

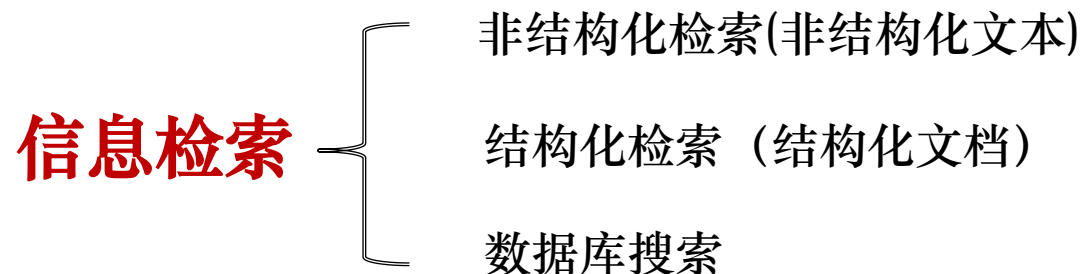
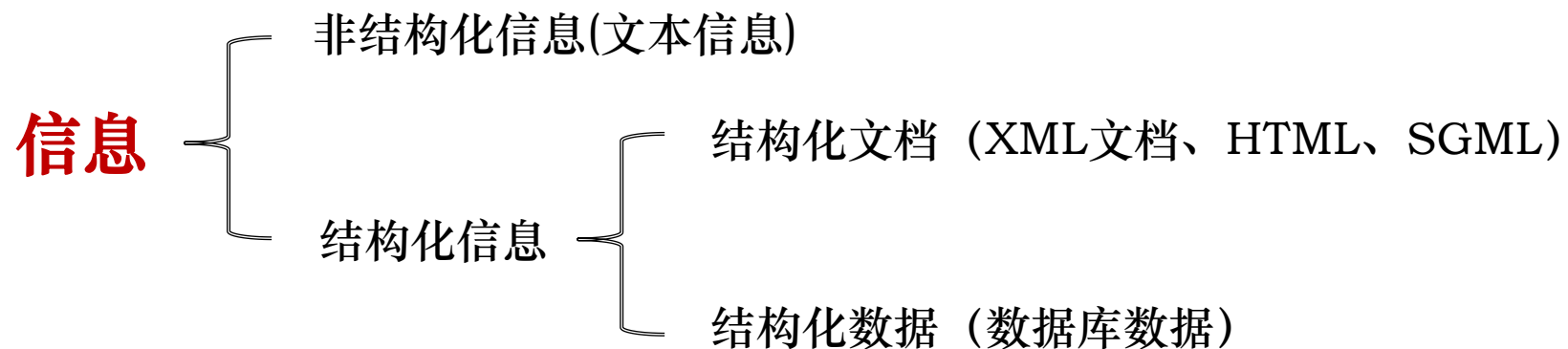
信息检索与Web搜索

第11讲 XML检索

XML Retrieval

授课人：高曙明

信息分类与检索分类



IR vs. 关系数据库

- 传统上说，IR 系统从无结构文本(指没有标记的“生”文本--“raw” text without markup)中返回信息
- RDB系统主要用于查询关系型数据(*relational data*)，即一系列记录集合，这些记录中包含预先定义的属性及属性值，如员工号、职位和工资

	RDB搜索	非结构化检索	结构化检索
对象	记录	非结构化文档	以文本为叶节点的树
模型	关系模型	向量空间或其他	?
主要数据结构	表格	倒排索引	?
查询语言	SQL查询	自由文本查询	?

结构化检索举例

结构化检索的应用场景

□ 数字图书馆、专利数据库、博客、包含已标注命名实体（如人名、地名）的文本

例子

- 数字图书馆: *give me a full-length article on fast fourier transforms*
- 专利: *give me patents whose claims mention RSA public key encryption and that cite US patent 4,405,829*
- 实体标记文本: *give me articles about sightseeing tours of the Vatican and the Coliseum*

RDB难以支持结构化检索

- ① 无序的DB系统可能返回大量文章，这些文章提到 Vatican、the Coliseum和sightseeing tours，但是并没有按照它们和查询的相关度排序
- ② 大部分用户都很难精确描述结构化限制条件。比如，用户可能并不知道搜索系统支持对哪些结构化元素的查询
tours AND (COUNTRY: Vatican OR LANDMARK: Coliseum)?
tours AND (STATE: Vatican OR BUILDING: Coliseum)?
- ③ 难以有效处理复杂文本属性

