信息组织与检索

主讲人：张蓉
华东师范大学
数据科学与工程学院

信息检索
Information Retrieval

- Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfifies 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。
- 关键词
  - Document –文档
  - Unstructured – 非结构化
  - Information need –信息需求
  - Collection—文档集、语料库
  - Organization-组织
  - Storage-存储

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
- 允许
  - 关键词加上操作符号的查询
  - 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

<title>李甲主页</title>

<body>…</body> …

- 半结构化查询
  - Title contains data AND Bullets contain search

華東師範大學
EAST CHINA NORMAL UNIVERSITY

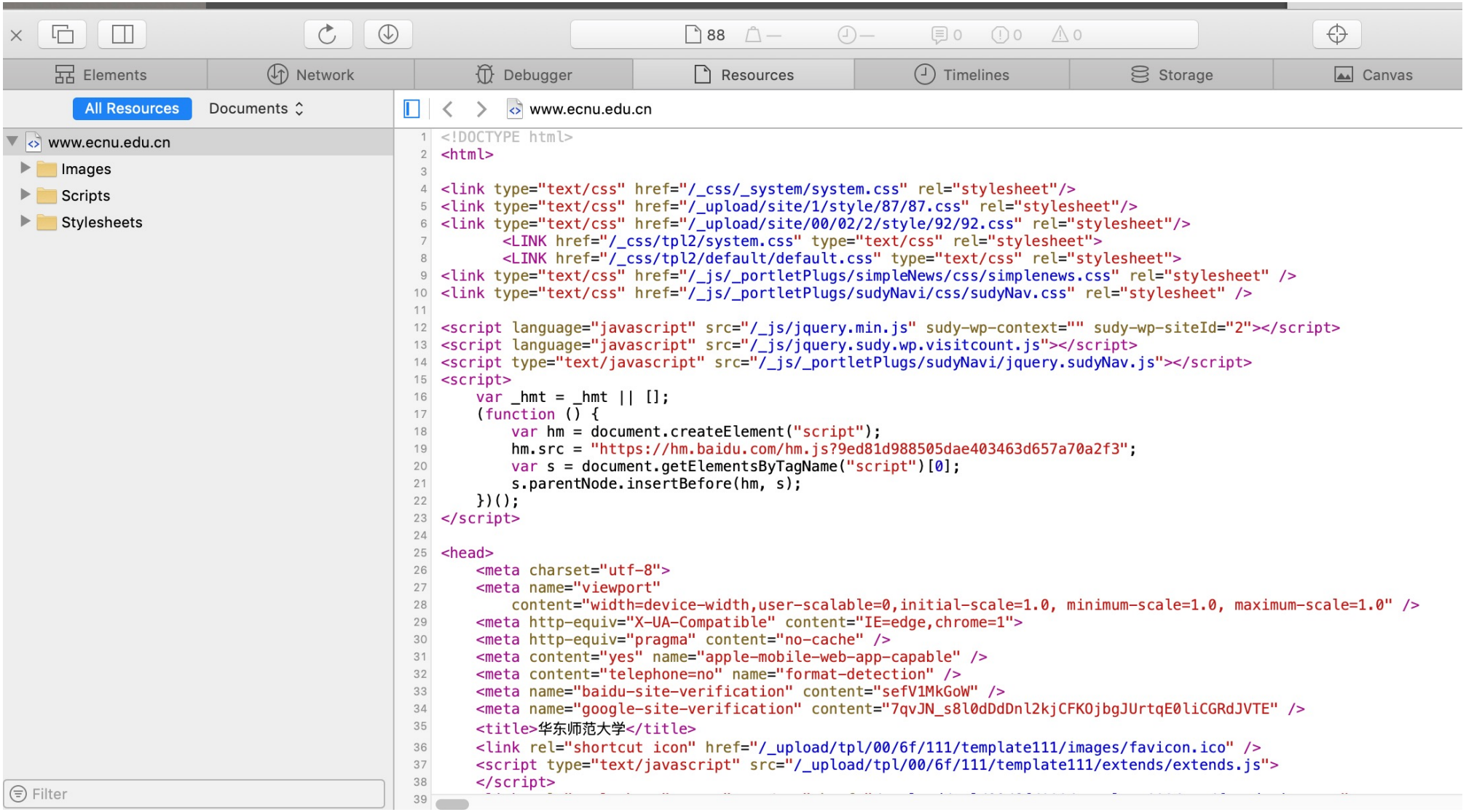
热点新闻

媒体视点

讲座报告

2020 08-14
张正彪
基于马来酰亚胺化学的高分子序列调控
腾讯会议（ID:105 181 836）

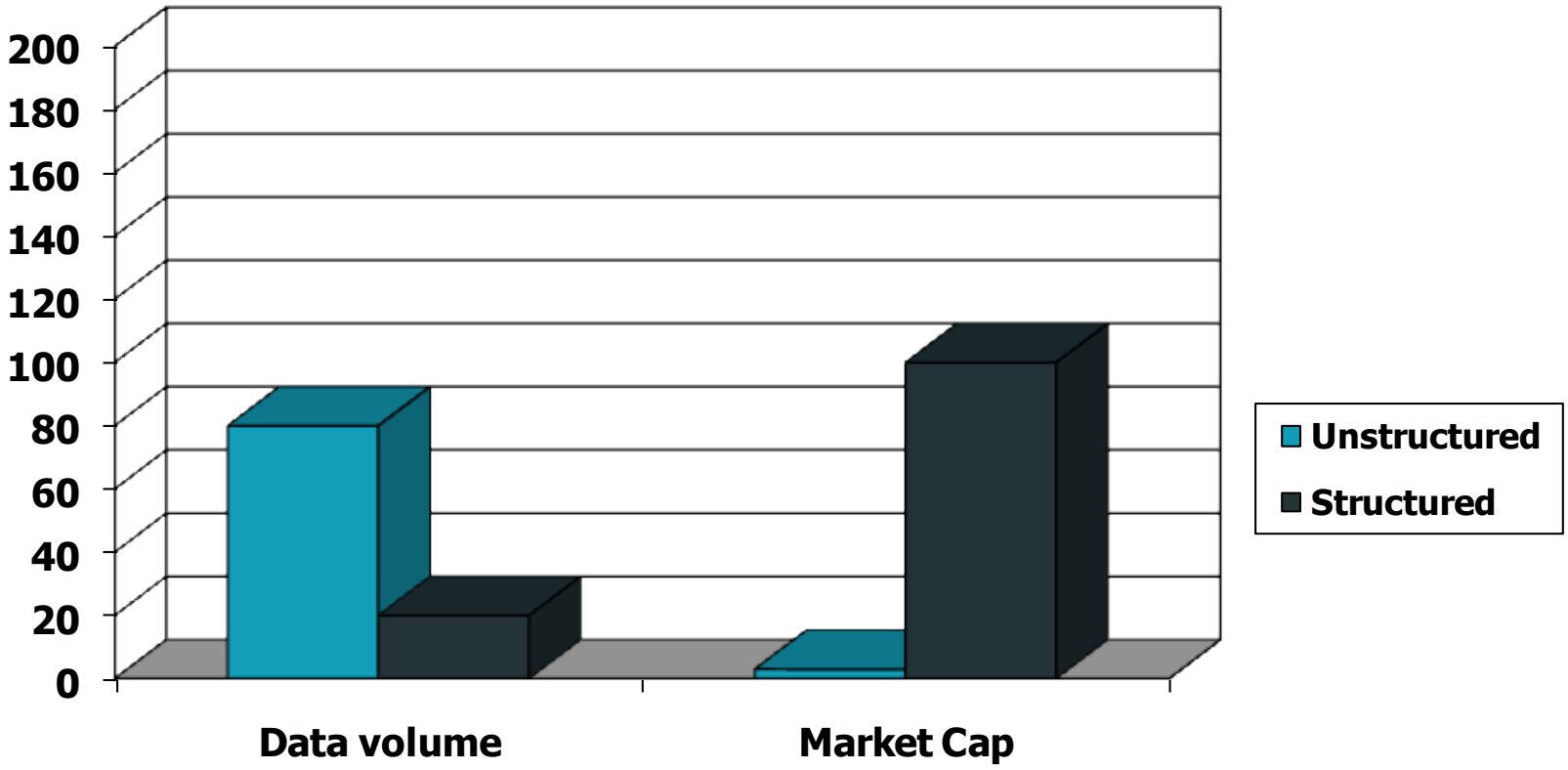
2020 08-10
George A. Elliott等
算子代数研究中心暑期国际研讨活动
Zoom会议（ID: 630


88
Elements
Network
Debugger
Resources
Timelines
Storage
Canvas
All Resources
Documents
www.ecnu.edu.cn
Images
Scripts
Stylesheets
Filter
www.ecnu.edu.cn
<!DOCTYPE html>
<html>
<link type="text/css" href="/_css/_system/system.css" rel="stylesheet"/>
<link type="text/css" href="/_upload/site/1/style/87/87.css" rel="stylesheet"/>
<link type="text/css" href="/_upload/site/00/02/2/style/92/92.css" rel="stylesheet"/>
<LINK href="/_css/tpl2/system.css" type="text/css" rel="stylesheet">
<LINK href="/_css/tpl2/default/default.css" type="text/css" rel="stylesheet">
