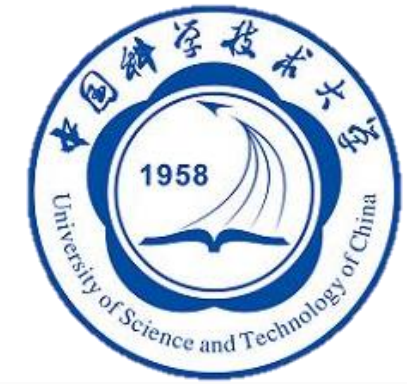


数据科学导引

第十三次课

信息检索简介

何向南 2020.6.5

声明：大部分课件内容来自徐童老师《Web信息处理与应用》的课件

- **什么是信息检索?**

- 历史：“Information Retrieval”产生于 Calvin Mooers 1948年在MIT的硕士论文
- 定义：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
  - 信息检索旨在帮助人们从海量的数据中快速的找到有用的信息
- 作为一门学科，是研究信息的获取 (acquisition)、表示(representation)、存储(storage)、组织(organization)和访问(access) 的一门学问。
- 以计算机为手段，处理信息对象和其他学科也融合： 语言学、认知科学

http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **信息检索 vs. 搜索引擎**

- 检索来自英文单词Retrieval，有些人把它翻译成获取。其本义是“获得与输入要求相匹配的输出”。 和我们平时所理解的搜索意义上的检索不一样。
- IR不仅仅是搜索，IR系统也不仅仅是搜索引擎
  - 例1：曾哥是狮子座的吗？->问答系统(Question Answering,QA)
  - 例2：返回Ipad的各种型号、配置、价格等 ->信息抽取 Information Extraction, IE)
  - 例3：使用Google Reader订阅新闻，并获取推荐->信息过 滤(Information Filtering)、信息推荐(Information Recommending)

:

http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **为什么要讲信息检索?**

- 时代背景：信息过载与大数据时代
  - 需要从海量的数据中快速找到有用信息的技术
- 市场需求：公司需要IR技术
  - 搜索引擎和推荐系统改变了人们获取信息的方式，并成为了各大科技公司的一项基本服务，Google、Microsoft、百度，还有如腾讯、头条、阿里等都加入到检索技术的竞争当中
- 市场需求：人才的竞争
  - 搜索与推荐相关人才紧缺，成为各大科技巨头争相抢夺的重要资源

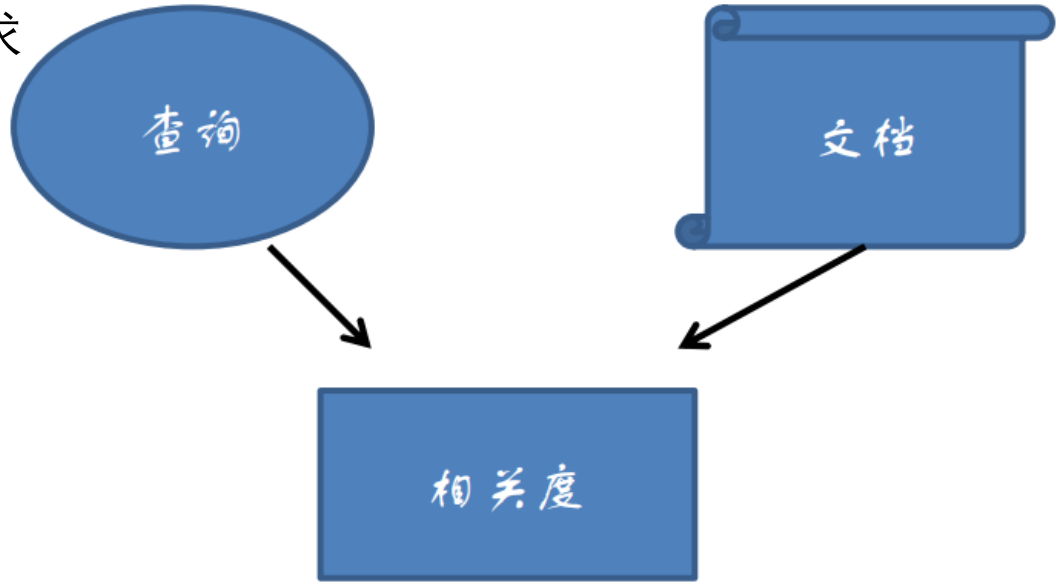
http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **信息检索的基本概念**

信息需求（Information need）： the information that the user is desired to know

查询（Query）：用户传递给计算机的信息，以表示其信息需求

文档（document）：要检索的对象（文本、网页、图片、视频等）



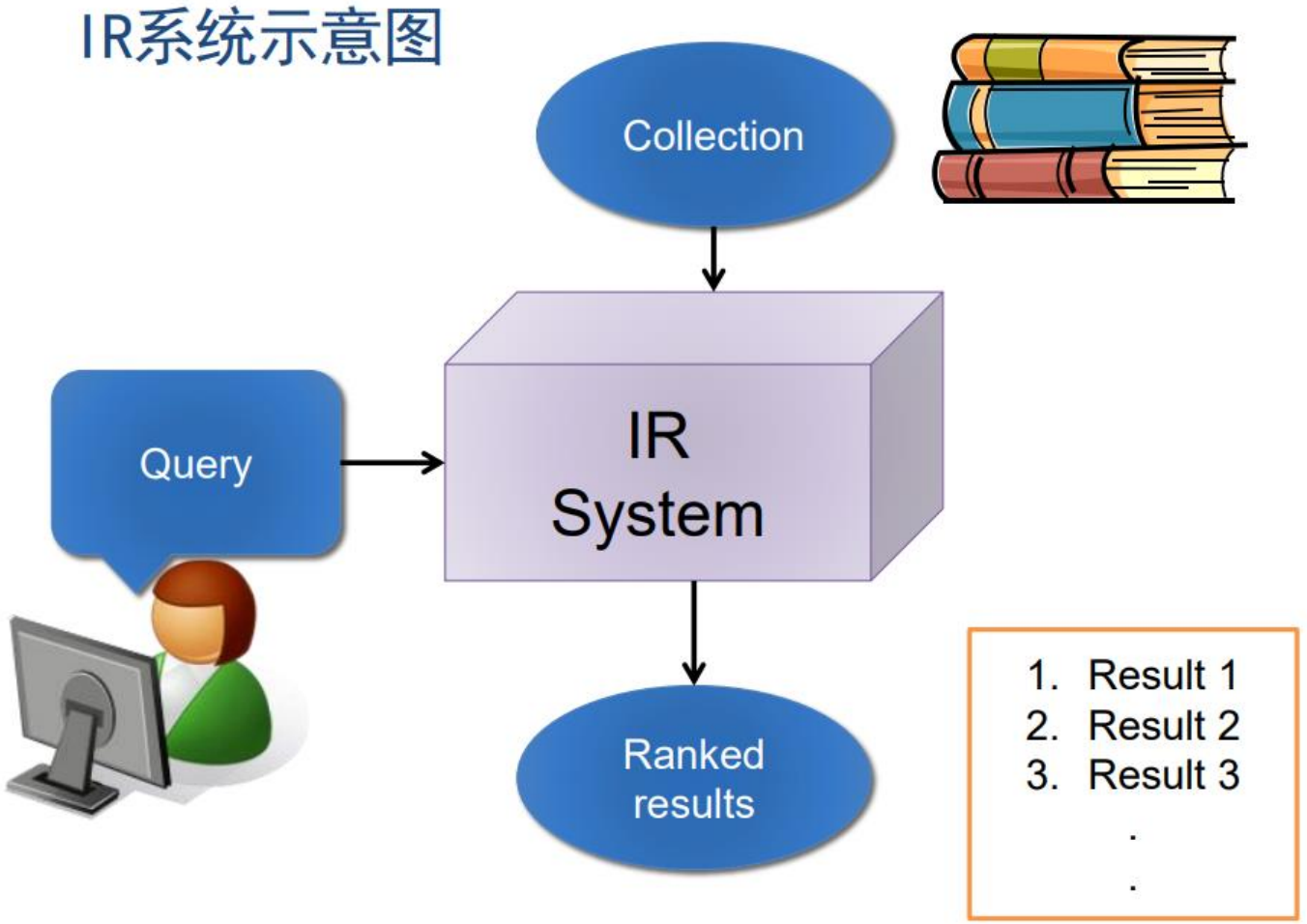
文档集（corpus）：所有待检索的文档构成的集合

相关度（relevance）： 客观角度 vs. 主观角度

传统信息检索主要考虑客观相关性（e.g., 主题相关性）， 但许多实际系统更重视用户的认知因素

http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **信息检索系统示意**



http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **信息检索 vs. 数据库检索**

	信息检索	数据库检索
检索对象	无结构、半结构数据 如网页、图片……	结构化数据 如：员工数据库
检索方式	通常是近似检索 如：每个结果有相关度得分	通常是精确检索 如：姓名==“李明”
检索语言	主要是自然语言 如：查与超女相关的新闻	SQL结构化语言