解决办法：将排序检索模型用于结构化文档搜索来解决上述问题

XML简介

- XML (Extensible Markup Language) : 是一个对结构化信息进行编码的标准
- XML文档: 以XML格式表示和存储的文本

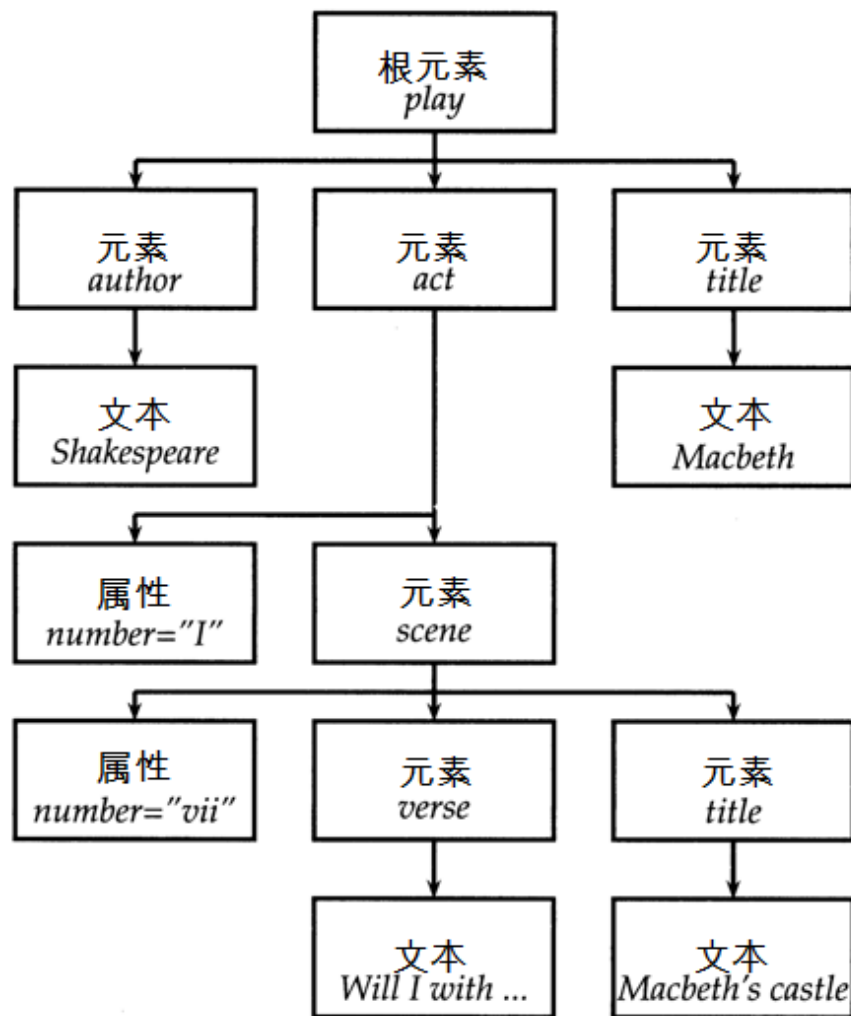
```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="1">
<scene number=""vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
...</verse>
</scene>
</act>
</play>
```

XML 文档的文档对象模型 (DOM)

- ❑ **DOM**(Document Object Model): 一个有序的带标记的树
- ❑ 树上的每个节点都是一个XML元素, 它由起始标签 (tag) 和结束标签来界定(如<title...>, </title...>)
- ❑ 一个XML元素可以有一个或多个XML属性 (如 `number`)
- ❑ 属性可以有属性值 (如 `vii`)
- ❑ 元素可以有子元素 (如 `title`, `verse`)