<link type="text/css" href="/_js/_portletPlugs/simpleNews/css/simplenews.css" rel="stylesheet" />
<link type="text/css" href="/_js/_portletPlugs/sudyNavi/css/sudyNav.css" rel="stylesheet" />
<script language="javascript" src="/_js/jquery.min.js" sudy-wp-context="" sudy-wp-siteId="2"></script>
<script language="javascript" src="/_js/jquery.sudy.wp.visitcount.js"></script>
<script type="text/javascript" src="/_js/_portletPlugs/sudyNavi/jquery.sudyNav.js"></script>
<script>
var _hmt = _hmt || [];
(function () {
var hm = document.createElement("script");
hm.src = "https://hm.baidu.com/hm.js?9ed81d988505dae403463d657a70a2f3";
var s = document.getElementsByTagName("script")[0];
s.parentNode.insertBefore(hm, s);
})();
</script>
<head>
<meta charset="utf-8">
<meta name="viewport"
content="width=device-width,user-scalable=0,initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<meta http-equiv="pragma" content="no-cache" />
<meta content="yes" name="apple-mobile-web-app-capable" />
<meta content="telephone=no" name="format-detection" />
<meta name="baidu-site-verification" content="sefV1MkGoW" />
<meta name="google-site-verification" content="7qvJN_s8l0dDdDnl2kjCFKOjbgJUrtqE0liCGRdJVTE" />
<title>华东师范大学</title>
<link rel="shortcut icon" href="/_upload/tpl/00/6f/111/template111/images/favicon.ico" />
<script type="text/javascript" src="/_upload/tpl/00/6f/111/template111/extends/extends.js">
</script>