近年来，随着XML的出现，两种检索已经逐渐融合，边界越来越不明显。

http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

- **本次教学重点：文本检索系统**

- 文本预处理：对查询和文档进行预处理的操作
  - E.g., 分词、词干还原（stemming）、停用词消除、查询扩展、查询重构
- 文本索引（Indexing）：基于索引项（term）对预处理后的文本进行表示，方便后续的搜索匹配
- 匹配（Matching）：从文本中查找包含查询中索引项的文本
- 排序（Ranking）：对匹配出的文本按照某种方式计算相关度，最终展示给用户

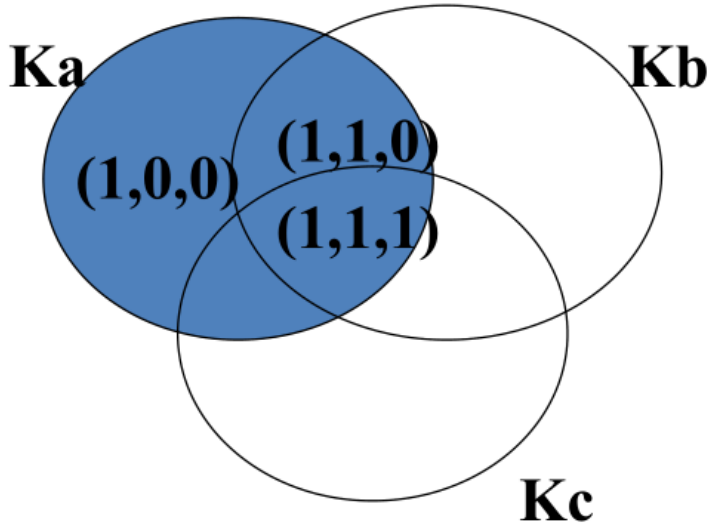
http://home.ustc.edu.cn/~zhufengx/ir/pdf/IR-1_2015.pdf

本节目录

- **布尔检索与倒排索引**
- 索引存储
- 搜索匹配与排序

- **布尔检索的概念**

- 在布尔检索中，文档被表示为**关键词的集合**。
- 所有的查询式都被表示为关键词的**布尔组合**。
  - 采用“与、或、非”关系加以连接
- 相关度计算
  - 一个文档当且仅当它能够满足布尔查询式时，才会将其检索出来。
  - 检索策略是**二值匹配**。



- **布尔检索的优缺点**

优点

- 查询简单，易于理解
- 使用布尔表达式，可以方便地控制查询结果
- 可通过扩展来包含更多功能

缺点

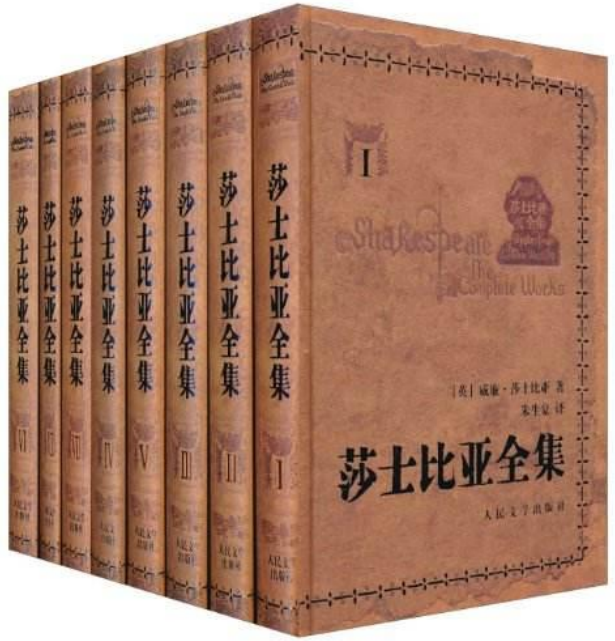
- 功能较弱，不支持部分匹配
- 所有匹配文档均返回，不考虑权重和排序
- 很难进行自动的相关性反馈

- **布尔检索的应用场景**

- 利用AND，OR或NOT等操作符，将词项连接起来
  - 操作符可以连续使用，如“信息 AND (检索 OR 挖掘) AND 教材”

- **一个布尔检索的实例**

- 《莎士比亚全集》中，哪些剧本包含Brutus和Caesar，但不包含Calpurnia?
  - 采用如下布尔表达式：Brutus AND Caesar AND NOT Calpurnia
- 一个解决的笨办法：从头到尾扫描所有剧本
  - 最大缺陷：速度超慢（尤其对于大型文档）
  - 处理NOT操作并不容易，需要遍历全文
  - 不支持检索结果的排序



- **另一种选择：关联矩阵**

- 通过采用非线性的扫描方式来解决，一种方法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

- **关联矩阵的查询实例**

- 关联矩阵的每一列都是0/1向量，每个0/1对应一个词项
  - 1代表包含这个词，0代表不包含
- 将给定的查询条件Brutus AND Caesar AND NOT Calpurnia转化为行向量运算
  - 取出三个行向量，对Calpurnia的行向量求补，最后按位进行与操作

- **110100 AND 110111 AND 101111 = 100100**

	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

- **关联矩阵的缺陷**

- 如果文档集很大……？
  - 假如有M篇文档，词汇表大小为N，那么矩阵规模为 M x N.
  - 在现实条件下，M和N的数量级都将达到百万甚至上亿的水平。
- 更为重要的是，矩阵中 1 的数量会微乎其微。
  - 高度稀疏的矩阵（可能小于1/500）
  - **只记录1的位置更为合理！**