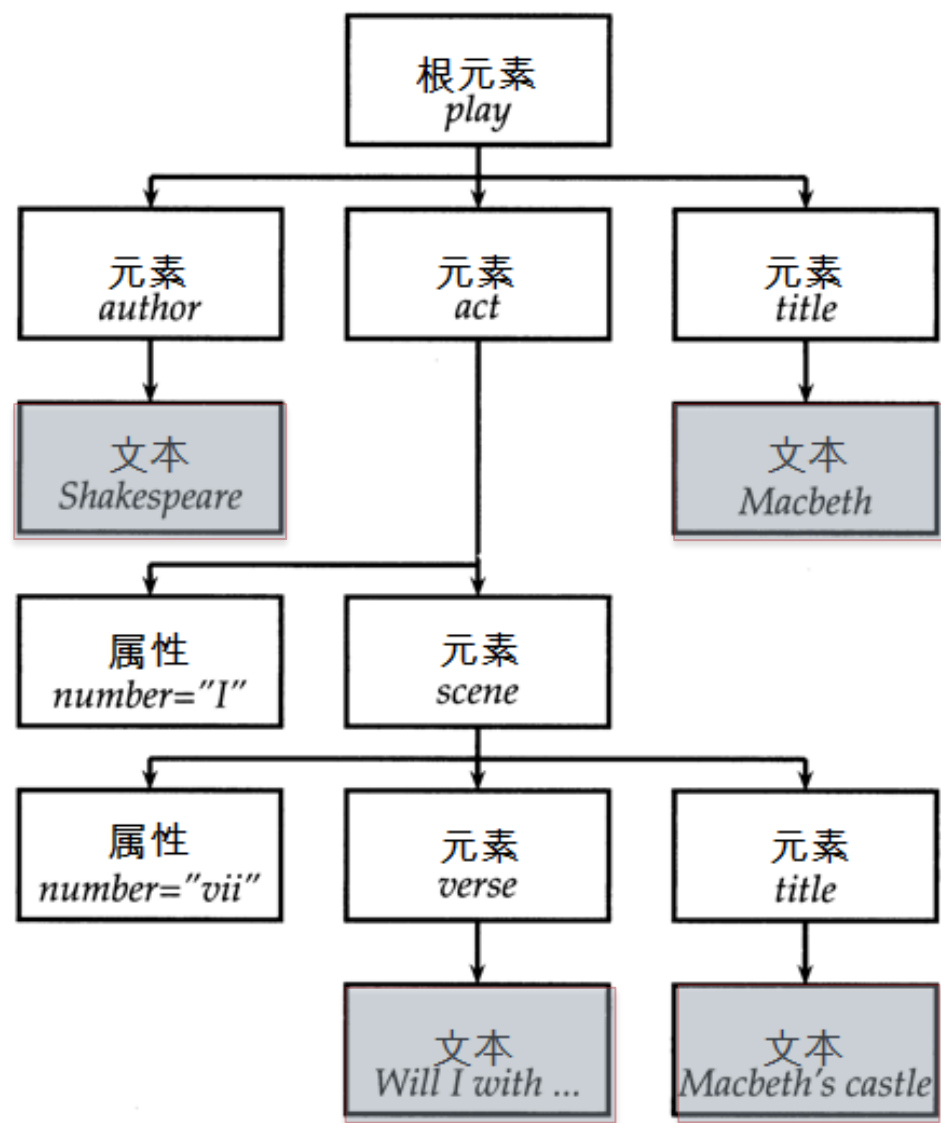
```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="1">
<scene number=""vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
...</verse>
</scene>
</act>
</play>
```