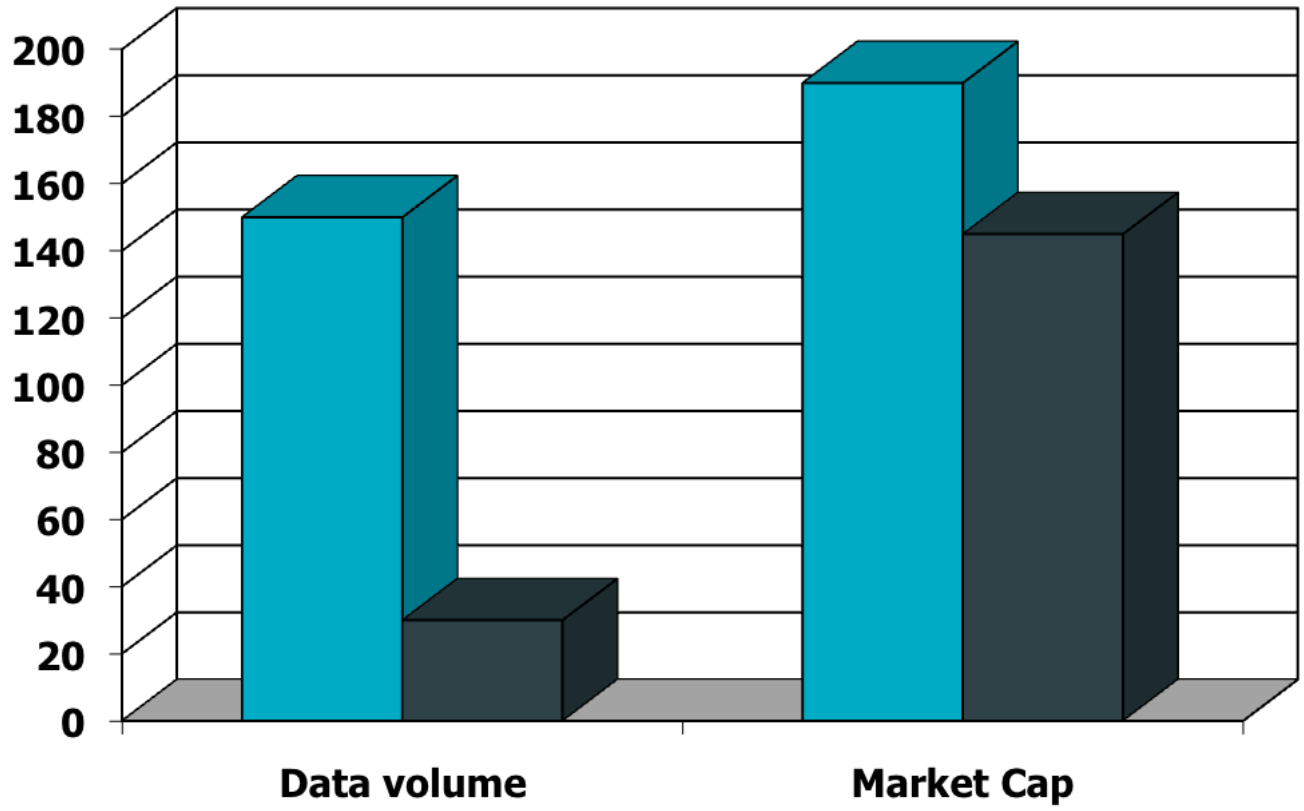
传统信息检索 vs. 现代信息检索

- 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

非结构化数据(文本) vs. 结构化数据(数据库) @ 1996年



非结构化数据(文本) vs. 结构化数据(数据库) @ 2009年



非结构化数据(文本) vs. 结构化数据 @ 2012

- 数据每两年翻一番(2012~2020)
- 2012年，23%的数据有实际分析价值，其中只有3%数据有标记(结构化数据)
- 2020年，33%的数据有实际分析价值

结构化数据好处理，但大部分是文本数据

- 1000 Gigabytes = 1 Terabyte
- 1000 Terabytes = 1 Petabyte
- 1000 Petabytes = 1 Exabyte
- 1000 Exabytes = 1 Zettabyte

IDC/EMC2012 https://www.emc.com/about/news/press/2012/20121211-01.htm

提纲

- 信息检索概述
- **倒排索引**
- 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**张无忌**?
  - 布尔表达式为 郭靖 AND 黄蓉 AND NOT **张无忌**
- 怎么完成查找呢?

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**张无忌**?
  - **布尔表达式**: 郭靖 AND 黄蓉 AND NOT **张无忌**
- 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**张无忌**
- 为什么是笨方法？Not good!

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**张无忌**?
  - 布尔表达式为 郭靖 AND 黄蓉 AND NOT**张无忌**
- 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**张无忌**
- 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., **找同时提到郭靖和黄蓉的段落**)
  - 不支持检索结果的排序 (即只返回较好的结果)

是否有简便方法?

- 目标：查询的快一点

- 回顾一下哈希表
  - 哈希表?
  - 如何判断某个词是否被包含在一个大词典?
  - 新华词典的查字方式比较

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
杨过	0	1	0	1	0

郭靖 AND 黄蓉 *BUT NOT* 张无忌

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* **张无忌**

- 取出三个行向量， 并对张 的行向量求补，最后按位进行与操作
  - 郭靖= 11010
  - 黄蓉= 11010
  - NOT张无忌=NOT 00010 = 11101
- 11010 AND 11010 AND 11101 = 11000.