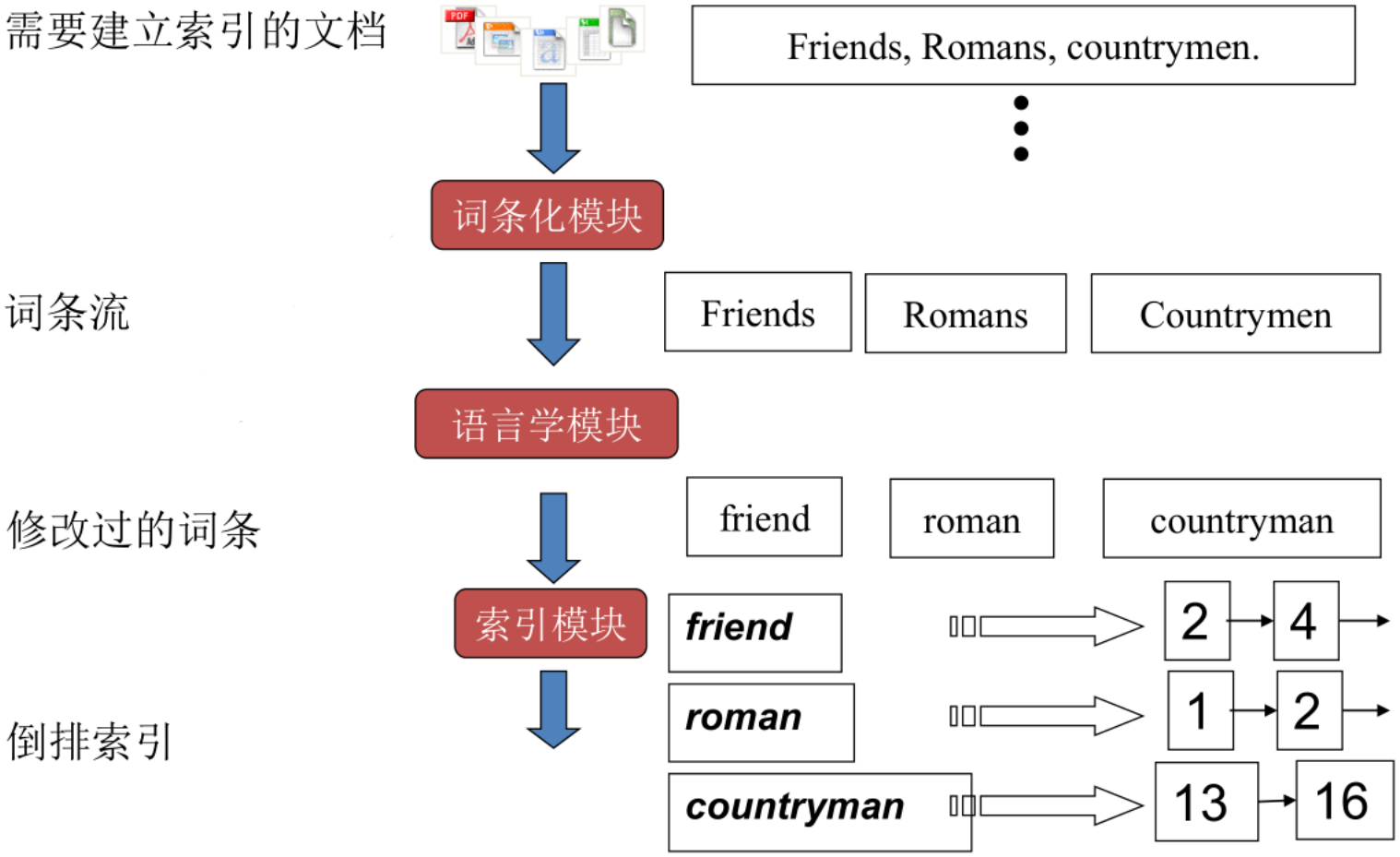
	T_1	T_2	….	T_t
D_1	0	0	…	1
D_2	0	0	…	0
:	:	:		:
:	:	:		:
D_n	0	1	…	0

- **倒排索引的概念与意义**

- 信息检索中非常流行的、基于词项的基础文本索引
- 主要包括以下两部分结构：
  - 词汇表（词典，Dictionary）：词项的集合
  - 倒排表（Posting List）：文档ID列表，列举词项在哪些文档中出现

Vocabulary		Postings List
term1	→	Document17, Document 45123
		.
		.
termN	→	Document991, Document123001

- **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
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

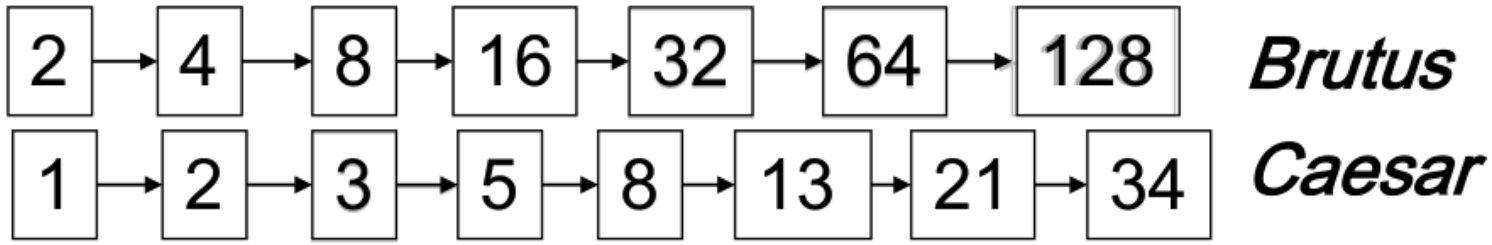
- **倒排索引的实例**

- 三篇简单文档的倒排表与检索实例
  - Yes, we got no bananas.
  - Johnny Appleseed planted apple seeds.
  - We like to eat, eat, eat apples and bananas.

Vocabulary		Postings List
yes	→	D1
we	→	D1, D3
got	→	D1
no	→	D1
bananas	→	D1, D3
Johnny	→	D2
Appleseed	→	D2
planted	→	D2
apple	→	D2, D3
seeds	→	D2
like	→	D3
to	→	D3
eat	→	D3
and	→	D3

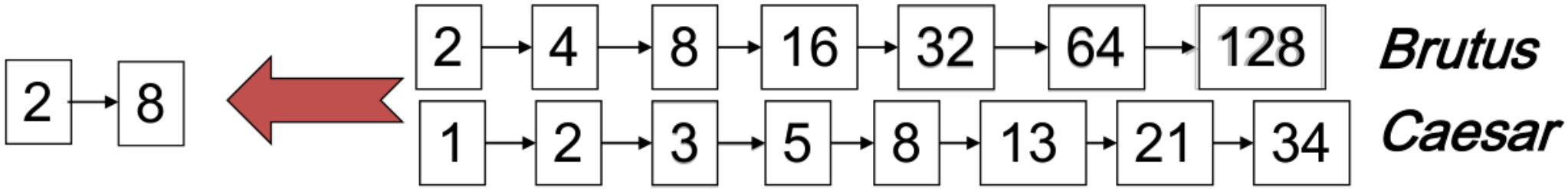
- **倒排索引的实例：基于倒排表的索引**

- 以莎翁的剧本检索为例，假如我们要查询Brutus AND Caesar
- 不妨假设两个词项的倒排表遵循以下形式：
  - Brutus：2，4，8，16，32，64，128……
  - Caesar：1，2，3，5，8，13，21，34……



- **倒排索引的实例：基于倒排表的索引**

- 基于倒排表的查询，本质上是倒排记录表的“合并”过程
- 同时扫描两个倒排表，所需时间与倒排记录的数量呈线性关系
  - 如果两个倒排表的长度分别为x和y，则合并供需O(x+y)次操作

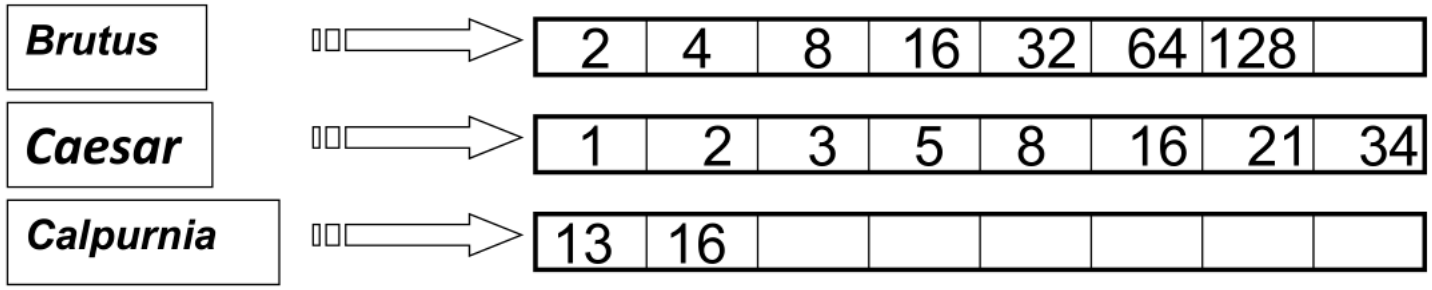


- **倒排索引的优化问题**

- 对于含有NOT操作的布尔查询，是否仍然能在O(x+y)的时间内完成?
  - 例如，Brutus AND NOT Caesar，如何处理?
- 进而，对于任意组合（例如，包含括号）的布尔查询，如何处理?
  - 例如，(Brutus OR Caesar) AND NOT (Antony OR Cleopatra)
  - 是否还能够在线性时间内完成?
- **核心问题：如何提升倒排索引的效率?**

- **倒排索引的优化问题**

- 处理查询的最佳顺序是什么?
  - 对于使用AND连接的查询，其本质是倒排表的合并操作



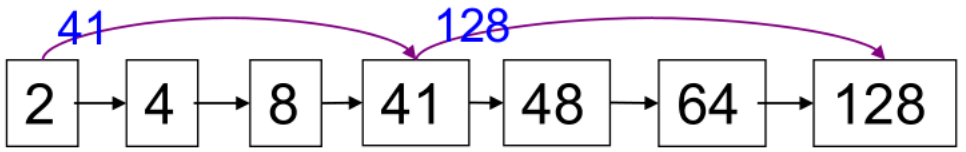
  - 按照查询词项的频率进行处理
    - 先处理频率小的，再处理大的
    - 在上例中，我们采用(Calpurnia AND Brutus) AND Caesar的顺序处理