文档对象模型举例

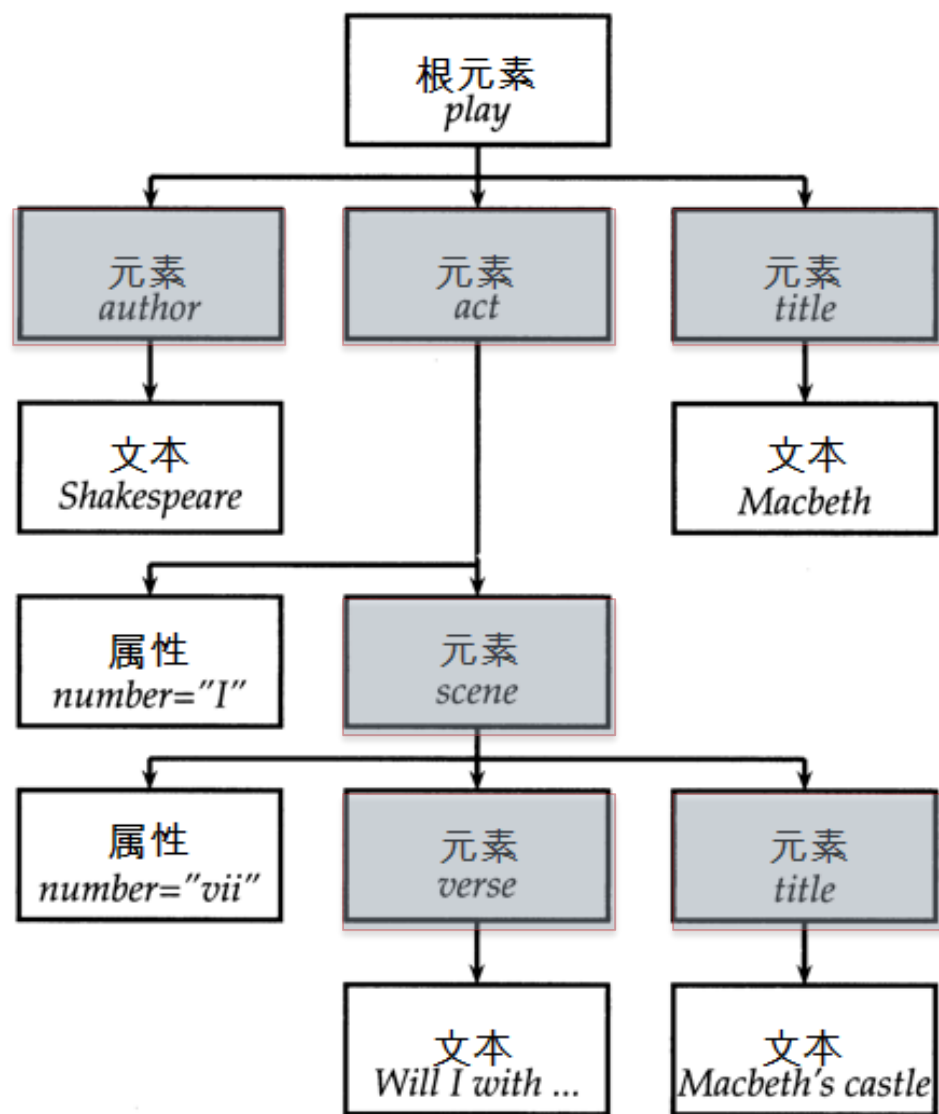


➤ 上述XML文档的
DOM

➤ 可以使用DOM
API对XML文档
进行处理



➤ XML 文档中的
叶节点主要由
文本构成



➤ XML文档中的
内部节点对文
档结构或元数
据进行编码

XML 中的 XPath 和 schema

□ XPath: XML文档集中的路径表达式描述标准

- 路径表达式也称为XML上下文或直接称上下文
比如: act/scene, play//scene

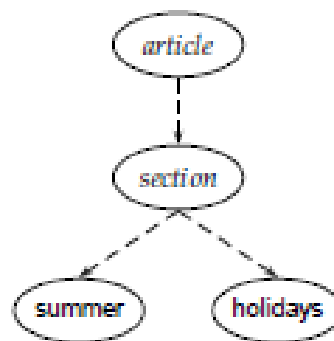
□ Schema: 定义了XML文档所允许的结构限制条件比如, 莎士比亚剧本的schema规定, 场 (scene) 只能以幕 (act) 的子节点方式出现。

- XML文档的两个schema标准分别是 XML DTD和 XML schema

XML查询

- ❑ XML查询的常用格式为NEXI (Narrowed Extended Xpath I)
- ❑ 例子

```
//article  
[.//yr = 2001 or .//yr = 2002]  
//section  
[about(.,summer holidays)]
```



► Figure 10.3 An XML query in NEXI format and its partial representation as a tree.

- ❑ 查询可以表示成树，能够反映结构需求

XML检索面临的挑战

□ **挑战1:** 希望返回文档的相关部分（即 XML元素），而非整个文档

例子

如果在《莎士比亚全集》中查找Macbeth's castle，那么到底应该返回场（scene）、幕（act）还是整个剧本呢？

- 上述情况下，用户可能在查找场(scene)
- 但是，另一个没有具体指定返回节点的查询
Macbeth，应该返回剧本的名称而不是某个子单位

□ **解决办法:** 结构化文档检索原则

挑战一的解决方案

结构化文档检索原则(structured document retrieval principle)

系统应该总是检索出回答查询的**最明确最具体**的文档部分

- **检索策略之一**：返回包含信息需求的最小单位
- **原则的算法实现很困难**：比如查询： title:Macbeth， 整个剧本的标题 Macbeth以及第一幕第六场的标题Macbeth' s castle都包含匹配词项 Macbeth
 - 在这个例子中，剧本的标题这个位于更高层的节点作为答案却更合适
 - **确定查询应答的正确层次非常困难**