上述查询的结果文档

- 结果：射雕英雄传/神雕侠侣
- 更复杂的查询例子？

	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
杨过	0	1	0	1	0

信息检索服务中的基本假设

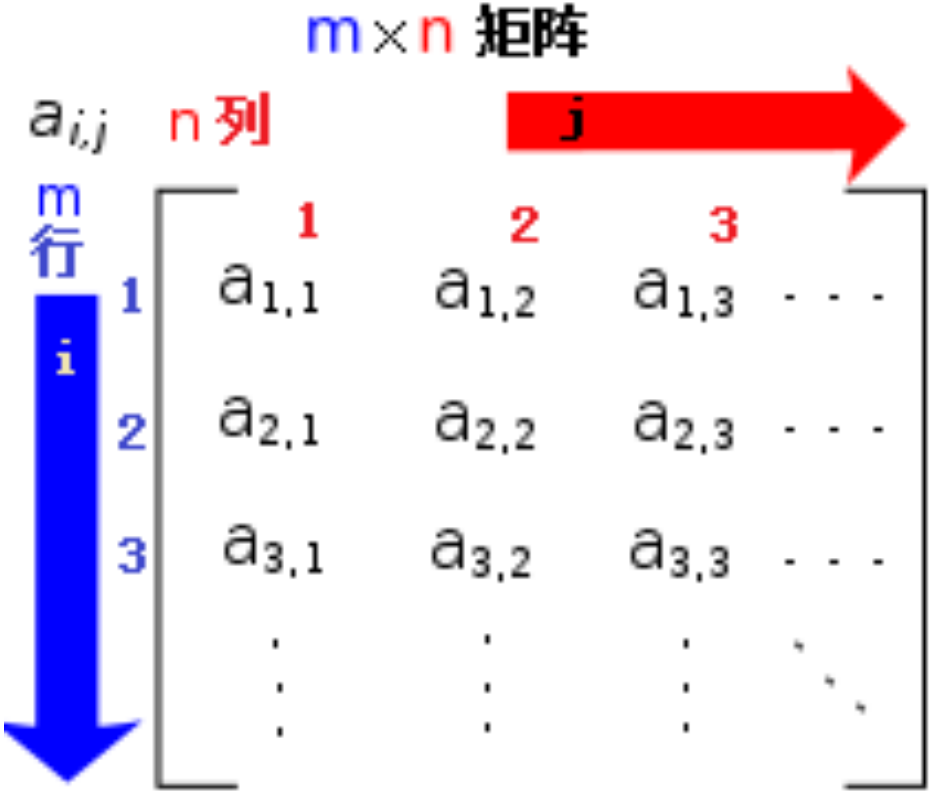
- 文档集Collection: 由固定数目的文档组成
- 目标: 返回与用户需求相关的文档并辅助用户来完成某项任务
- 相关性Relevance
  - 主观的概念
  - 反映对象的匹配程度
  - 不同应用相关性不同

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
  - 将在后面介绍(有兴趣的可以先看)

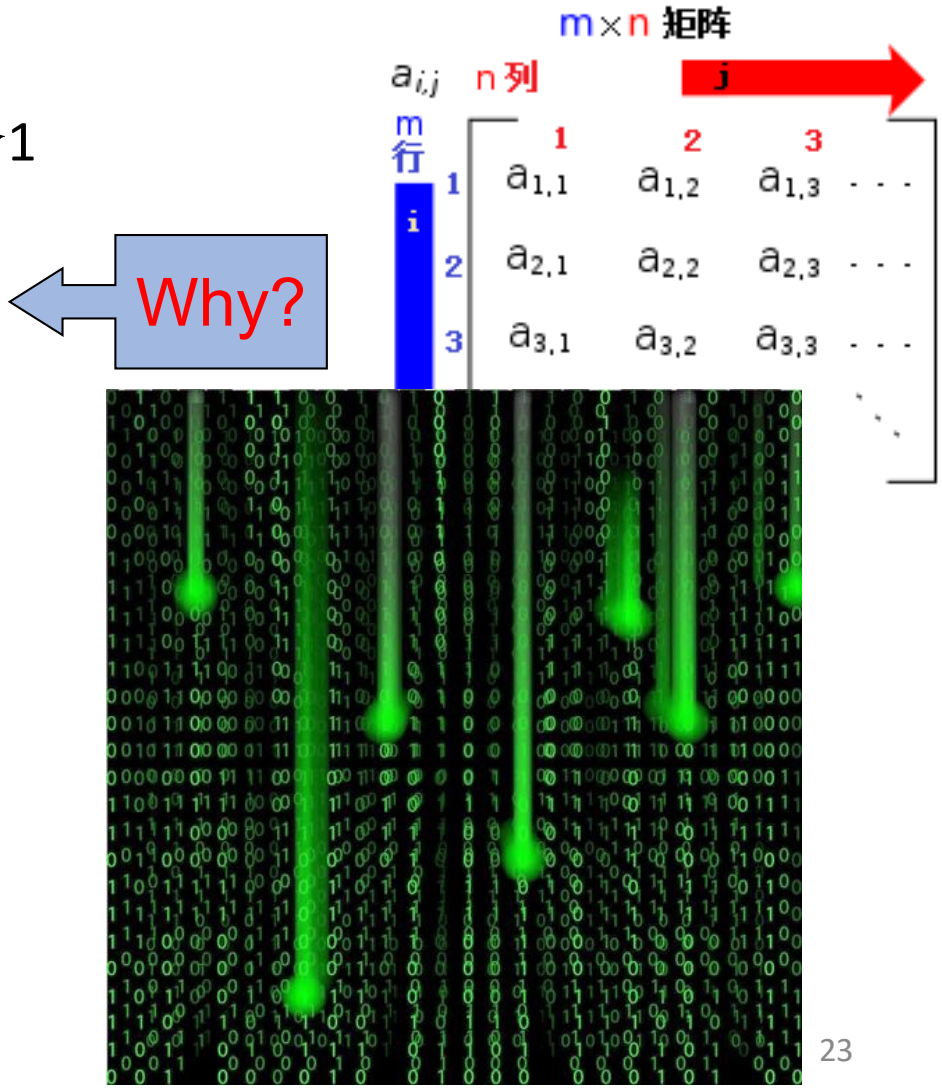
大文档集

- 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) D = 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

