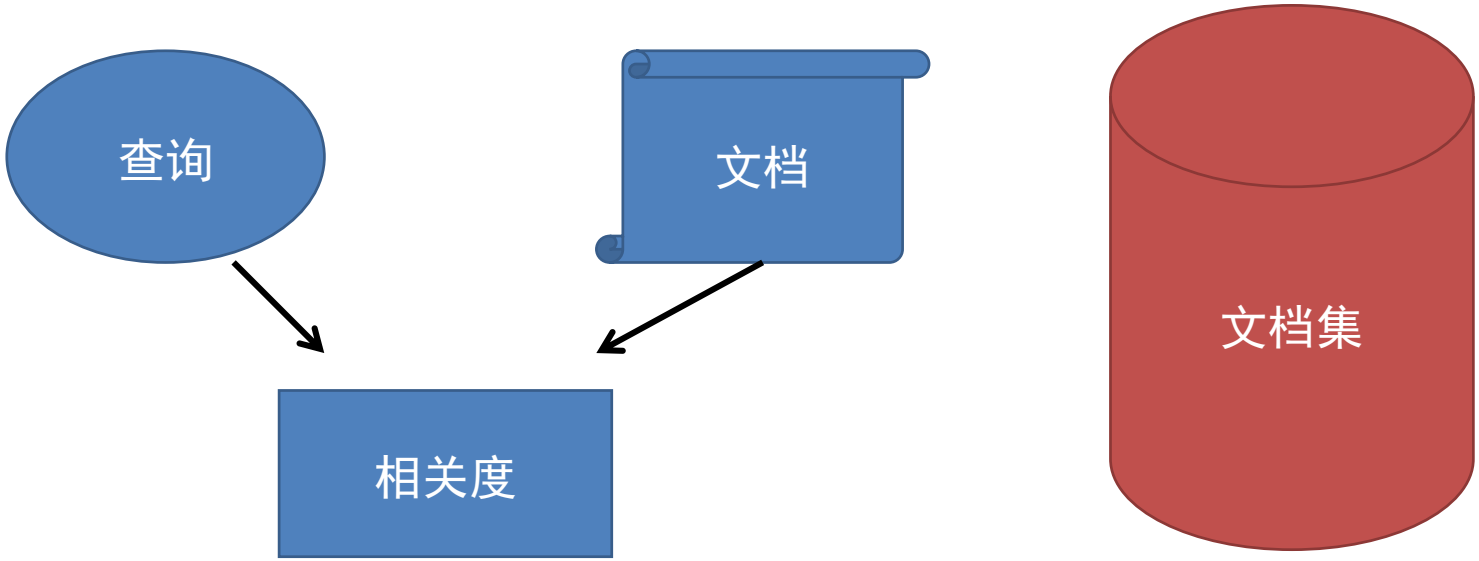
1.1.3 数据挖掘

- 数据挖掘（Data Mining）从*大量的*、*不完全的*、*有噪声的*、*模糊的*、*随机的*实际应用数据中，提取隐含在其中的、人们事先不知道的、但又是潜在有用的信息和知识的过程
- 数据挖掘的基本内容
  - 特征提取、分类、聚类
  - 话题检测、自动摘要
  - 智能问答等