XML检索面临的挑战

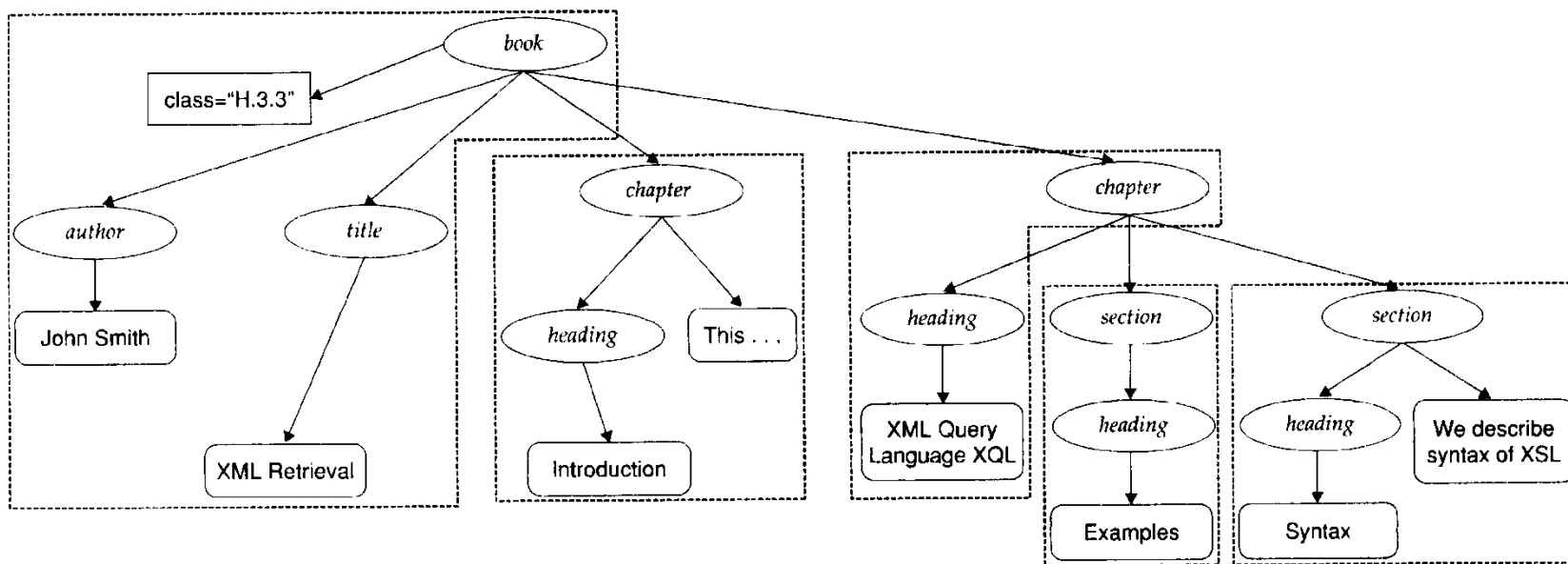
□ 挑战2: 如何确定文档的索引单位

在结构化文档检索中, 存在定义索引单位的多种方法:

- ① 将节点分组, 形成多个互不重叠的伪文档
- ② 索引最大元素, 然后自顶向下(top down)进行后处理
- ③ 索引叶节点, 然后自底向上(bottom up)进行后处理
- ④ 对所有元素建立索引

XML索引单位：不重叠伪文档

□ **基本思想：** 将节点分组，形成多个互不重叠的伪文档



索引单位： *book*、*chapter*、*section*等没有重叠的部分

缺点： 这些伪文档的内容不连贯，它们对用户而言可能没有意义

XML索引单位：最大元素

- **基本思想：**使用XML文档中最大的一个元素作为索引单位
比如，在一个书的集合中以元素book为索引单位
- **检索结果的后处理：**从最大元素开始自顶向下找到更适合作为答案的子元素
- **优点：**索引规模最小
- **缺点：**往往并不能返回最佳匹配子元素，这是因为子元素很相关未必其根元素相关

XML索引单位：叶节点

- **基本思想：**直接对所有叶节点进行索引
- **检索结果的后处理：**从相关叶节点开始，自底向上扩展成更合适的单位
- **优点：**索引规模较小，不会漏掉最相关的叶节点
- **缺点：**上层元素很相关未必其叶节点很相关