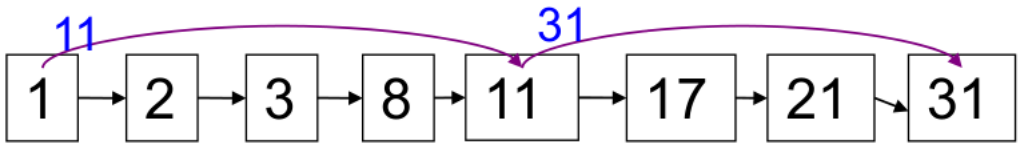
- **倒排索引的优化问题**

- 更一般的优化问题：任意组合的布尔查询
  - 例如：(Brutus OR Caesar) AND (Antony OR Cleopatra)
  - 同样，按照文档频率的顺序进行处理
    - 首先，获得所有词项的文档频率
    - 其次，保守地估计出每个OR操作后的结果大小
      - 考虑x+y的最坏情况
    - 最后，按照结果从小到大的顺序执行AND

- **倒排表合并中的优化问题**

- 更进一步：倒排表的合并需要O(x+y)次，能否做得更好?

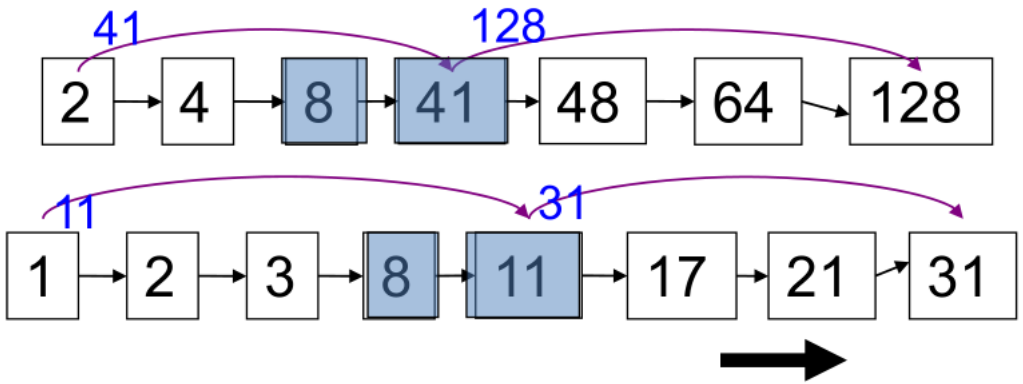




- 通过设置跳表指针跳过部分文档，从而实现快速合并
  - 如何利用跳表指针实现快速合并?
  - 在什么位置设置跳表指针?

- **倒排表合并中的优化问题**

- 带有跳表指针的查询处理过程



- 首先，通过遍历发现了共同的记录8，继续移动指针
- 其次，表2在11的位置，我们发现跳表指针31小于表1的下一个数31
- 因此，我们直接将表2跳到31，而跳过其中的17、21两个数

本节目录

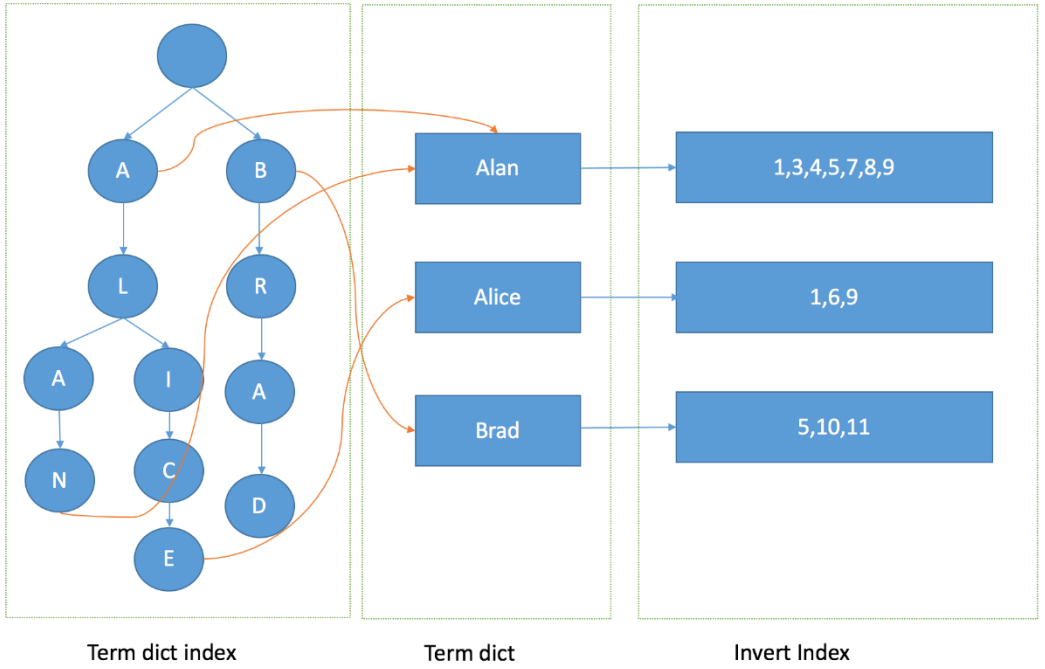
- 布尔检索与倒排索引
- 索引存储
- 搜索匹配与排序

- **倒排索引的存储问题**

- 一种常用方式：词典与倒排表一起存储
  - 便于同时读取，但文档规模大时将导致索引过大，影响性能
- 另一种常用方式：两者分开存储
  - 词典与倒排表分别存储为不同的文件，通过页指针关联
    - 倒排表文件也可采用分布存储的方式
  - 优点：性能大幅提升
    - 词典可以常驻内存，至少常驻一部分（例如主索引）
    - 可以支持并行、分布式查询

- **倒排索引的存储问题**

- 常见的词汇表存储结构
  - 顺序存储
  - Hash Table
  - B+-Tree
  - Trie树

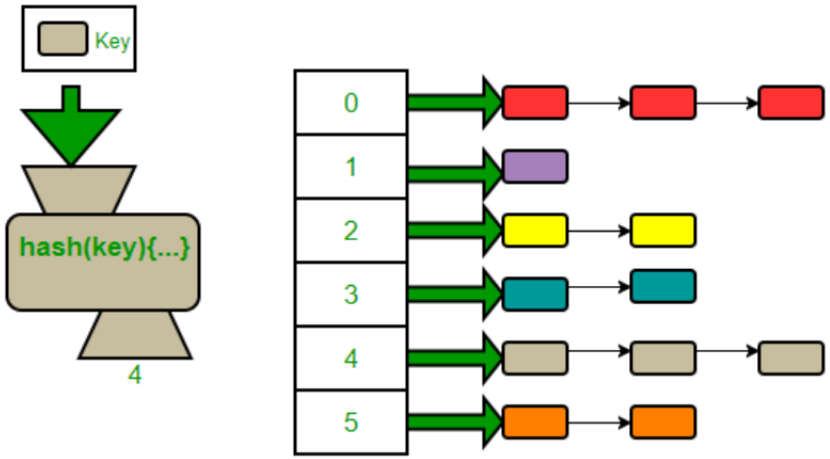


- **最基本的存储方式：顺序存储**

- 即词汇表的顺序排列方式
  - 把词汇表按照字典顺序进行排列（查询的前提）
  - 词汇表的查找采用二分查找法
- 优点：简单粗暴
- 缺点：效率一般
  - 索引构建的效率一般（文档插入需要反复调用查找和排序）
  - 索引检索的效率也一般
    - 二分查找LogN的复杂度，与词汇数量N有关（通常海量词汇）