信息检索可以帮助人们从海量的数据中快速的找到有用的信息

数据挖掘可以从大数据中提取出隐含的、先前未知的并有潜在价值的信息

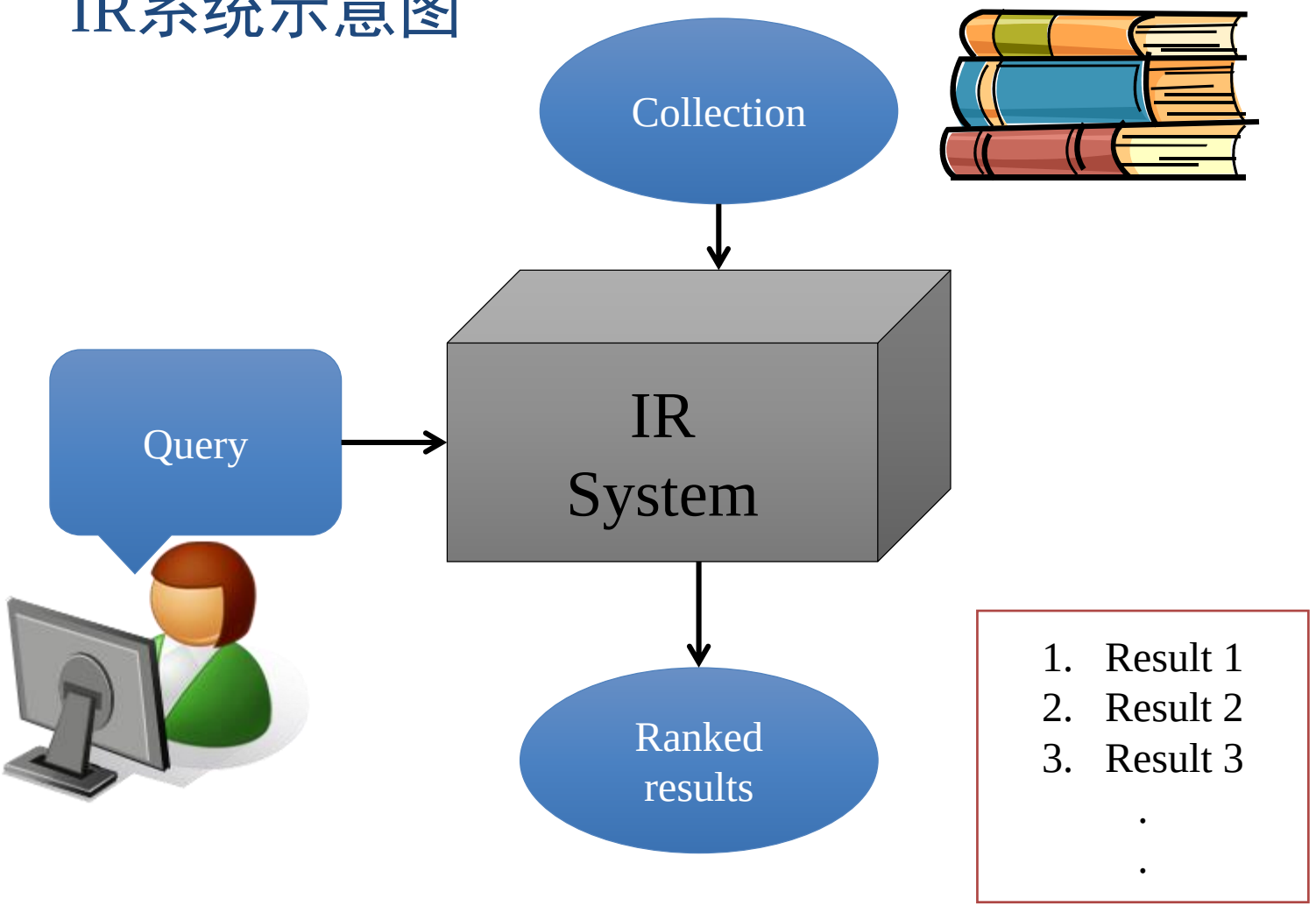
1.4.1 信息检索的基本概念



确定文档和查询之间的相关度是IR的核心问题

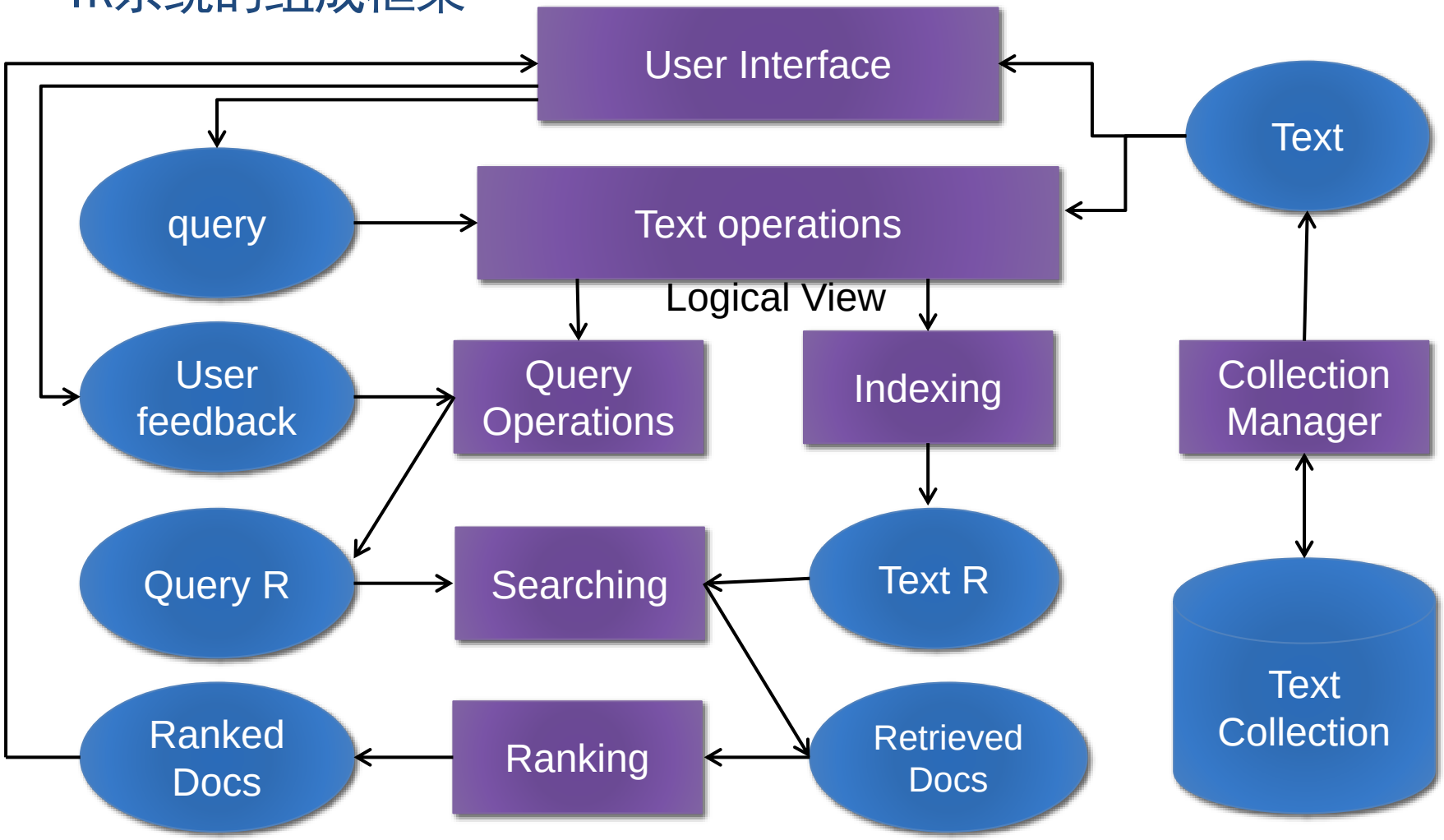
1.4.2 信息检索系统的基本组成

IR系统示意图



1.4.2 信息检索系统的基本组成

IR系统的组成框架



1.4.2 信息检索系统的基本组成

- **用户接口(User Interface)：用户和IR系统的人机接口**
  - 输入查询(Query)
  - 返回排序后的结果文档(Ranked Docs)并对其进行可视化(Visualization)
  - 支持用户进行相关反馈(Feedback)
- **用户的两种任务：retrieval 或者 browsing**
- **IR的两种模式： pull (ad hoc) 或者 push (filtering)**
  - Pull: 用户是主动的发起请求，在一个相对稳定的数据集合上进行查询
  - Push:用户事先定义自己的兴趣，系统在不断到来的流动数据上进行操作，将满足用户兴趣的数据推送给用户

1.4.2 信息检索系统的基本组成

- **文本处理(Text Operations)：对查询和文本进行的预处理操作**
  - 中文分词(Chinese Word Segmentation)
  - 词干还原(Stemming)
  - 停用词消除(Stop-word removal)
- **查询处理(Query operations)：对经过文本处理后的查询进行进一步处理，得到查询的内部表示(Query Representation)**
  - 查询扩展(Query Expansion)：利用同义词或者近义词对查询进行扩展
  - 查询重构(Query Reconstruction)：利用用户的相关反馈信息对查询进行修改
- **文本索引(Indexing)：对经过文本处理后的文本进行进一步处理，得到文本的内部表示(Text Representation)，通常基于索引项(Term)来表示**
  - 向量化、概率计算
  - 组成成倒排表进行存储

1.4.2 信息检索系统的基本组成

- 搜索(Searching)：从文本中查找包含查询中索引项的文本
- 排序(Ranking)：对搜索出的文本按照某种方式来计算其相关度
- Logical View：指的是查询或者文本的表示，通常采用一些关键词或者索引项(index term)来表示一段查询或者文本。

1.5.4 授课内容

- **第一章　绪论**

信息检索的典型应用。信息检索的基本概念和发展历史。信息检索和其他相关学科(自然语言处理、机器学习、概率统计、模式识别、数据库、数据挖掘等等)的关系。信息检索系统的基本构架和一般流程。

- **第二章 布尔检索及倒排索引**

字符串匹配及倒排索引。布尔查询处理及其优化。扩展的布尔操作。短语查询的处理。布尔检索模型及其扩展。

- **第三章 词典查找及扩展的倒排索引**

支持词典快速查找的数据结构(哈希表、二叉树等)。支持通配查询处理的索引结构。支持拼写或发音纠错处理的索引结构。

1.5.4 授课内容

- **第四章 索引构建和索引压缩**

文本预处理。一般构建过程。基于块排序的构建过程。单遍内存式扫描构建方法。分布式及动态索引方法。词项的统计特性。词典的压缩。倒排记录表的压缩。

- **第五章 向量模型及检索系统**

向量空间模型及词项权重计算机制。检索中的快速实现方法。检索系统的一般构成。隐性语义索引方法。基于开源工具搭建简单搜索引擎。

- **第六章 检索的评价**

效率和效果的评价。查全率和查准率。其他效果评价方法。用户体验及结果摘要。相关评测语料和评测会议。

1.5.4 授课内容

- **第七章 相关反馈和查询扩展**

相关反馈和伪相关反馈。查询扩展及重构。全局方法及局部方法。

- **第八章 概率模型**

概率排序原理。回归模型。二值独立概率模型。OKAPI BM25公式。

- **第九章 基于语言建模的检索模型**

查询似然模型。其他语言模型。语言模型的相关反馈。

- **第十章 文本分类**

文本分类的概念及评价方法。文本分类中的特征选择方法。

1.5.4 授课内容

- **第十一章 文本聚类**

文本聚类的概念及评价方法。文本聚类算法。检索结果聚类的标签生成。

- **第十二章 Web搜索**

Web结构。信息采集。网页查重方法。链接分析算法(PageRank和HITS)。

- **第十三章 多媒体信息检索**

自动图像标注，语义距离的度量，图像搜索，视频概念检测

- **第十四章 其他应用简介**

数字图书馆，过滤及推送系统、XML检索、跨语言检索、信息抽取、问答系统、互联网广告系统等等。

课程内容

- 第1章 绪论
- 第2章 布尔检索及倒排索引
- 第3章 词典查找及扩展的倒排索引
- 第4章 索引构建和索引压缩
- 第5章 向量模型及检索系统
- 第6章 检索的评价
- 第7章 相关反馈和查询扩展
- 第8章 概率模型
- 第9章 基于语言建模的检索模型
- 第10章 文本分类
- 第11章 文本聚类
- 第12章 Web搜索
- 第13章 多媒体信息检索
- 第14章 其他应用简介

第2章 布尔检索及倒排索引

- **2.1 信息检索模型概述**
- **2.2 一个简单的搜索示例**
- **2.3 倒排索引**
- **2.4 布尔检索模型**
- **2.5 布尔检索模型的优化与扩展**

如何度量相关性？

- 确定文档和查询之间的相关度是IR的核心问题

Query:

Bill Clinton

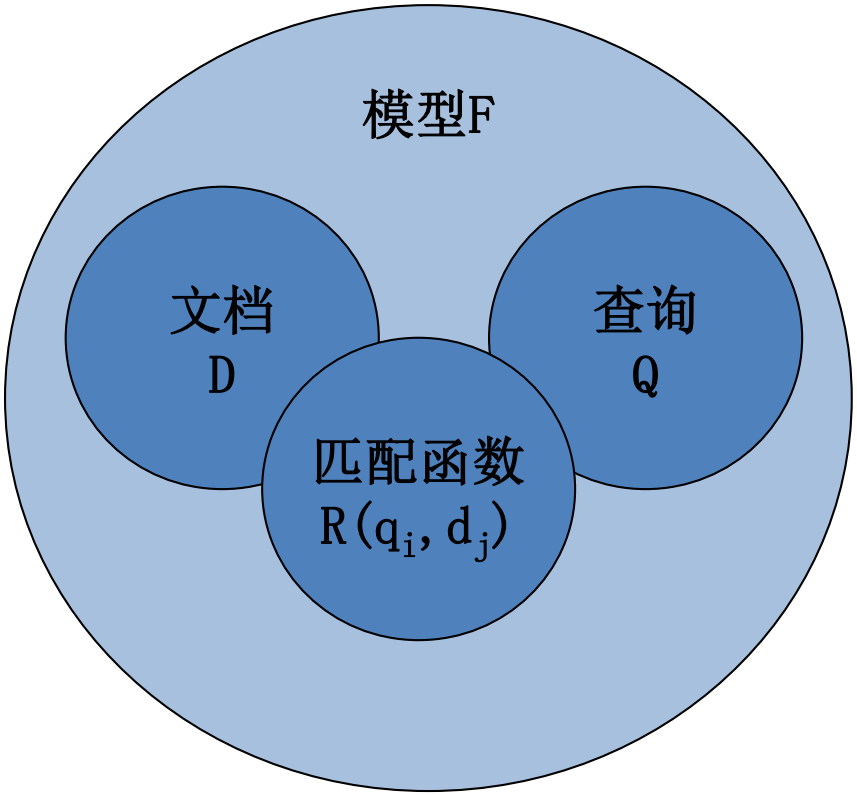
Document:

Relevant?

How relevant?