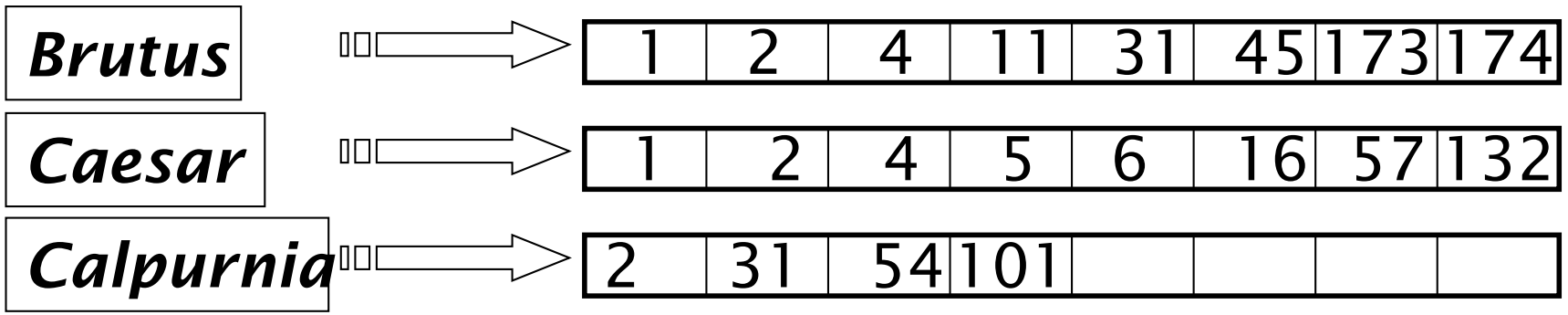
	射雕英雄传	神雕侠侣	天龙八部	倚天屠龙记	鹿鼎记
郭靖	1	1	0	1	0
黄蓉	1	1	0	1	0
洪七公	1	1	0	0	0
张无忌	0	0	0	1	0
杨过	0	1	0	1	0

倒排索引


单词ID
单词
倒排列表（DocID）
1 谷歌 1,2,3,4,5
2 地图 1,2,3,4,5
3 之父 1,2,4,5
4 跳槽 1,4
5 Facebook 1,2,3,4,5
6 加盟 2,3,5
7 创始人 3
8 拉斯 3,5
9 离开 3
10 与 4
11 Wave 4
12 项目 4
13 取消 4
14 有关 4
15 社交 5
16 网站 5

倒排索引(Inverted index)

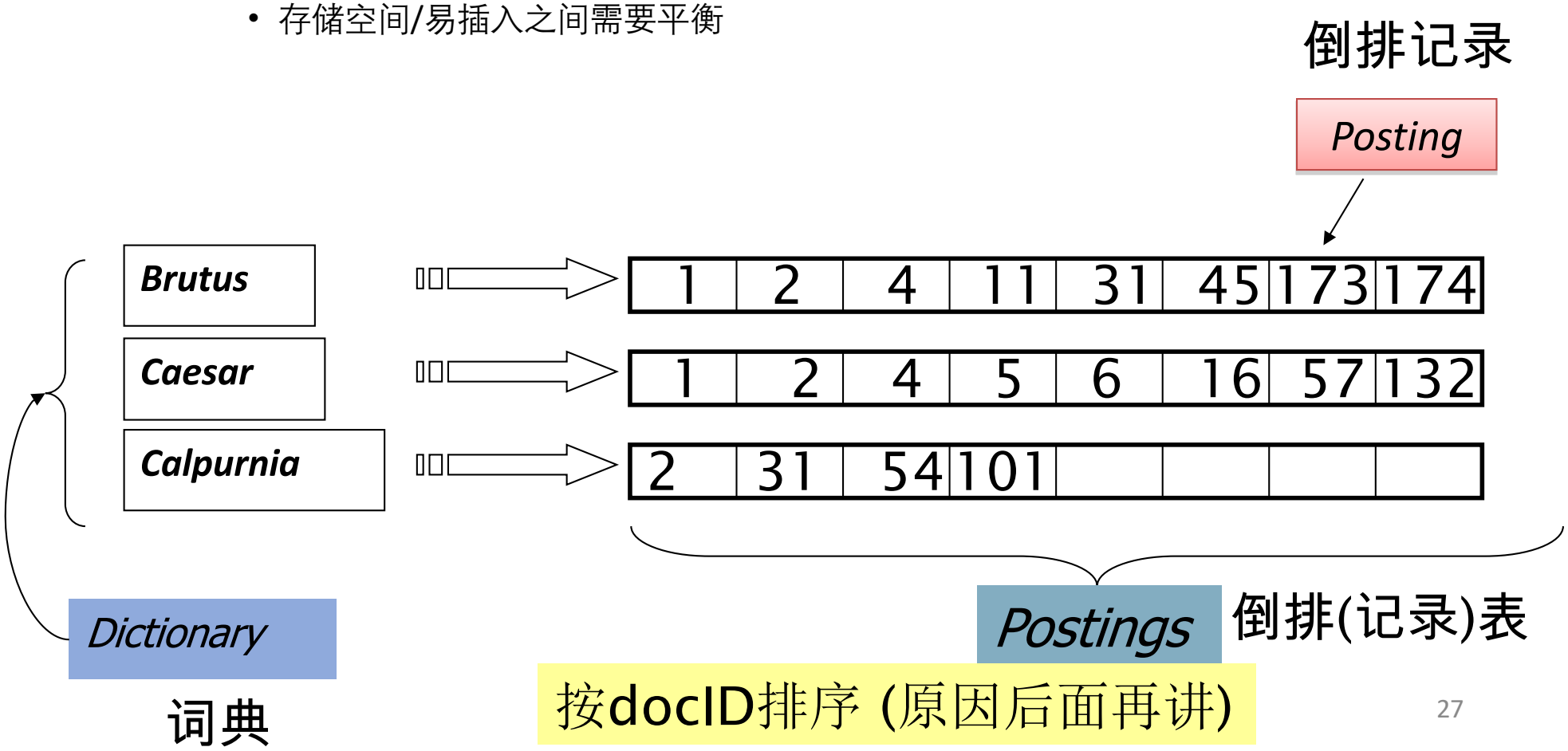
- 对每个词项t, 记录所有包含t的文档列表.
  - 每篇文档用一个唯一的 docID来表示，通常是正整数，如1,2,3…
- 能否采用定长数组的方式来存储docID列表