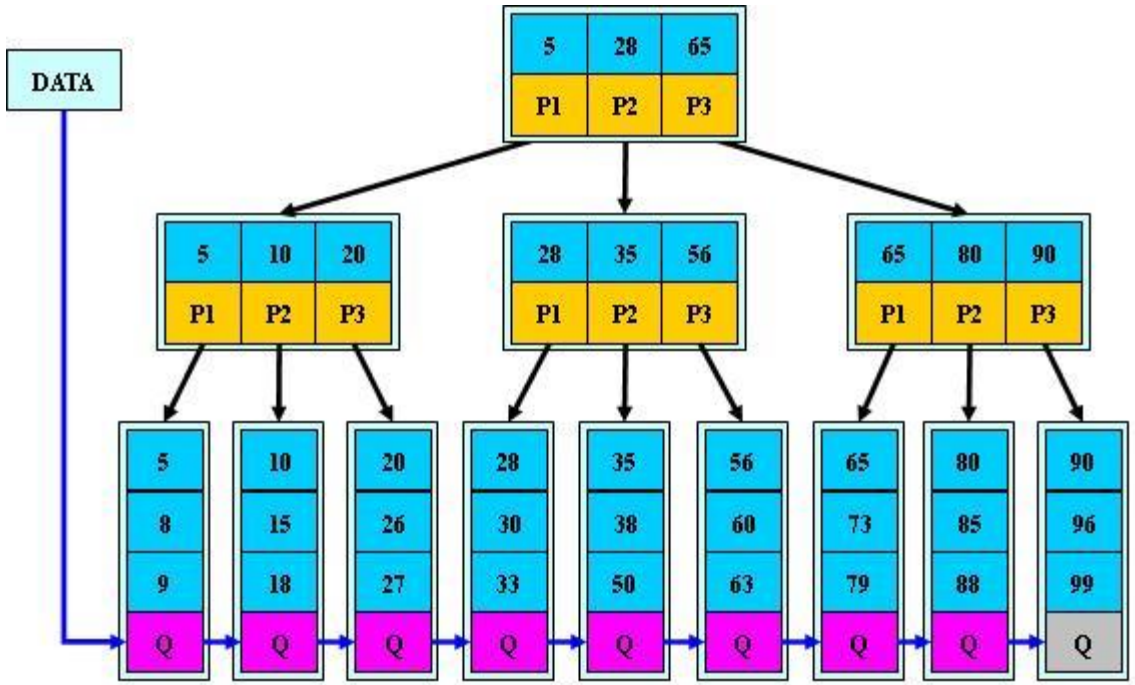
- **改进存储：哈希存储**

- 对词汇表进行哈希
  - 根据给定的词项，散列成一个整数
  - 用该整数作为词项的访问地址
- 优点：实现简单、检索速度快，理论时间O(1)
- 缺点：
  - 当冲突过多时效率会下降
    - 例如，对姓名使用简称进行哈希，则姚明≈杨幂 = YM
  - 关键在于找到一个好的散列函数



- **B+-树存储**

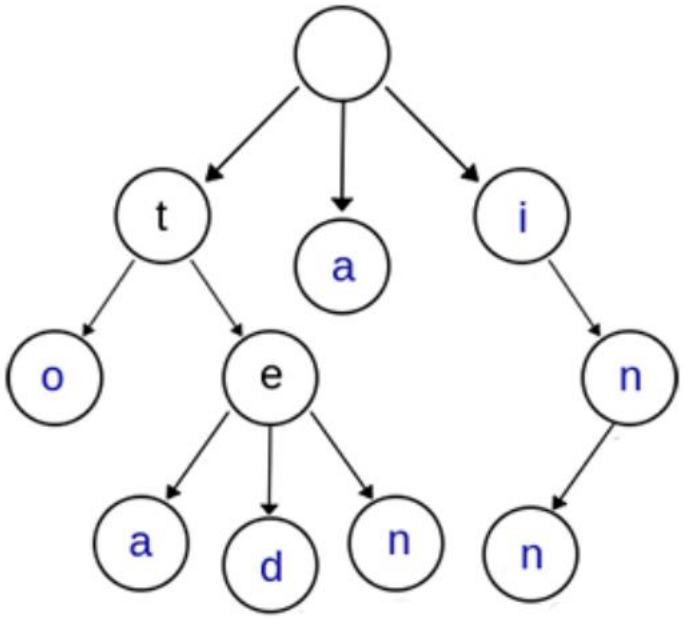
- 对词汇表进行B+-树存储
  - 多叉平衡有序树
- 优点：
  - 性能好且稳定，查找时间 = 树高
- 缺点：
  - 维护代价较高
  - 实现相对复杂



- **Trie树**

- Trie树，又称前缀树，利用字符串的公共前缀来节约存储空间
  - 适用于快速检索的多叉树结构

- 例如，以下Trie树对应词典单词：
  - t、a 、i 、to 、te 、in 、tea 、ted、ten、inn



- 查找时间只与词的长度有关，而与词典中词的个数无关。
  - 当词表较大时，才能体现出速度的优势

- **Trie树**

- Trie树的优缺点：
  - 优点：效率高
    - 查找效率高，与词表长度无关
    - 索引的插入，合并速度快
  - 缺点：所需空间较大，本质是“以空间换时间”
    - 如果是完全m叉树，节点数指数级增长
    - Trie树虽然不是完全m叉树，但所需空间仍然很大，尤其当词项公共前缀较少时

本节目录

- 布尔检索与倒排索引
- 索引存储
- 搜索匹配与排序
  - 相关性
  - 文档质量

- **从匹配到排序**

  - 布尔模型的根本局限性在于二值化匹配
    - 非0即1，无法实现相关性排序
  - 一个好的信息检索系统，应该可以做到：相关度高的文档，应该比相关度低的文档排名更高
    - 所以要有区分！不能吃大锅饭
    - 如何实现？对每个【查询-文档】赋予[0,1]之间的一个分值
      - 该分值度量了文档与查询之间的匹配程度

- **排序问题的难点何在?**

- 传统信息检索方法的两个重要内在假设
  - 被索引的信息具有很高且同等质量，至少在信息的组织和内容上可靠。
  - 信息检索的用户应具有一定的相关知识和技能。
    - 更类似于数字图书馆或专业搜索引擎，e.g., 知网
- 然而，Web Search的现实与假设有巨大落差
  - Web网页的信息组织与内容层次不齐
  - 用户庞杂且缺乏知识和经验
  - 用户意图多样，存在巨大差异

- **一个好的排序应该是怎样?**

- 如果我们有更丰富的信息?
  - B医生，从医二十年，经验丰富
  - C医生，只有五年从医经验
- 从择医的角度考虑，需要同时考虑专长与医术
- 对于**网页排序**而言，也是如此
  - 网页内容匹配程度 → 医生的专长
  - 网页内容的质量 → 医生的经验与水平

- **相关性量化方法（1）：Jaccard系数**

  - 布尔模型的一个假设：所有的查询条件都必须满足
    - 假如打破这一约束，能够实现相关性的[0,1]量化?
  - 1901年，Paul Jaccard提出计算两个集合重合度的方法
    - 如果A与B为两个集合，那么有：

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Paul Jaccard (1868-1944)

    - 显然，我们有：JACCARD(A,A)=1， JACCARD(A,B)=0（如果A ∩ B = 0）

- **相关性量化方法（1）：Jaccard系数**

  - 由此，我们得到了一个描述集合重合度的，处于[0,1]之间的值
    - 注意前提：“集合”，即仍将文档视作词项的集合
  - 通过计算查询与文档之间的重合度，可以初步估算文档的相关性

查询 “ides of March”

文档 “Caesar died in March”

JACCARD(q, d) = 1/6

- **相关性量化方法（1）：Jaccard系数**

  - 然而，Jaccard系数显然仍过于简单、粗糙
    - 从完全匹配到部分匹配是一种进步，但查询词本身未做重要性区分
    - 不考虑词项频率，即词项在文档中的出现次数
    - 没有仔细考虑文档的长度因素
    - 罕见词比高频词的信息量更大，Jaccard系数没有考虑这个信息

- **相关性量化方法（2）：TF-IDF**

  - 比Jaccard更进一步，如何通过词项区分文档权重?
  - 一个直观的想法：查询词在文档中出现得越多，该文档越相关
    - 词项频率TF(t,d)，指词项 t 在文档 d 中出现的次数（Term Frequency）
      - 但存在一定问题，相关性与频率并不线性相关
        - 某个词在文档A中出现100次，在文档B中出现10次，A比B相关10倍?
    - 对数词频：缓和数量级差异性造成的影响

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **相关性量化方法（2）：TF-IDF**

  - 一个什么样的词真正有区分度?