检索模型的定义

- 信息检索模型是描述信息检索中的文档、查询和它们之间的关系(匹配函数)的数学模型。

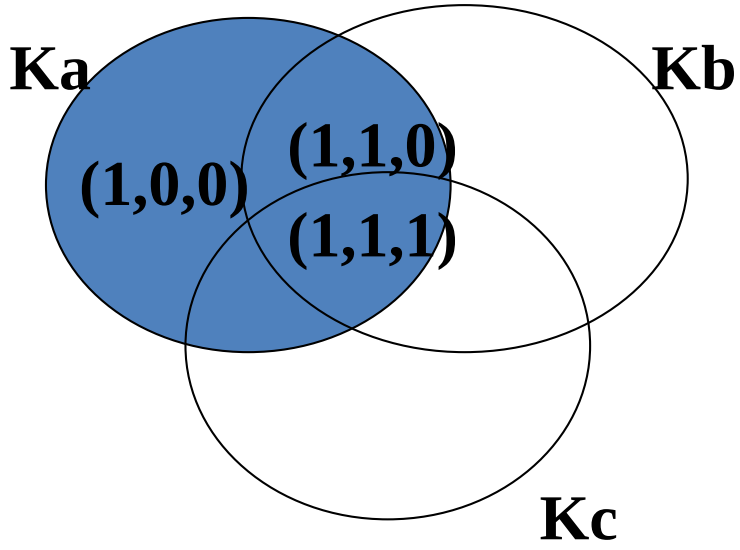


两大类信息检索模型

- 基于文本内容的检索模型
  - 布尔模型　　第2章 布尔检索及倒排索引
  - 向量空间模型　　第5章 向量模型及检索系统
  - 概率模型　　第8章 概率模型
  - 统计语言模型　　第9章 基于语言建模的检索模型
- 与内容无关的其他检索模型
  - 基于链接分析的模型　　第12章 Web搜索
  - 基于关联的模型　　第10章 文本分类
  - ……

【基于内容】布尔模型：定义

- 文档表示
  - 一个文档被表示为关键词的集合
- 查询表示
  - 查询式(Queries)被表示为关键词的布尔组合，用“与、或、非”连接起来（主析取范式 DNF ）
- 相关度计算
  - 一个文档当且仅当它能够满足布尔查询式时，才将其检索出来
  - 检索策略是二值匹配



$$q = k_a \wedge (k_b \vee \neg k_c)$$

$$q_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$$

【基于内容】布尔模型：优缺点

- 优点
  - 由于查询简单，因此容易理解
  - 通过使用复杂的布尔表达式，可方便地控制查询结果
  - 相当有效的实现方法
  - 经过某种训练的用户可以容易地写出布尔查询式
  - 布尔模型可以通过扩展来包含排序的功能
- 缺点
  - 弱。不支持部分匹配，完全匹配会导致结果太多或太少
  - 非常刚性：“与”意味着全部；“或”意味着任何一个
  - 原则上讲，所有被匹配的文档都将被返回
  - 不考虑索引词的权重，所有文档都以相同的方式和查询相匹配
  - 很难进行自动的相关反馈
  - 如果一篇文档被用户确认为相关或者不相关，怎样相应地修改查询式呢？

【基于内容】向量空间模型：定义

- 文档表示
  - 一个文档被表示为关键词构成的向量。
  - 一个文档D就可以表示为$D(t_1,t_2,\ldots,t_n)$，其中n就代表了关键字的数量。
  - 特征项权重W_k（Term Weight）：指特征项t_n能够代表文档D能力的大小，体现了特征项在文档中的重要程度。
- 查询表示
  - 一个文档被表示为关键词构成的向量$Q(t_1,t_2,\ldots,t_n)$。
- 相关度计算
  - 向量间的距离： $D(t_1,t_2,\ldots,t_n)$， $Q(t_1,t_2,\ldots,t_n)$

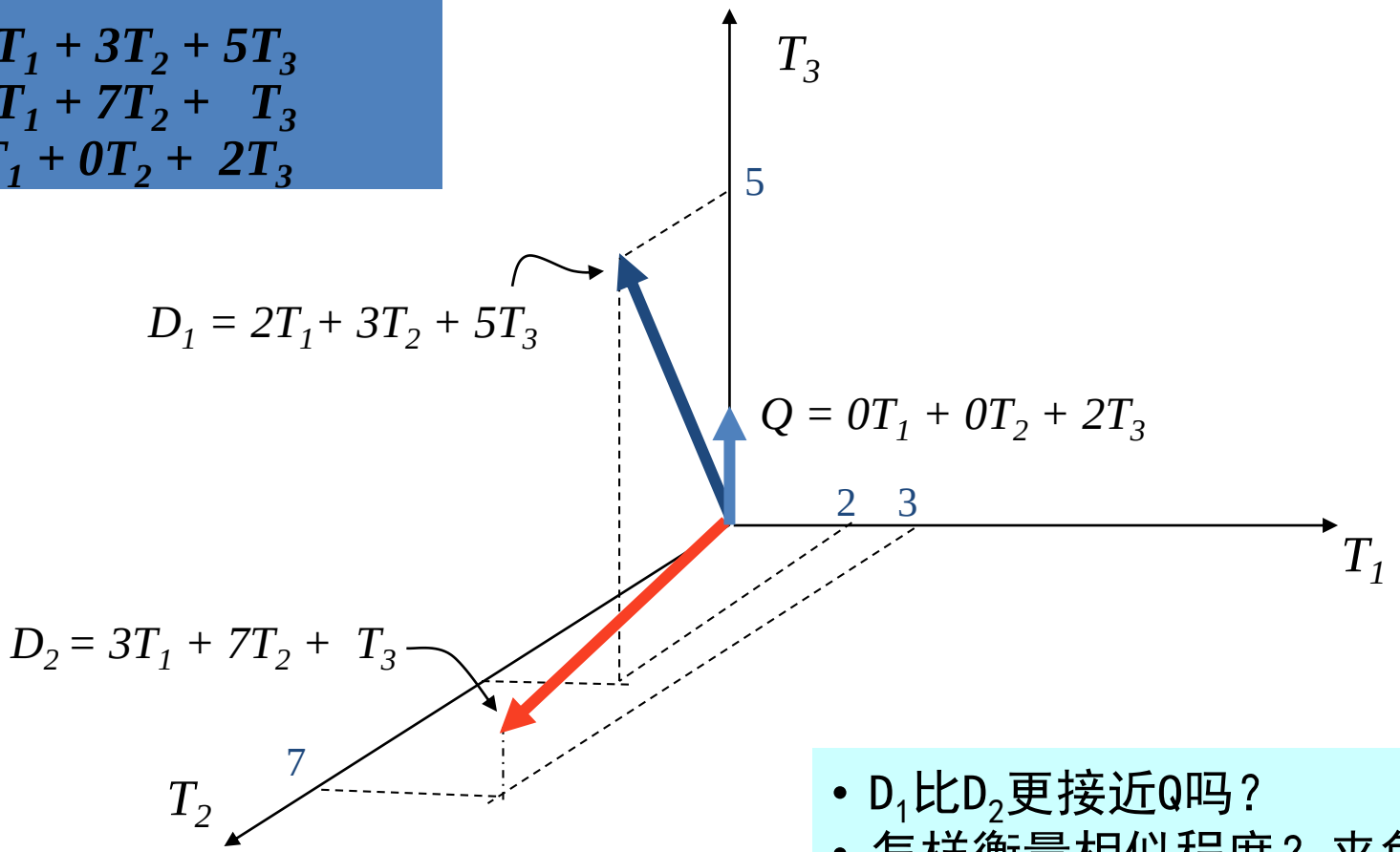
【基于内容】向量空间模型：距离图示

举例:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$

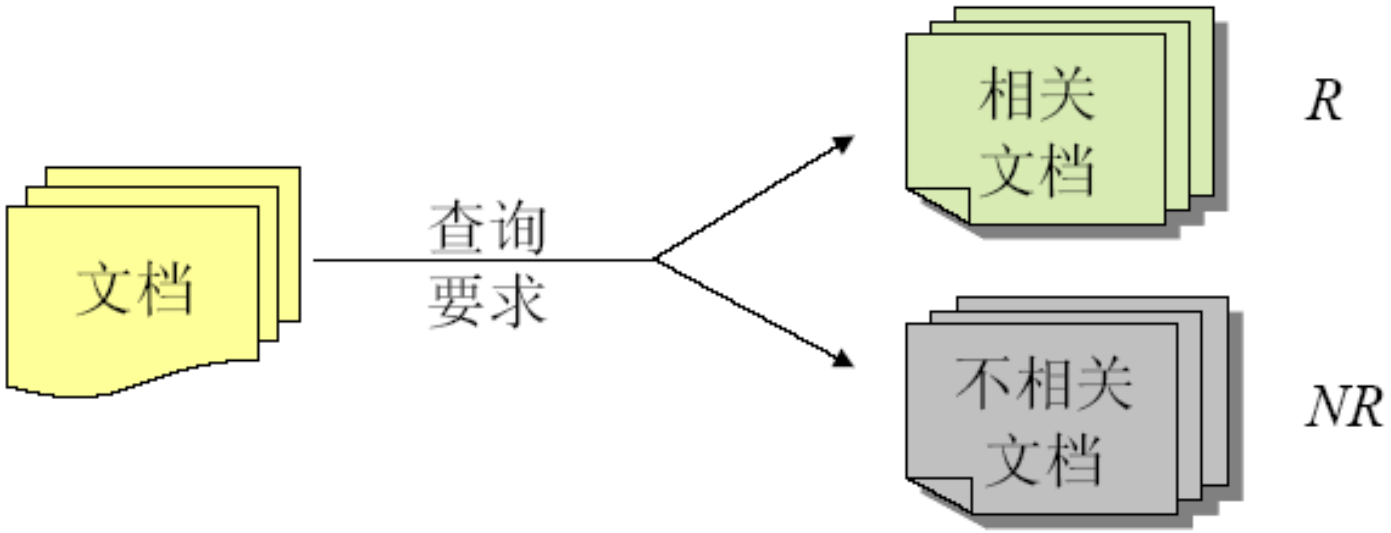


- D_1比D_2更接近Q吗？
- 怎样衡量相似程度？夹角还是投影

【基于内容】向量空间模型：特点

- 特点
  - 基于关键词(一个文本由一个关键词列表组成)
  - 根据关键词的出现频率计算相似度
    - 例如：文档的统计特性
  - 用户规定一个词项(term)集合，可以给每个词项附加权重
    - 未加权的词项: Q = < database; text; information >
    - 加权的词项: Q = <database 0.5; text 0.8; information 0.2 >
    - 查询式中没有布尔条件
  - 根据相似度对输出结果进行排序
  - 支持自动的相关反馈
    - 有用的词项被添加到原始的查询式中
    - 例如：Q = <database; text; information; document >
- 三个关键问题
  - 怎样确定文档中哪些词是重要的词？（索引项）
  - 怎样确定一个词在某个文档中或在整个文档集中的重要程度？（权重）
  - 怎样确定一个文档和一个查询式之间的相似度？（相关度）