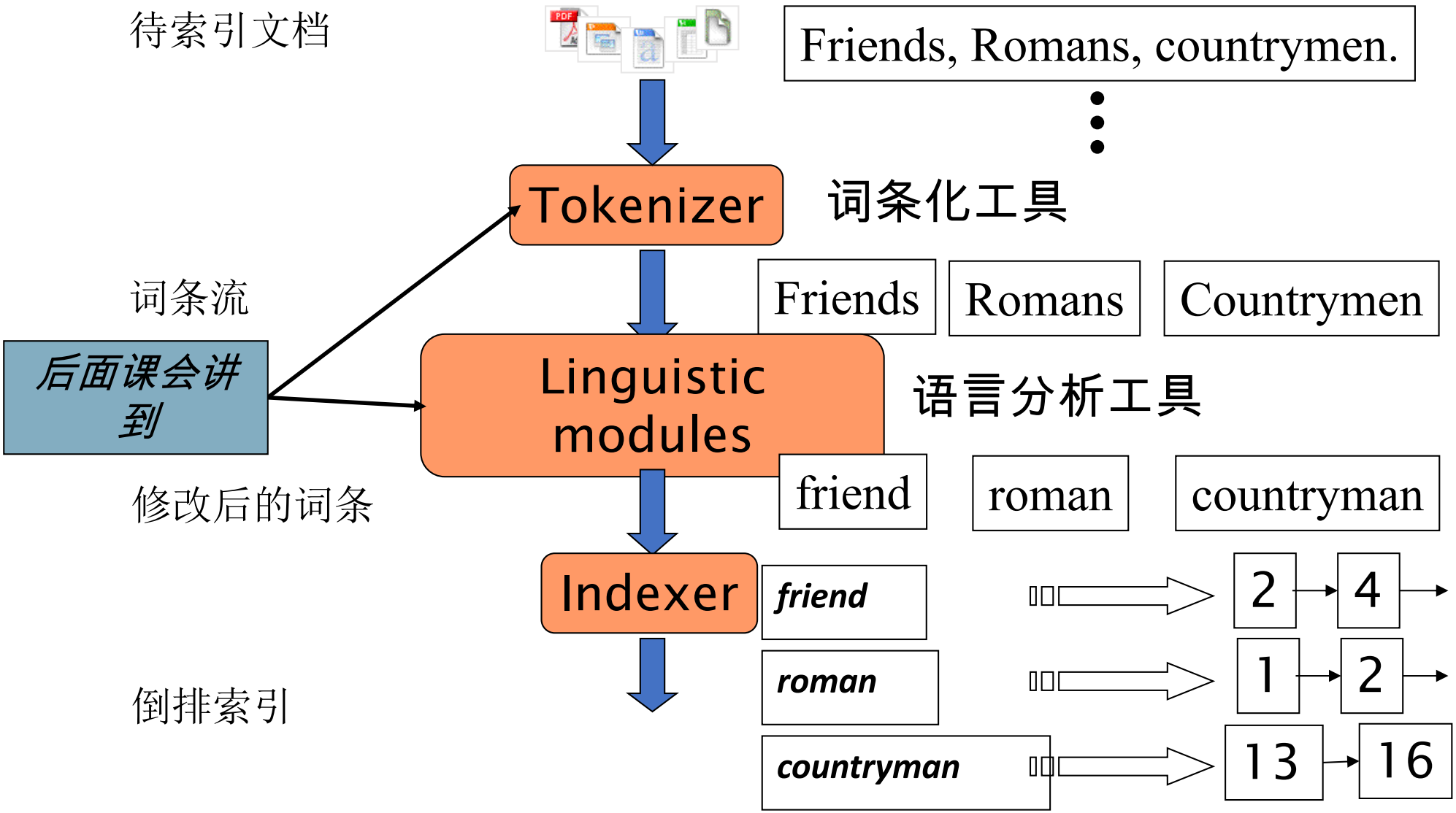
文档14中加入单词*Caesar*时该如何处理?