    - 香农（Shannon）有话说:

香农（C. E. Shannon）信息论

信息量是指从N个相等可能事件中选出一个事件所需要的信息度量或含量，也就是在辩识N个事件中特定的一个事件的过程中所需要提问“是或否”的最少次数. 香农信息论应用概率来描述不确定性。信息是用不确定性的量度定义的。
一个消息的可能性愈小，其信息量愈多；而消息的可能性愈大，则其信息愈少。事件出现的概率小，不确定性越多，信息量就大，反之则少。

  - 总结起来：罕见词的信息量更为丰富，而频繁词的信息量相对较少

    - 相应的，如果查询中包含某个罕见词，则包含这个词的文档可能很相关

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

- **相关性量化方法（2）：TF-IDF**

  - TF与DF，从两个角度为衡量文档相关提供了量化依据
    - 如何将两者整合起来，实现更完整的描述？
    - 乘起来就完事了~

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

    - 同时，解答了一个问题：什么样的词适合衡量文档相关性？
      - 在文档内很多，出现的文档很少的词
      - 指标随词项频率增大而增大，随词项罕见度增大而增大

- **相关性量化方法（3）：向量空间模型(Vector Space Model)**
  - 向量空间模型（Vector Space Model，VSM）
    - 每个文档和查询视作一个词项权重构成的向量
    - 查询时通过比较向量之间相似性来进行匹配
  - 文档表示示例

D1对应的向量为(5.25, 1.21, 8.59, 0, 2.85, 1.51, 1.37)

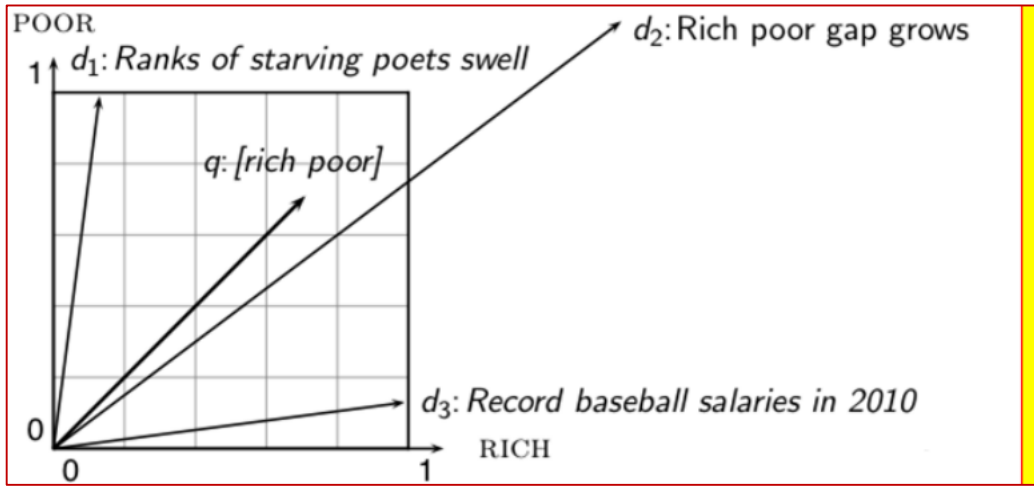
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

- **相关性量化方法（3）：向量空间模型(Vector Space Model)**

- 如何计算查询与文档之间的相似度?
  - 最基本的方法：欧氏距离

$$dist(\vec{x},\vec{y})=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots+(x_n-y_n)^2}$$

  - 然而，欧氏距离并不是一个好的选择，它对于向量长度非常敏感



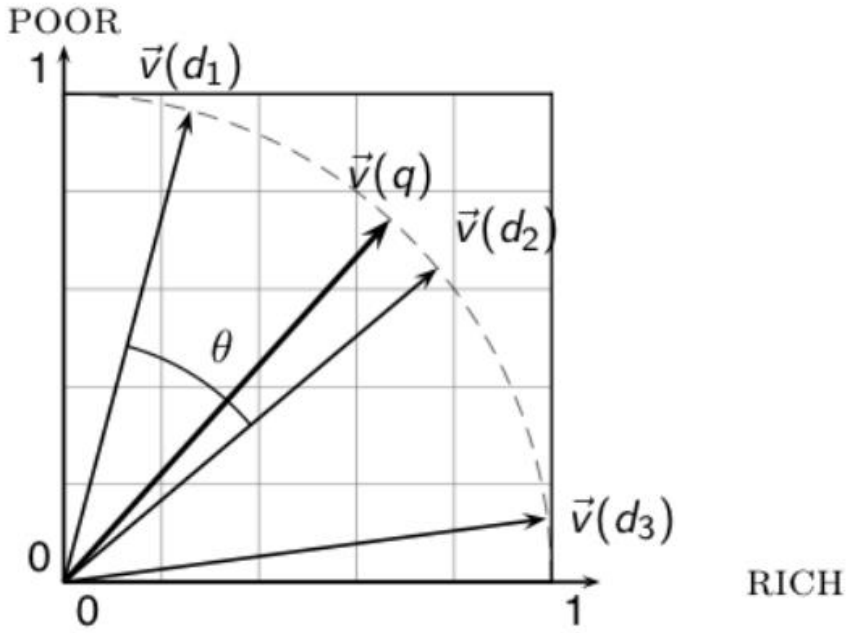
尽管查询*q*和文档d_2的词项分布非常相似，但是采用欧氏距离计算它们对应向量之间的距离非常大

- **相关性量化方法（3）：向量空间模型(Vector Space Model)**

- 更为合适的方案：余弦相似度
  - 按照文档向量与查询向量的夹角大小来计算

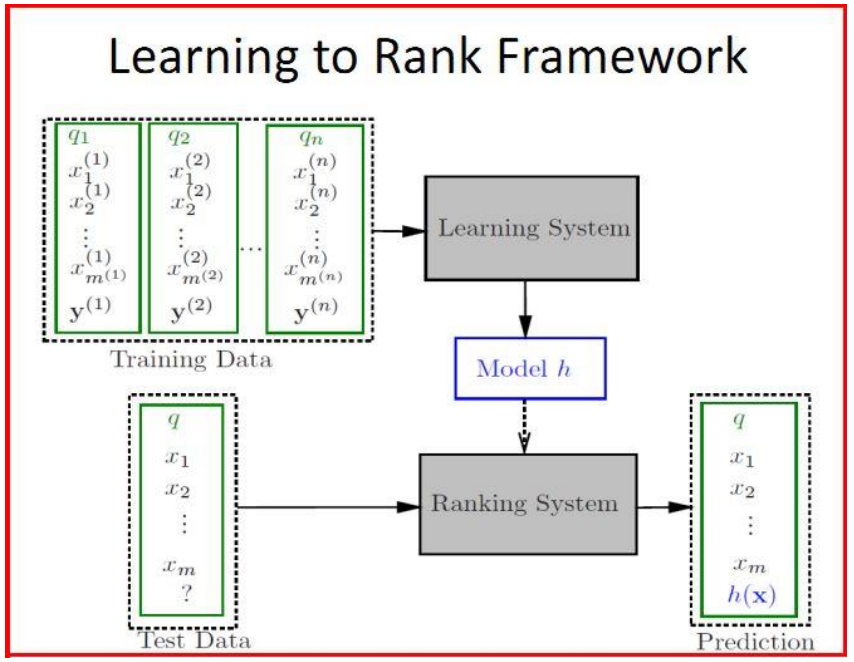
$$\cos(\vec{q},\vec{d}) = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

  - 显然，向量越一致，夹角越小，cosine值越高
    - 相应的，它们之间的相似度也就越高
    - 即使这种情况，它们的欧氏距离可能很大



- **相关性量化方法（4）：学习排序(Learning to Rank)**

- 对于网页内容的排序问题，本质上是一个学习排序器（Ranker）的问题
  - 输入网页（文档）内容及查询，输出该网页合适的排序位置
  - 排序学习是个有监督学习问题
    - 基于已知的排序（如用户反馈）
    - 为新网页-查询对给出排序
  - 位置更为敏感，排在前面的部分更重要
    - 用户可能只看开头的几个页面
  - 评价指标基于排序结果进行，不是简单的分类问题