【基于内容】概率模型



检索问题即求条件概率问题

If Prob($R|d_i, q$) > Prob($NR|d_i, q$) then d_i是检索结果，否则不是检索结果

【基于内容】统计语言模型（Probabilistic Language Models）

- **“语言”就是其字母表上的某种概率分布,该分布反映了任何一个字母序列成为该语言的一个句子(或其他任何的语言单元) 的可能性，称这个概率分布为语言模型。**
- **给定的一个语言，对于一个语言“句子”（符号串），可以估计其出现的概率。**

- **P(南京市长)P(江大桥|南京市长) <<**
  P(南京市)P(长江大桥|南京市)

- **P(中国科技大学) >> P(中国大学科技)**

【内容无关】链接分析模型

- 对于超文本（例如WWW上的网页），超链结构是个非常丰富和重要的资源，如果能够充分利用的话，可以极大地提高检索结果的质量。

- Sergey Brin 和Larry Page 在1998 年提出了 PageRank 算法
- J.Kleinberg 于1998年提出了HITS 算法
- 其它一些学者也相继提出了另外的链接分析算如SALSA，PHITS，Bayesian等算法。

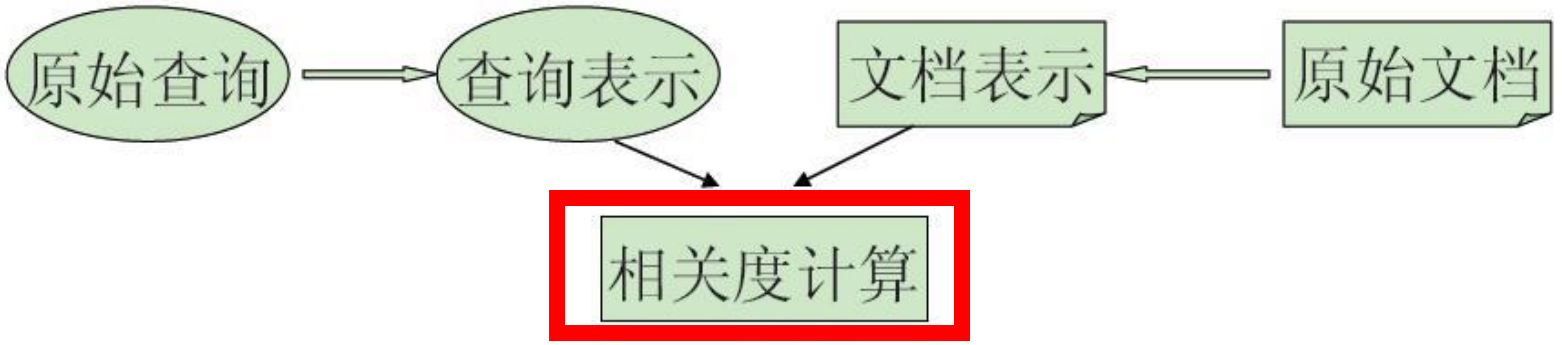
【内容无关】基于关联的模型

- **基本思想：同一个类型下的两个对象,如果经常连接到相同的其他对象，那么这两个对象的相似性应该很高。**

- **示例（ Simrank算法）**
  - **1.**利用文章的相互之间的引用关系计算文本的相似度。
    - **----两个文档的引文相同，那么这两个文档的相似性很高。**
  - **2.**利用文章的一些外部信息(关联)计算文本的相似度。
    - **---文档外部信息（作者，发表会议）**
    - **---两个文档有共同的作者，发表到共同的会议上，那么这两个文档的相似度很高。**

小结2-1：信息检索模型的使命

- 信息检索模型是指如何对查询和文档进行表示，然后对它们进行相似度计算的框架和方法
  - 本质上是对相关度建模
- 信息检索模型是IR中的核心内容之一



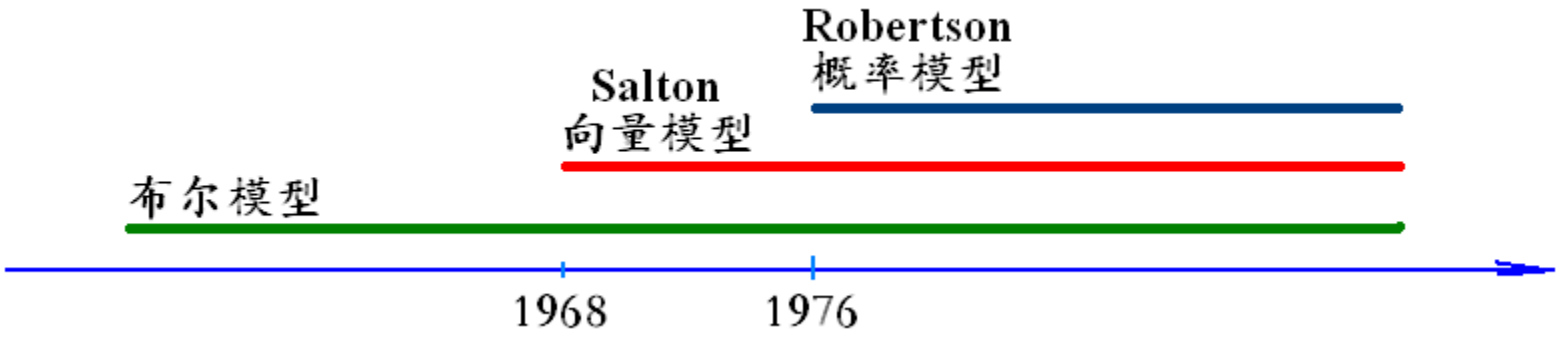
小结2-2：信息检索模型解决的问题

- 核心问题：相关度的计算
- 查询的表示/文档的表示
  - 关键字的集合？
  - 关键字构成的向量？
  - 关键字的重要程度？
  - 文档可视为关键字的集合？
  - 文档中单词出现的概率？

- 术语集合（representative terms）
- 标引词（index term）
- 标引项的权重（Weight）

小结2-3：信息检索模型之经典模型

- 集合论模型 (Set Theoretic models)
  - 布尔模型（ Boolean Model, BM）、模糊集合模型、扩展布尔模型
- 代数模型 (Algebraic models)
  - 向量空间模型（Vector Space Model, VSM）、广义向量空间模型、潜在语义标引模型、神经网络模型
- 概率模型 (Probabilistic models)
  - 经典概率论模型（PM）、推理网络模型、置信网络模型



第2章 布尔检索及倒排索引

- 2.1 信息检索模型概述
- 2.2 一个简单的搜索示例
- 2.3 倒排索引
- 2.4 布尔检索模型
- 2.5 布尔检索模型的优化与扩展

一个信息检索的例子

- 很多人都有《莎士比亚全集》这本大部头的书。我想知道：哪些剧本包含**Brutus**和**Caesar**但是不包含**Calpurnia**？
- 一种方式是采用**Unix**下的**grep**程序，先找出所有包含**Brutus**和**Caesar**的剧本，然后在将包含**Calpurnia**的剧本排除
- 但是很多情况下，采用上述线性扫描的方式是远远不够的。为什么？

一个信息检索的例子

- 上述方式不合适的原因：
  1. 对于大规模文档的搜索太慢
  2. 有时我们需要更灵活的匹配方式。比如，要求查找countrymen附近的Romans。
  3. 我们需要对结果进行排序。用户希望能在多个能满足自己要求的文档中得到最佳答案。

词项文档索引

词项—文档关联矩阵（incidence matrix）

- 解决方法是采用非线性的扫描方式，一种方法就是事先给文档建立索引（index）

	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

如果文档（这里就是剧本）包含某个词，则对应的项为1，否则为0

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

处理查询的过程

- 分别取出**Brutus\Caesar**以及**Calpurnia**对应的行向量，并对**Calpurnia**对应的向量求反。
- 进行按位与操作

  110100 *AND* 110111 *AND* 101111 = 100100.
- 结果向量中第一个和第四个元素为**1**，表明查询结果对应的剧本是 **Antony and Cleopatra** 和 **Hamlet**

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

查询的返回结果

- **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' t**
Capitol; *Brutus* killed me.