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
后面课会讲到
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

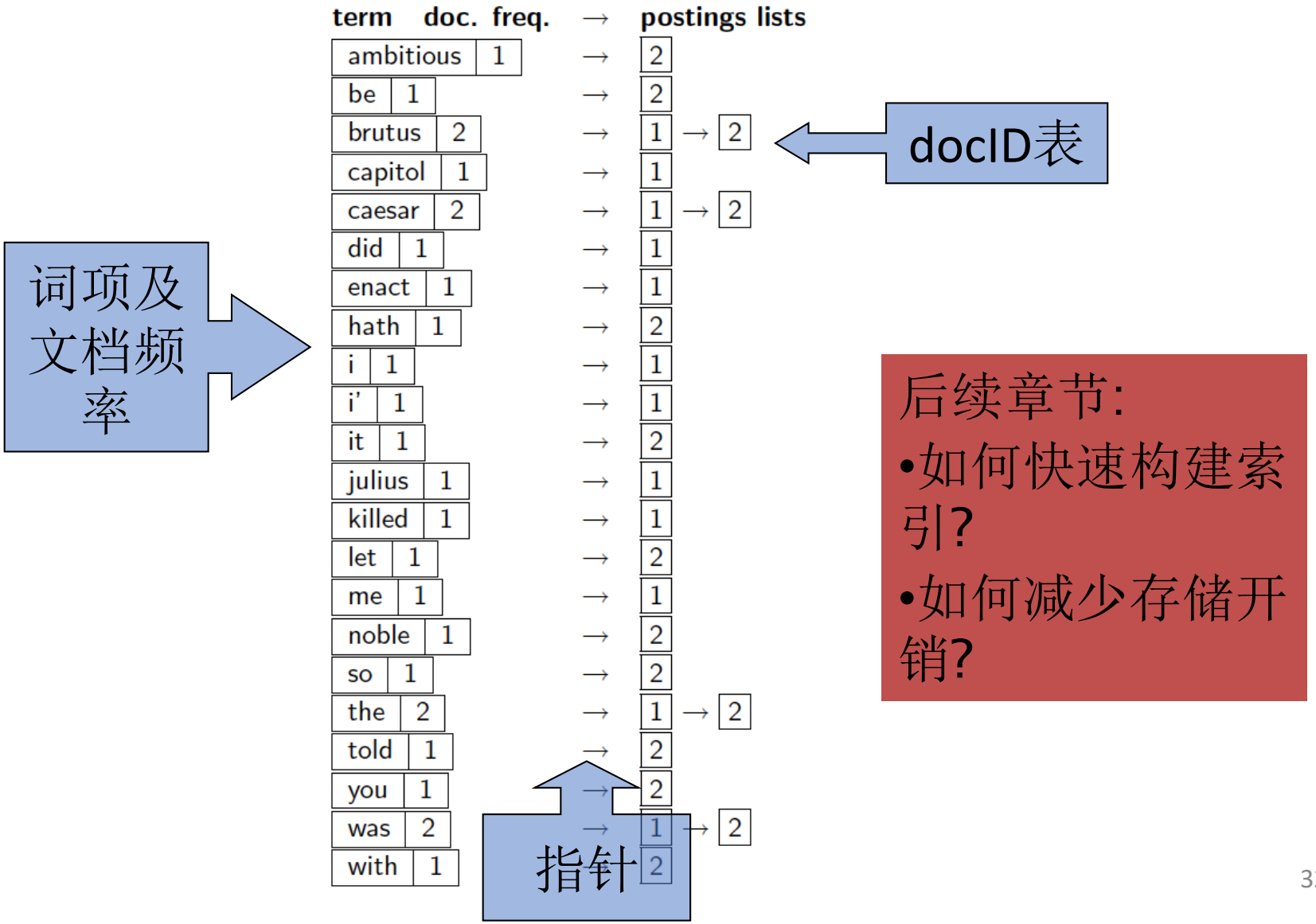
- 某个词项在单篇文档中的多[illegible]被合并
- 拆分成词典和倒排记录表两[illegible]
- 每个词项出现的文档数目(d[illegible]ncy, DF)[illegible]

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 → 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
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?

提纲

- 信息检索概述
- 倒排索引
- **布尔查询的处理**

假定索引已经构建好

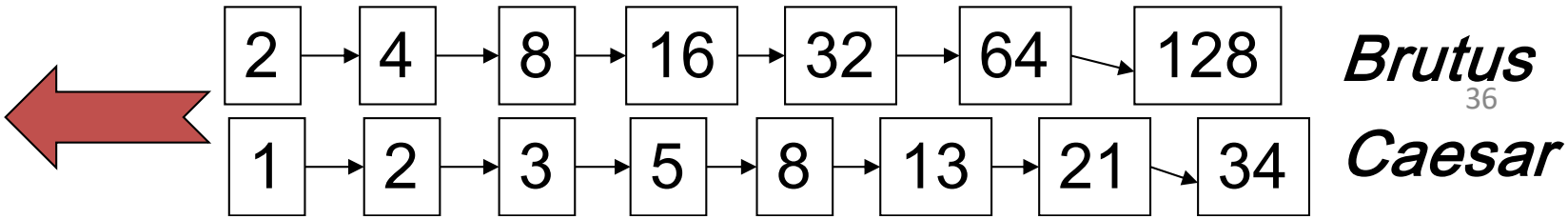
- 如何利用该索引来处理查询?
  - 后面会讲 – 如何处理不同类型的查询? 比如带通配符的查询 “信息*检索”