- **相关性量化方法（4）：学习排序(Learning to Rank)**

- 三类常见的排序学习算法
  - Pointwise，将排序退化为分类或回归问题
    - 输出：网页对应的分类（有序）、回归值或有序回归值
  - Pairwise：比较一对网页之间的相关度，e.g., 相关 > 不相关
    - 输出：网页对之间的偏序关系
  - Listwise：对整个网页集合进行排序
    - 输出：整个集合的完整排序，往往依赖特定排序指标

- **相关性量化方法（4）：学习排序(Learning to Rank)**

- 一个常用的Learning to Rank工具包：Ranklib
  - 原始网站：http://people.cs.umass.edu/~vdang/ranklib.html（已关闭）
  - 目前可用：https://sourceforge.net/p/lemur/wiki/RankLib/
  - 包含多种主流排序学习算法，文件格式及操作简单
    - RankBoost、AdaRank、LambdaMART等
  - 另外，SVM-Rank也是一个好用的工具包
    - http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

本节目录

- 布尔检索与倒排索引
  - 倒排表的构建与查询
  - 倒排表的优化与扩展
- 索引存储
- 搜索匹配与排序
  - 相关性
  - 文档质量

- 从Web1.0到Web2.0

  我们每个人，既是信息的消费者，也是信息的生产者

twitter

新浪微博 weibo.com

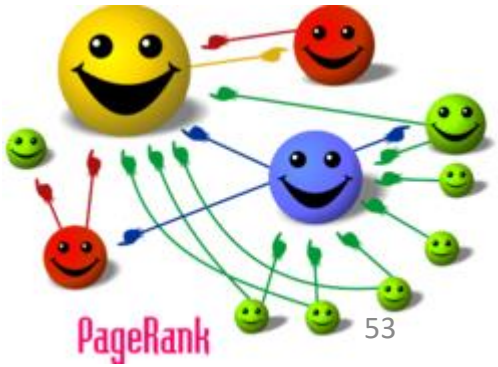
YouTube

bilibili

- 大众创造内容的时代后遗症：内容权威性的下降
  - 信息的错漏、谣言与虚假信息的兴起
  - 低质内容服务商大量涌现
- 如何在衡量相关性的同时，确保内容权威性？

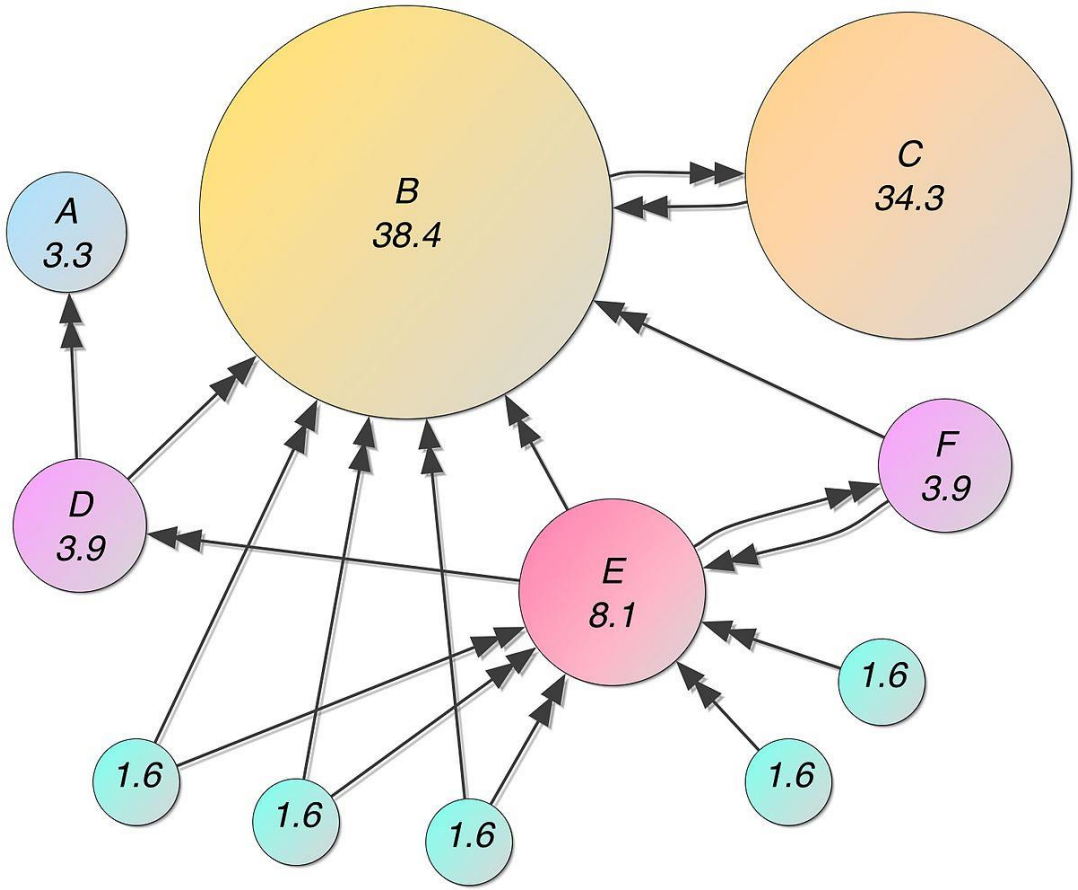
- **PageRank的历史**

- Sergey Brin 和Lawrence Page 在1998年提出了PageRank 算法
  - Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Stanford InfoLab, 1999
    - 截止2019.10.16，引用12437次
    - http://www-db.stanford.edu/~backrub/pageranksub.ps
  - Sergey Brin, Lawrence Page, The Anatomy of a Large-scale Hypertextual Web Search Engine. WWW'98, Computer Networks 30(1-7): 107-117 (1998)
    - 截止2019.10.16，引用18541次
    - https://ai.google/research/pubs/pub334.pdf



- **PageRank的核心思想**

- 将网页（或文档）视作一个点，网页之间的超链接视作一条边，从而形成一个巨大的有向图
  - 点之间的有向连接表示了网页之间相互引用，相互推荐的关系
  - 入度越多，则被引用或推荐的次数越多，网页的重要性就越大

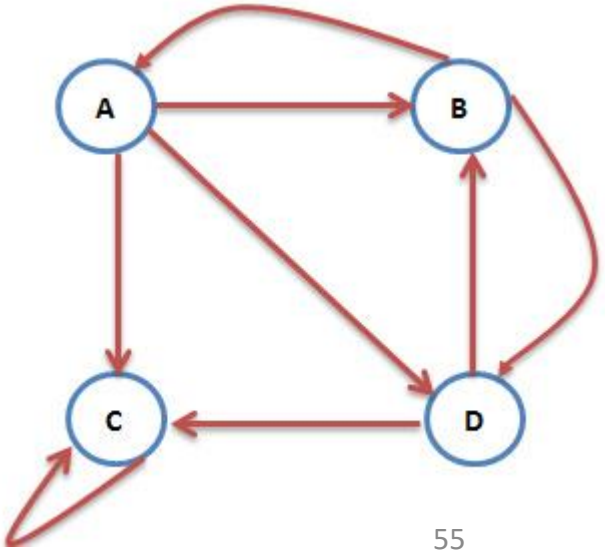


- **PageRank的计算方法**

- PageRank的核心公式如下：

$$\mathrm{PR}(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{\mathrm{PR}(p_j)}{L(p_j)}$$

  - 其中：
    - PR(pi)为网页pi的PageRank值
    - PR(pj)为指向网页pi的某个网页pj的PageRank值
    - L(pj)为网页pj发出的链接数量
    - d为阻尼系数，取值在0-1之间
    - N为网页总数，M(pi)为链入pi的页面集合