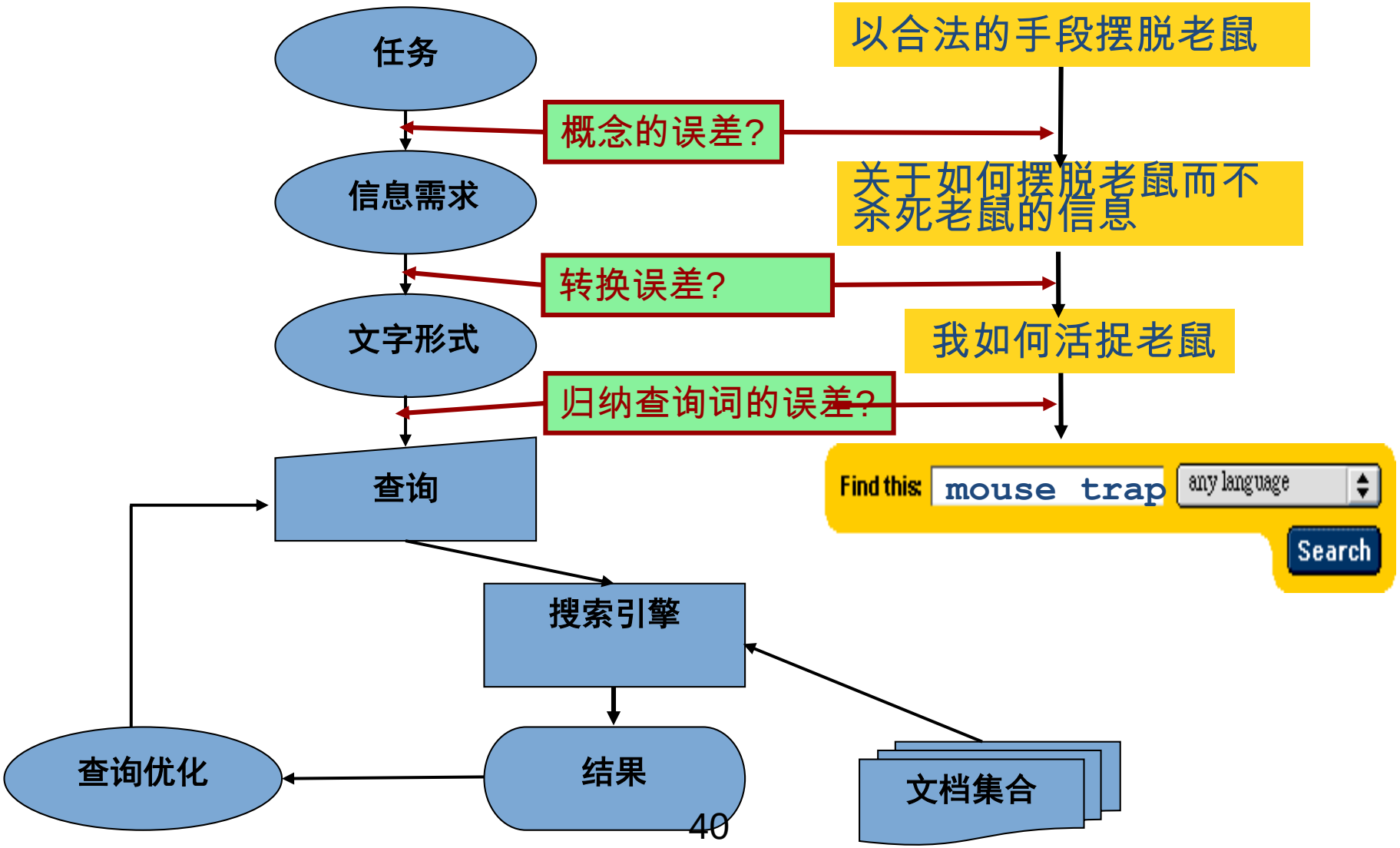
信息检索的基本假设

- 集合：固定数量的文档
- 目标：找到与用户信息需求相关的含有信息量的文档，帮助用户完成一个任务

ad hoc检索（ad hoc retrieval）任务（一种常见的信息检索任务）的系统。在这个任务中，任一用户的信息需求通过一次性的、由用户提交的查询传递给系统，系统从文档集中返回与之相关的文档。

信息检索中与ad hoc 检索任务相对的另一种任务称为**过滤（filtering）**。在ad hoc 任务中，信息需求动态变化，而文档集则相对静止。在过滤任务中，信息需求在一段时间内保持不变，而文档集则变化频繁。过滤任务的一个典型应用是信息订阅

典型的搜索模型


任务
以合法的手段摆脱老鼠
概念的误差?
信息需求
关于如何摆脱老鼠而不杀死老鼠的信息
转换误差?
文字形式
我如何活捉老鼠
归纳查询词的误差?
查询
Find this:
mouse trap
any language
Search
搜索引擎
查询优化
结果
文档集合

返回文档的好坏

- 查准率：返回的能满足用户信息需求的文档占总的返回文档的百分比
- 召回率：返回的能满足用户信息需求的文档占总的能满足用户信息需求的文档的百分比
- 信息检索的评价将在后续课程中详细介绍

更大的数据集

- 假设有一个更大的数据集：共有N=100万个文档，每个文档包含大概1000个词汇
- 假设一个词汇需要6个字节（将空格和标点计算在内），则这些文档共有6G
- 假设其中共有500k个唯一的不重复的词汇

- 在这种情况下，前面所使用的词项文档矩阵会如何？

无法构建矩阵

- 矩阵中有 500K * 1M = 0.5T 个 0和1
- 但是其中只有不到10亿个1，其他都是0.这个矩阵十分稀疏。
- 更好的索引表示方法是？
  - 只记录1的位置

	T_1	T_2	….	T_t
D_1	0	0	…	1
D_2	0	0	…	0
:	:	:		:
:	:	:		:
D_n	0	1	…	0

小结：一个简单的搜索示例

- 线性扫描的搜索：**grep**
- 非线性的扫描方式：**index**
- 处理查询
  - 构造矩阵 → **110100 AND 110111 AND 101111 = 100100**
- 典型的搜索模型
  - 任务→ 信息需求→ 文字形式→ 查询→ 查询优化→ 结果
  - 返回文档的好坏：查准率、召回率
- 简单模型存在的问题：大的数据集无法构建矩阵

第2章 布尔检索及倒排索引

- 2.1 信息检索模型概述
- 2.2 一个简单的搜索示例
- 2.3 倒排索引
- 2.4 布尔检索模型
- 2.5 布尔检索模型的优化与扩展

RDBMS中的索引

RDBMS: Relational Database Management System

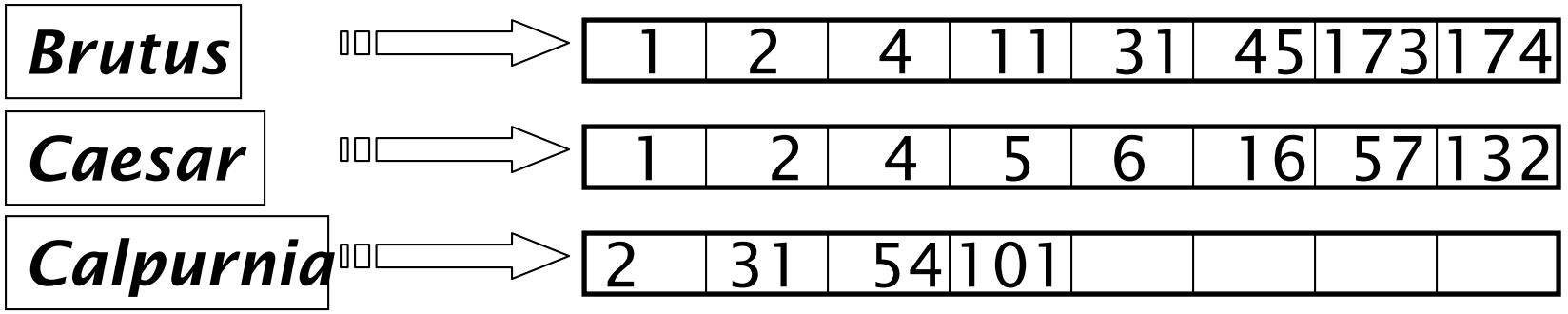
- 索引是一个单独的、物理的数据库结构，它是某个表中一列或若干列值的集合，和相应的指向表中物理标识这些值的数据页的逻辑指针清单。
- 表的存储由两部分组成，一部分用来存放数据页面，另一部分存放索引页面。通常，索引页面相对于数据页面来说小得多。
- 数据检索花费的大部分开销是磁盘读写，没有索引就需要从磁盘上读表的每一个数据页，如果有索引，则只需查找索引页面就可以了。所以建立合理的索引，就能加速数据的检索过程。

RDBMS中的数据类型

- 数值
  - **TINYINT：1字节，非常小的正整数，带符号：-128~127，不带符号：0~255**
  - **SMALLINT：2字节，小整数，带符号：-32768~32767，不带符号：0~65535**
  - **MEDIUMINT：3字节，中等大小的整数，带符号：-8388608~8388607，不带符号：0~16777215**
  - **INT：4字节，标准整数，带符号：-2147483648~2147483647，不带符号：0~4294967295**
  - **BIGINT：8字节，大整数，带符号：-9223372036854775808~9233372036854775807，不带符号：0~18446744073709551615**
  - **FLOAT：4字节，单精度浮点数，最小非零值：+-1.175494351E-38，最大非零值：+-3.402823466E+38**
  - **DOUBLE：8字节，双精度浮点数，最小非零值：+-2.2250738585072014E-308，最大非零值：+-1.7976931348623157E+308**
- **字符串：CHAR[255]** ←用来做索引？
- **文本：TEXT，用指针存储**