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

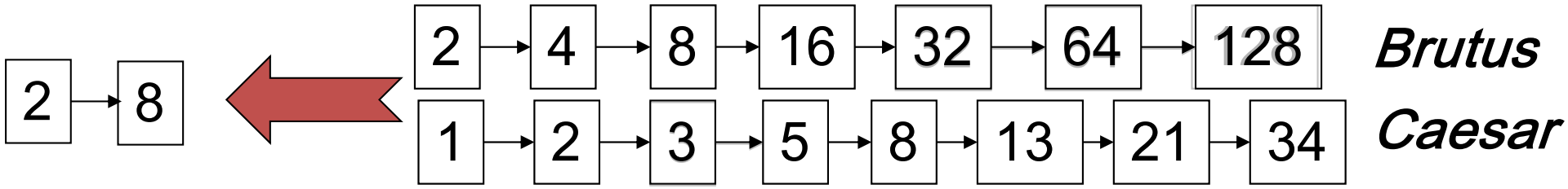
AND查询的处理

- 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

上述合并算法的伪代码描述

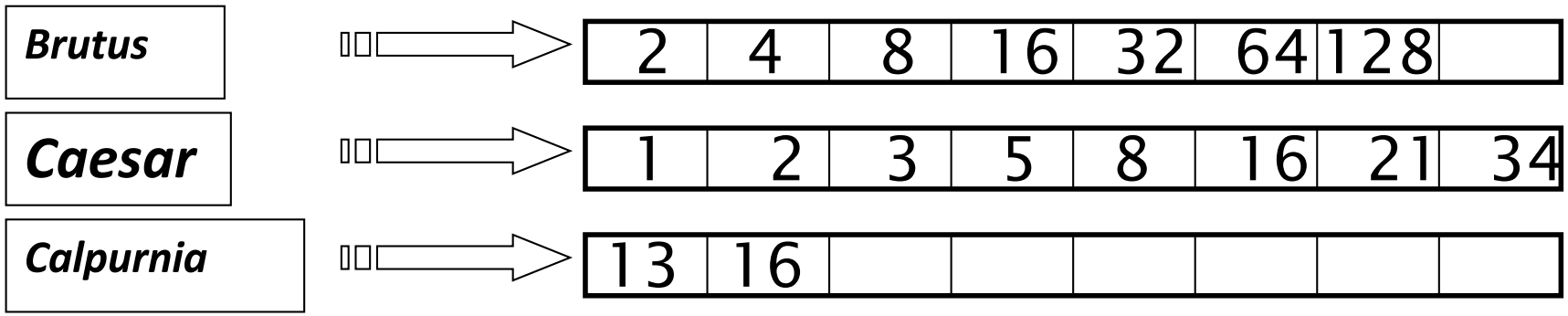
```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

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

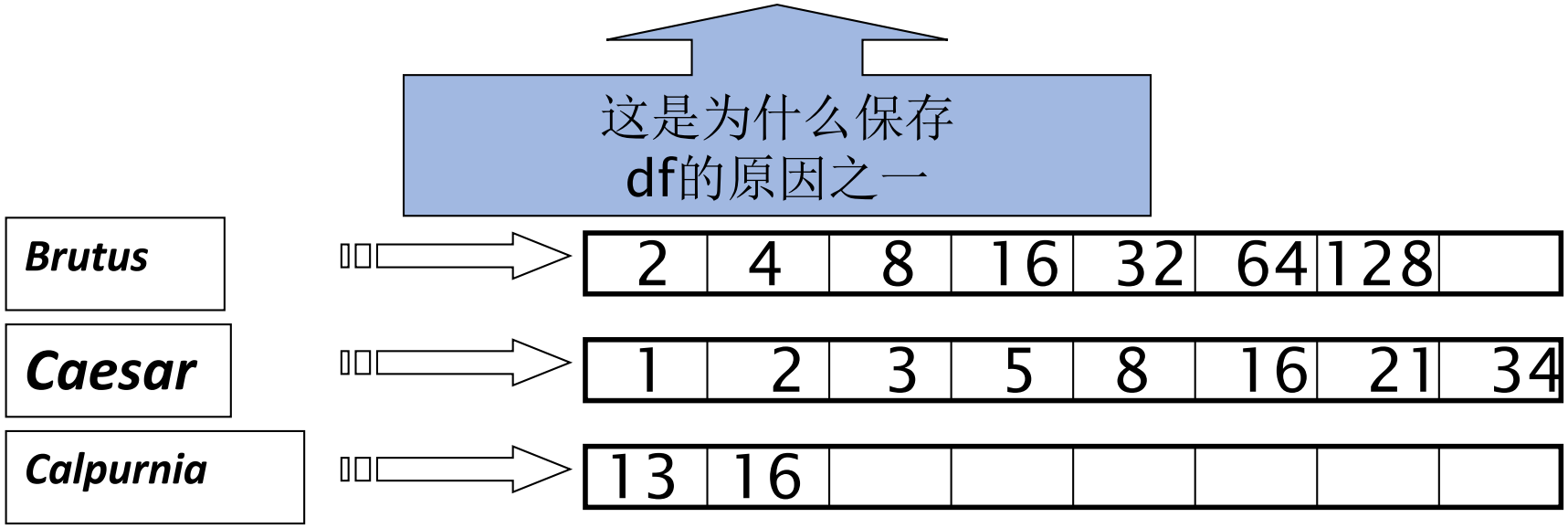
- 查询处理中是否存在处理的顺序问题？
- 考虑n 个词项的 AND
- 对每个词项，取出其倒排记录表，然后两两合并：1+3先来



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***.

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

布尔检索例子: 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 (1975年开始提供服务; 1992年加入排序功能)
- 几十T数据，700,000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
    - /3 = within 3 words, /S = in same sentence

布尔检索例子: 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2021年王牌对王牌的新闻，用布尔表达式怎么构造查询？
- (2021 or 第五季) AND (王牌 or 王牌对王牌)
- 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