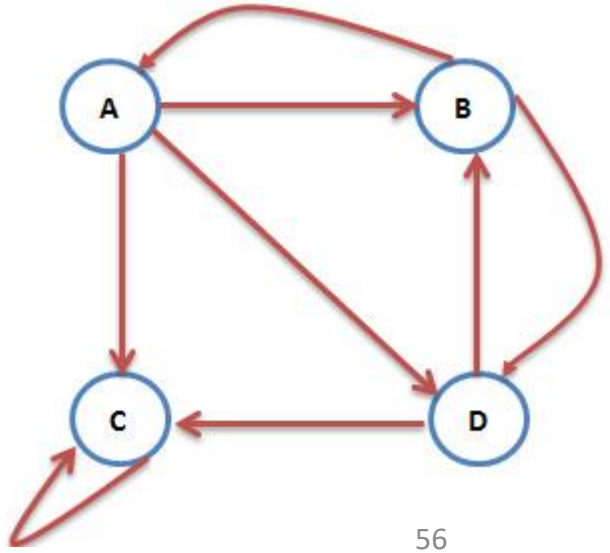


- **PageRank的计算方法**

- 随机游走中的Restart机制

$$\mathrm{PR}(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{\mathrm{PR}(p_j)}{L(p_j)}$$

  - 其中的(1-d)/N的部分，相当于以一定概率重新选择起点
    - 此时，所有节点以一定等概率被选中
    - 由此，跳出了陷阱和黑洞的干扰
    - d一般选择为0.85左右



• PageRank的拓展模型：个性化PageRank

- PageRank在衡量网页的整体权威性方面具有显著的意义
- 然而，用户的个性化因素却没有在PageRank中得以体现
  - 每个用户有自己特定的偏好，绝大多数网页并不会被用户浏览
  - 一种解决方案是将用户偏好排序与PageRank值结合起来
    - 从用户喜爱的网页中挑选更权威的，较为简便，但可能效果一般
  - 另一种方案，模拟用户自己的网页浏览行为，从而计算个性化权威
    - 个性化PageRank（Personalized PageRank）

- **PageRank的拓展模型：个性化PageRank**

- 在个性化PageRank中，从用户出发的跳转概率不一定视作均等（如1/N）

$$\mathrm{PR}(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{\mathrm{PR}(p_j)}{L(p_j)}$$

修改为反映用户偏好的向量v

  - 可根据用户对不同网页的偏好决定其概率
  - v 体现了用户偏好，可视作不同网站在用户偏好下的点击概率

- **PageRank的拓展模型：主题敏感PageRank**

- 个性化PageRank固然体现了用户偏好，然而代价过于巨大
  - 为每一个用户单独计算其PageRank值事实上是个Mission Impossible
- 如何既在一定程度上考虑偏好因素，又减轻计算资源的负担？
  - 一个折中的方案是，以主题为中介，为特定的主题计算相应的PageRank
    - TH Haveliwala, Topic-sensitive pagerank, WWW‘02，已引用2052次
    - 某种意义上，这一模型体现了“仅有同行的评价才有价值”的思想
      - 隶属不同主题的网站，其相互引用和推荐的作用将被削弱

- **从PageRank到HITS算法**

- 在PageRank提出的同一年（1988），康奈尔大学的Jon Kleinberg（又一尊大神）提出了Hyperlink – Induced Topic Search（HITS）
  - Kleinberg博士认为，既然搜索是用户发起的提问，那么页面的重要性衡量就应该遵循用户的检索意图

Kleinberg J M. Authoritative sources in a hyperlinked environment, In Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA 1998). 1998. （引用3015次）

Kleinberg J M. Authoritative sources in a hyperlinked environment[J]. Journal of the ACM (JACM), 1999, 46(5): 604-632. （引用9530次）

- **HITS算法的两个核心概念**

- 权威（Authority）网页与枢纽（Hub）网页的区分
- 权威网页：指某个领域或某个话题相关的高质量网页
  - 如科研领域的中科院之声，视频领域的优酷与爱奇艺等
- 中心网页：类似中介，指向了很多高质量的权威网页
  - 如“hao123”，各个浏览器自带的首页
- HITS的目的即在海量网页中找到并区分这些与用户查询主题相关的高质量“Authority”与“Hub”网页，尤其是“Authority”（因为更能满足用户的信息需求）

- **HITS算法的两个基本假设**

- 基本假设与核心概念相互对应
  - 基本假设1：好的Authority会被很多好的Hub指向

$$\forall p, a(p) = \sum_{i=1}^{n} h(i),$$

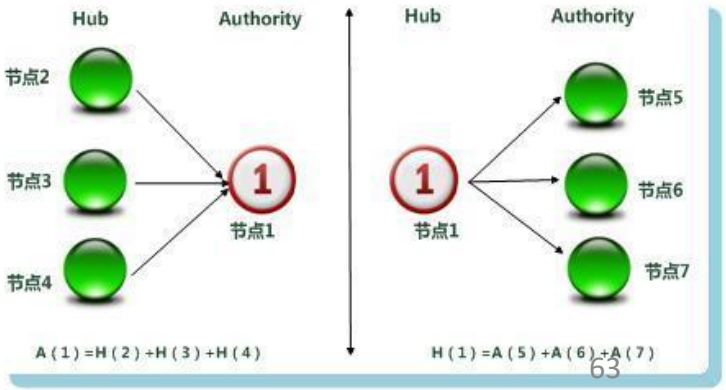
  - 基本假设2：好的Hub会指向很多个好的Authority

$$\forall p, h(p) = \sum_{i=1}^{n} a(i)$$

  - 因此，在HITS算法中，每个网页需要计算两个权威值

- **HITS算法的计算过程**

- 基于先前的基本假设，HITS的计算过程如下：
- 首先，根据关键字获取与查询最相关的少数页面，及与这些页面有链接关系的页面，作为待选集合
- 其次，对所有网页的 a(p) 与 h(p) 进行初始化，可都设为1
- 最后，迭代计算两个步骤，即基本假设所对应的两个公式
  - 重复这一步，直到最终收敛为止
  - 输出Authority或Hub值较高的页面



- **HITS算法的计算过程**

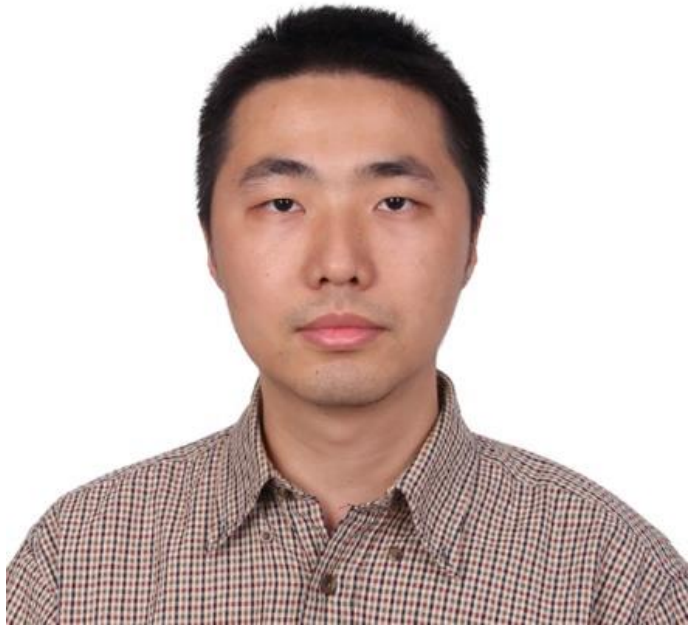
- 基于先前的基本假设， HITS的计算过程如下：
  - 假定邻接矩阵为M，Authority向量为a，Hub向量为h
  - 则有如下迭代式：
    - $a_{k+1} = M^T h_k$， $h_{k+1} = M a_{k+1}$
  - 或者，可采用如下迭代式：
    - $a_{k+1} = (M^T M)^k M^T a_0$， $h_{k+1} = (M M^T)^{k+1} h_0$
    - 其中，a_0, h_0为Authority/Hub向量的初始值，可设为全1向量

HITS算法的优缺点

- 相比于PageRank，HITS算法是一种能够区分网页功能的排序算法
- 优点
  - 更好地描述互联网组合特点
  - 主题相关，因此可以单独用于网页排序
- 缺点
  - 需要在线计算，时间代价较大
  - 对链接结构变化敏感，且依然可能受到“链接作弊”的影响

接下来

业界分享：淘宝搜索算法介绍



个人简介：陈曦（阿里花名：公达），2009年武汉大学计算机学院博士毕业，加入阿里巴巴集团研发院，2010年加入淘宝搜索算法部门。在淘宝搜索排序、反作弊、和相关性方向有10年以上的工作经验。现在是淘宝搜索相关性和Query理解方向的负责人。