倒排索引（Inverted index）

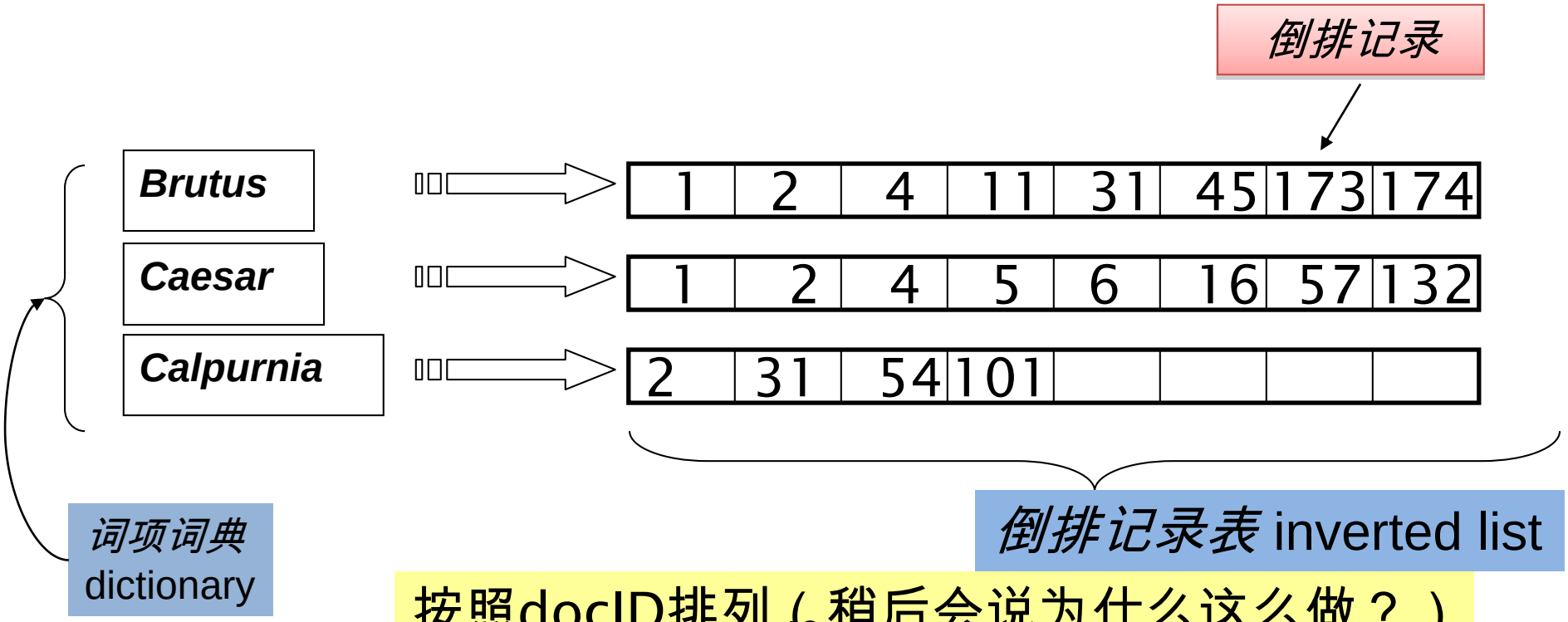
- 对于每一个词项，存储所有包含这个词项的文档的一个列表。一个文档用一个序列号**docID**来表示。
- 我们能用一个固定长度的数组来存储它吗？



如果词项*Caesar* 被添加到14号文档中，索引如何变化?

倒排索引

- 应当使用可变长度的记录列表
  - 在硬盘上，一串连续的记录是正常的，也是最好的。
  - 在内存里，可以使用链表，或者可变长度的数组。



按照docID排列（稍后会说为什么这么做？）

倒排索引的建立

需要建立索引的文档

Friends, Romans, countrymen.

词条化模块

词条流

Friends | Romans | Countrymen

后面会有更详细的说明

语言学模块

修改过的词条

friend | roman | countryman

索引模块

倒排索引

friend	2 → 4 →
roman	1 → 2 →
countryman	13 → 16

建立索引的步骤：词条序列

- （修改过的词条，文档ID）对应的序列

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

建立索引的步骤：排序

先按照词条排序，
再按照docID排序

建立索引的核心步骤

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

建立索引的步骤：词典和倒排表

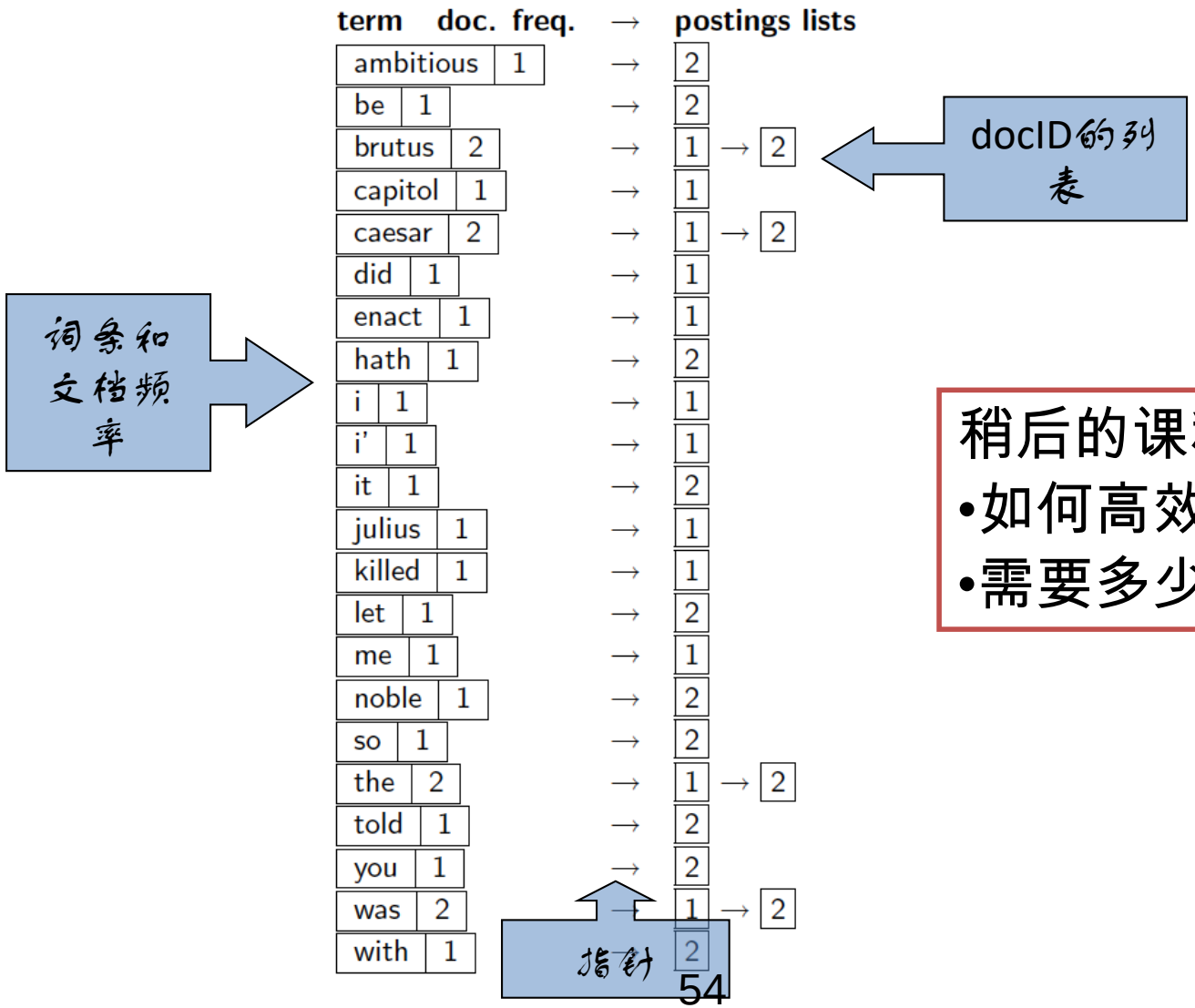
- 同一篇文档中多次出现的词被合并
- 分割成词典和倒排表
- 词汇的文档频率也被记录

后面会讨论为什么记录文档频率

Term	docID
ambitiou	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

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



稍后的课程会讨论:

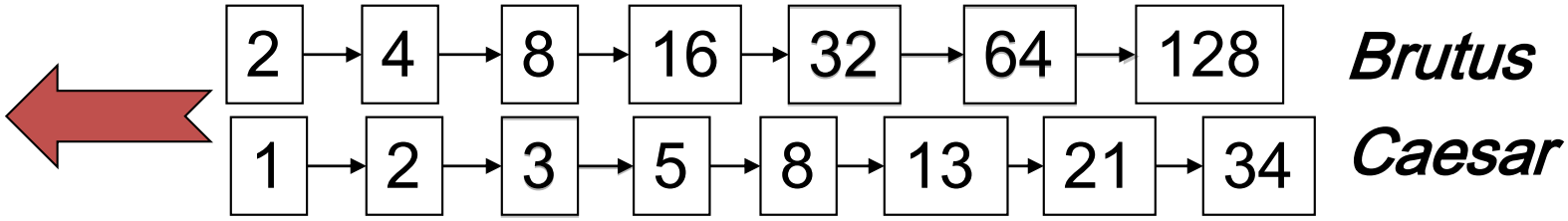
- 如何高效地建立索引?
- 需要多少存储空间？

查询的处理：AND

- 考虑处理这样的查询：

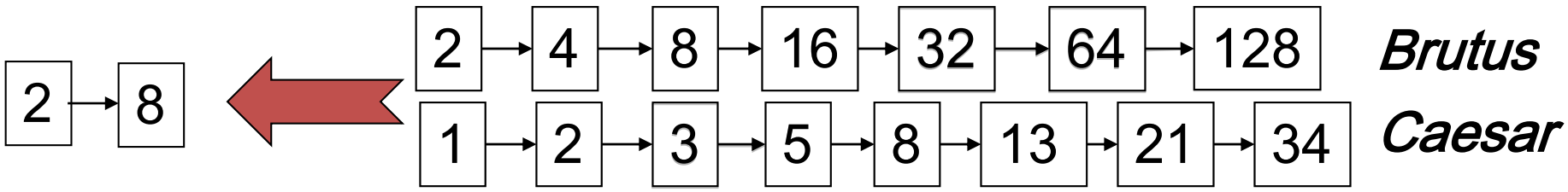
  Brutus ***AND*** *Caesar*

  - 在字典中找到Brutus，得到它的倒排记录表
  - 在字典中找到Caesar，得到它的倒排记录表
  - 合并两个倒排列表



倒排记录表的“合并”

- 同时扫描两个倒排记录表，所需时间和倒排记录的数量呈线性关系。



如果倒排记录表的长度分别为x和y, 则合并共需 O(*x*+*y*)次操作

注意：倒排记录已经按照docID排练好了

倒排记录表的“合并”算法（求交集）

```
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
```

小结：倒排索引

- **RDBMS中的索引（Index）**
- **倒排索引（Inverted index）**
  - **对于每一个词项，存储所有包含这个词项的文档的一个列表**
  - **词项 + 倒排记录**
- **倒排索引的建立过程**
  - **词条化模块 →语言学模块→索引模块**
- **查询的处理 ←→ 倒排记录表的“合并”**
  - **AND**

第2章 布尔检索及倒排索引

- 2.1 信息检索模型概述
- 2.2 一个简单的搜索示例
- 2.3 倒排索引
- 2.4 布尔检索模型
- 2.5 布尔检索模型的优化与扩展

布尔检索模型：定义

- 文档表示
  - 一个文档被表示为关键词的集合
- 查询表示
  - 查询式(Queries)被表示为关键词的布尔组合，用“与、或、非”连接起来（主析取范式DNF ）
- 相关度计算
  - 一个文档当且仅当它能够满足布尔查询式时，才将其检索出来
  - 检索策略是二值匹配

布尔检索模型：形式化表示

- 定义：用$\mathbf{q_{dnf}}$表示查询q的析取范式，$\mathbf{q_{cc}}$表示$\mathbf{q_{dnf}}$的任意合取分量。文献$\mathbf{d_j}$ 与查询q的相似度为

$$sim(d_j,q)=\begin{cases}1 \text{ if } \exists q_{cc} \mid (q_{cc}\in q_{dnf})\wedge(\forall k_i, g_i(d_j)=g_i(q_{cc})) \\ 0 \text{ otherwise}\end{cases}$$

如果$sim(d_j,q)=1$，则表示文献dj与q相关，否则为不相关。

$sim(d_j, q)$ 为该模型的匹配函数。

布尔检索模型：布尔代数

- 布尔变量
  - 只有“真”、“假”取值的变量
  - 计算机中常常用1表示“真”true，0表示“假”false
- 布尔操作(关系)
  - 与(AND)：(A AND B) = true iff A=true and B=true
  - 或(OR)：(A OR B) = true iff A=true OR B=true
  - 非(NOT)：(NOT A) = true iff A=false
- 布尔表达式
  - 多个布尔变量之间通过布尔操作组成的表达式
    - 如：A AND (B OR C) AND NOT D
  - 蕴含：两个布尔表达式P、Q，如果A为true，那么Q为true，则称P蕴含Q，记为P→Q

布尔检索模型：形式化表示

- **任意查询都可转化为一个主析取范式DNF**
  - 例如：查询为$q=k_a \wedge (k_b \vee \neg k_c)$可表示为

$q=k_a \wedge (k_b \vee \neg k_c)=k_a k_b k_c \vee k_a k_b \neg k_c \vee k_a \neg k_b \neg k_c$

$\rightarrow$

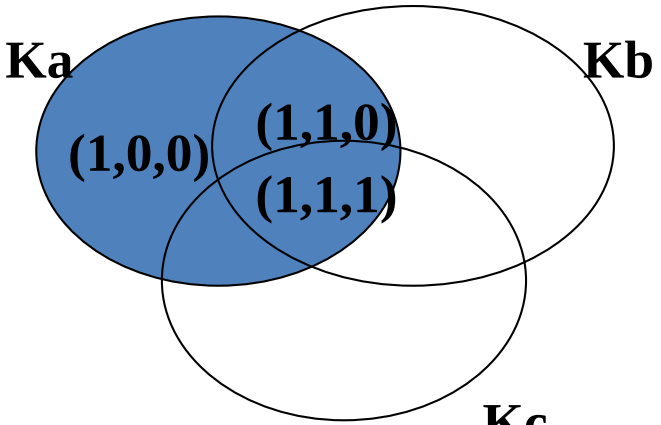
$q_{dnf}=(1,1,1) \vee (1,1,0) \vee (1,0,0)$

  - 即：每一个分量都是三元组的二值向量

$$(k_a, k_b, k_c)$$

  - 任一文本可以写成所有Term的交，如doc=a∧b∧c∧d∧e

因为doc→(蕴含)q，所以相似度为1



布尔检索：精确匹配

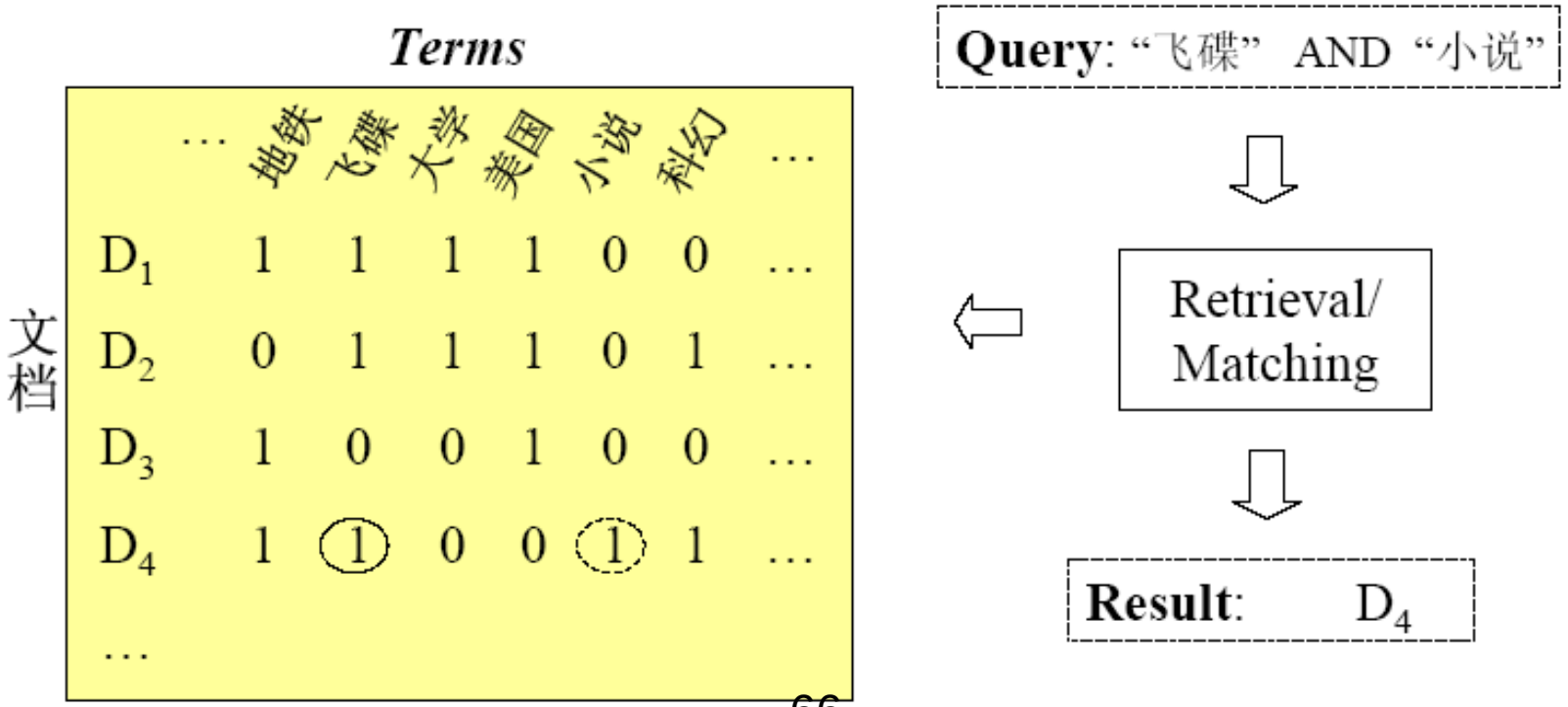
- 布尔模型可以用来处理布尔表达式形式的查询
  - 布尔查询使用AND,OR,和NOT来连接查询词汇
    - 将文档看作词汇的集合
    - 精确：匹配 或 不匹配
  - 布尔模型也许是IR系统中的最简单的模型
  - 是近30年来最主要的商业搜索工具
  - 当前使用的很多系统仍然是使用的布尔模型
    - 电子邮件，图书馆分类系统，mac osx的spotlight

布尔检索应用：关系数据库SQL

- SELECT * FROM Persons WHERE FirstName='Bush'
- SELECT * FROM Persons WHERE Year>1965
- SELECT * FROM Persons WHERE firstname='Thomas‘
  OR lastname='Carter'
- SELECT * FROM Persons WHERE
  (FirstName='Thomas‘
  OR FirstName='William') AND LastName='Carter'
- SELECT * FROM Persons
  WHERE LastName IN ('Adams','Carter')
- SELECT * FROM Persons WHERE LastName
  NOT BETWEEN 'Adams' AND 'Carter'
- SELECT * FROM Persons WHERE City LIKE 'N%'

示例：精确匹配的欠缺

- “飞碟”AND “小说”：只能检索出D4，无法显现D1,D2,D3的差异
- “飞碟”OR “小说”：可以检出D1,D2,D4，但无法显现它们的差异



示例：WestLaw http://www.westlaw.com/

Westlaw BETTER RESULTS FASTER.

- 最大的收费的法律搜索服务提供商
- 几十T的数据; 700,000多用户
- 大多数用户仍然使用布尔查询
- 查询的例子:
  - **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**
- 许多专业的用户通常喜欢使用布尔查询
  - 因为用户精确的知道自己会得到什么
- 但是这并不意味着能得到更好的结果

如何表示查询：grep示例（正则表达式）

- **grep -c "48" test.txt**
  - 统计所有以“48”字符开头的行有多少
- **grep "48[34]" test.txt**
  - 显示输出以字符“48”开头，第三个字符是“3”或是“4”的所有的行）
- **grep "^[^48]" test.txt**
  - 显示输出行首不是字符“48”的行）
- **grep "K…D" test.txt**
  - 显示输出第一个字符是“K”，第二、三、四是任意字符，第五个字符是“D”所在的行）
- **grep "[A-Z][9]D" test.txt**
  - 显示输出第一个字符的范围是“A-D”，第二个字符是“9”，第三个字符的是“D”的所有的行
- **grep "[35]..1998" test.txt**
  - 显示第一个字符是3或5，第二三个字符是任意，以1998结尾的所有行
- **grep "4\{2,\}" test.txt**
  - 模式出现几率查找：显示输出字符“4”至少重复出现两次的所有行

第2章 布尔检索及倒排索引

- 2.1 信息检索模型概述
- 2.2 一个简单的搜索示例
- 2.3 倒排索引
- 2.4 布尔检索模型
- 2.5 布尔检索模型的优化与扩展

布尔查询：更宽泛的合并算法

思考

- 修改合并的算法，处理一下的查询
  - *Brutus **AND NOT** Caesar*
  - *Brutus **OR NOT** Caesar*

  在这种情况之下，合并依然能够在O(x+y)的时间内完成吗？
  应该如何做？

- 对于任意的布尔查询组合怎么做？

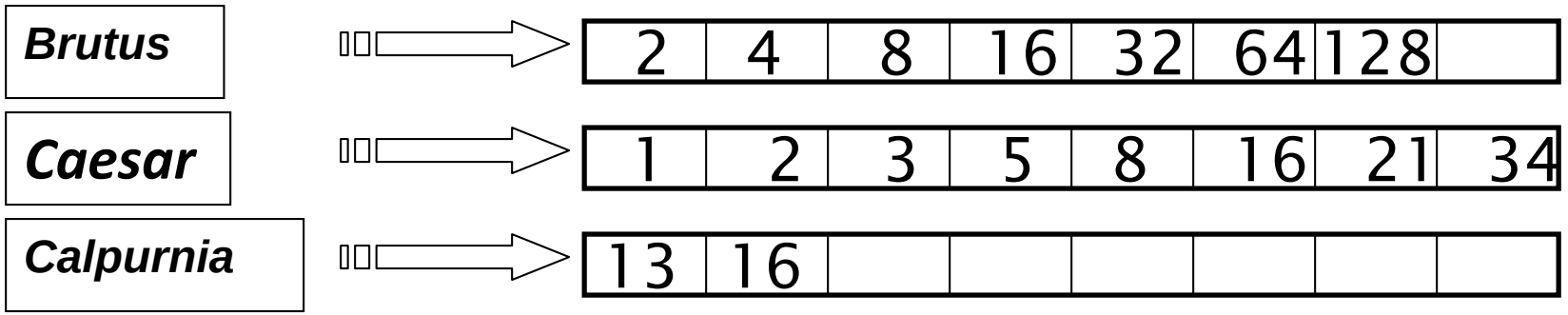
(Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

- 依然能够在“线性”时间内完成吗？
  - 和什么呈线性关系？

- 能否做得更好？

查询的优化

- 处理查询的最佳顺序是什么？
- 考虑一个使用AND连接n个词汇的查询
- 对于每一个词汇，都得到它的倒排记录表，然后进行“合并”操作

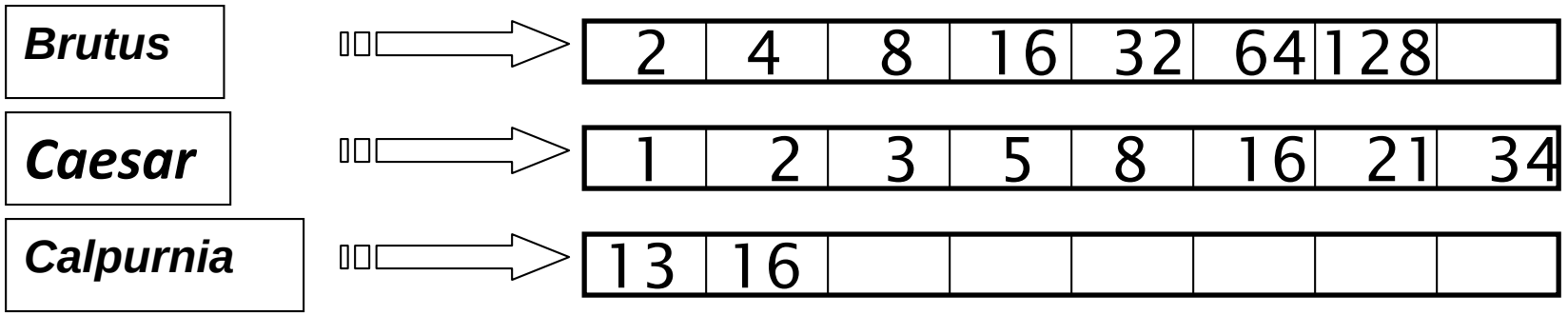


查询: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

查询优化的例子

- **按照**文档频率的顺序**进行处理：**
  - 先处理文档频率小的，再处理大的

这就是为什么我们前面提到要存储词条的文档频率



按照(***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caesar***的顺利处理查询

更一般的优化

- **e.g., *(madding OR crowd) AND (ignoble OR strife) AND (killed OR slain)***
- **获得所有词项的文档频率**
- **保守地估计出每个OR操作后的结果大小**
- **按照结果从小到大的顺序执行AND**

73

思考题（习题1-7）

- 请推荐如下查询的处理次序

***(tangerine** OR **trees)** AND*
***(marmalade** OR **skies)** AND*
(kaleidoscope** OR **eyes)

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

除了词查询外，信息检索还有什么问题？

- 词组？
  - 中国科学技术大学
- 词项邻近: 在“合肥”附近的“老母鸡”.
  - 对文档建索引的时候需要考虑词出现的位置
- 文档中的区域:
  - 查找*作者*是“**D.Manning**”，正文含有“**Information**”的文档

75

可信度的累加

- 查询词出现1次或0次
  - 2次 vs. 1次
  - 3次 vs. 2次， 等等
  - 一般来说出现次数越多越好
- 需要统计文档中的词出现的频率

76

布尔模型→引入模糊关系

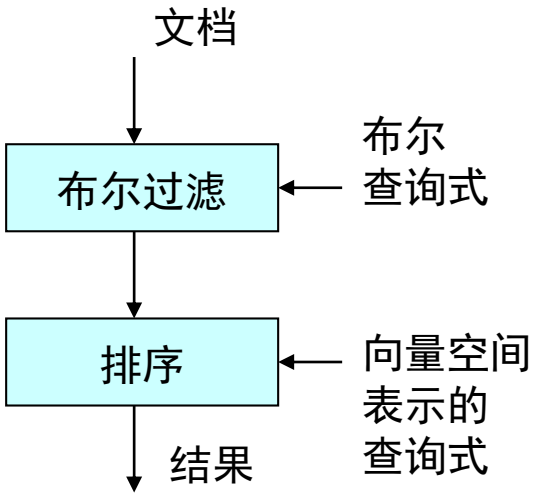
$R(A,B)$ B_i / A_i	40	50	60	70	80
140	1	0	0	0	0
150	0	1	0	0	0
160	0	0	1	0	0
170	0	0	0	1	0
180	0	0	0	0	1

$R(A,B)$ B_i / A_i	40	50	60	70	80
140	1	0.8	0.2	0.1	0
150	0.8	1	0.8	0.2	0.1
160	0	0	1	0.8	0.2
170	0	0	0.8	1	0.8
180	0	0.1	0.2	0.8	1

布尔模型和向量空间模型相结合

- 布尔模型可以和向量空间模型相结合，先做布尔过滤，然后进行排序：
  - 首先进行布尔查询
  - 将全部满足布尔查询的文档汇集成一个文档
  - 用向量空间法对布尔检索结果进行排序

- 如果忽略布尔关系的话，向量空间查询式和布尔查询式是相同的



扩展的布尔检索相似度计算示例

Query / Doc Sim()	“飞碟” AND “小说”	“飞碟” OR “小说”
D_1	0.293	0.707
D_2	0.293	0.707
D_3	0	0
D_4	1	1

从“一刀切”到“合理拉开差距”

$x, y = 1$ if a term exists in d_j

$x, y = 0$ otherwise

检索结果排序

- 布尔查询给出了包含和不包含关系
- 通常来说我们还希望对结果进行排序/聚类
  - 需要度量查询和文档的近似度
  - 需要确定返回给用户的文档是否是独一无二的，或者返回给用户的是一组包含了查询词各个方面的文档

80

非结构化数据 vs 结构化数据

- 结构化数据一般指表格中存储的数据

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

- 非结构化数据通常指自由格式的文本
  - 允许：包含关系操作符的关键词查询；更复杂的如概念搜索如：找所有关于“吸毒”的网页

半结构化搜索

- 实际上没有数据是非结构化的
  - 例如这一页PPT里面就有标题和项目编号两种特定的区域
  - 利用半结构信息的查询示例：
  - **Title contains data AND Bullets contain search**
- 查询：
  - ***Title* is about Object Oriented Programming AND *Author* something like stro*rup**
  - 其中*是通配符
  - 问题：
  - 怎么处理“about”？
  - 结果怎么排序？

小结：

- 优化
  - 文档频率
  - 临近词
  - 可信度的累加
- 扩展
  - 引入模糊关系（模糊集合论）
  - 布尔模型和向量空间模型相结合
  - 检索结果排序
- 非结构化数据 vs 结构化数据 半结构化

作业与实验

- **实验：倒排索引的建立和布尔查询**

- **习题1-8: 对于查询**

  friends AND romans AND (NOT countrymen)

  如何使用*countrymen*的文档频率来估计最佳的查询处理次序?
- **习题1-5: 将倒排记录表合并算法推广到任意布尔查询表达式，考虑其时间复杂度？对下面的查询**

  (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

  我们能保证在线性时间内完成合并吗？能否对此加以改进?

谢谢大家！