XML索引单位：所有元素

- **优点：**索引所有元素限制最少，能够保证召回率
- **缺点：**很多XML元素并不是有意义的搜索结果，比如，排版相关的元素（如definitely）或者不能脱离上下文进行单独解释的ISBN书号等；搜索结果会存在高度冗余

例子

查询Macbeth's castle，我们会返回从根节点到Macbeth's castle路径上的所有play、act、scene和title元素。此时，叶节点在结果集中会出现4次，其中1次是作为索引对象直接出现的，而其他3次是作为其他元素的一部分出现的

造成冗余的重要原因：元素嵌套

XML检索面临的挑战

- **挑战3:** 元素嵌套问题
- **解决方案:** 基于启发式规则, 对返回元素进行限制:
 - 忽略所有的小元素
 - 忽略用户不会浏览的所有元素类型
 - 忽略通常被评估者判定为不相关性的元素类型
 - 只保留系统设计人员或图书馆员认定为有用的检索结果所对应的元素类型
- 经过上述处理的结果集中仍然包含嵌套元素

元素嵌套的进一步处理

- 可以通过一个后处理过程来去掉某些元素，从而降低检索结果的冗余性
- 在结果列表中将多个嵌套元素折叠起来，并通过对查询词项进行高亮显示 (highlighting) 来吸引用户关注相关段落

折叠+高亮显示

- 好处 1: 允许用户扫描中等规模的元素 (如 section)，因此，如果 section 和 paragraph 都出现在结果列表中，那么显示 section 就足够了
- 好处2: paragraph 会和它的上下文 (包含该 paragraph 的 section) 在一起展示。即使 paragraph 本身就可以满足查询的需求，这种上下文仍然对于解释该 paragraph 很有帮助

嵌套元素和词项统计信息

与嵌套相关的另一个问题：在计算用于排序的词项统计信息(特别是idf)时，需要区别词项的不同上下文

例子

Gates出现在author节点下与其出现在内容元素中（如section中，此时代表的是gate的复数）毫无关系。于是，在这个例子中，为Gates计算一个单独的文档频率df意义不大。

解决办法：为每一个XML上下文-词项对计算idf

- 数据稀疏问题（许多上下文-词项对出现过少，从而导致对文档频率估计的可靠性不足）
- 一个折中方案是在区分上下文时只考虑词项的父节点 x ，而不考虑从根节点到 x 路径上的其他部分

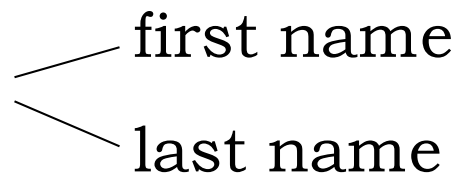
XML检索面临的挑战

□ **挑战4:** schema异构性

□ **问题:**

- 对等的元素具有不同的名称，如：creator、author

- 结构化组织方式不同，如：作者—姓名

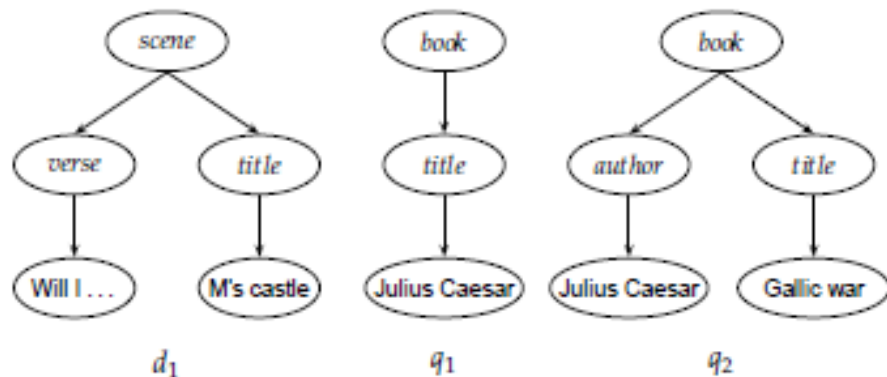
作者 

- 导致匹配效果变差

□ **解决方案:** 对元素名称进行扩展，放松结构化限制条件，比如将父子关系 → 后裔关系

基于向量空间模型的XML检索

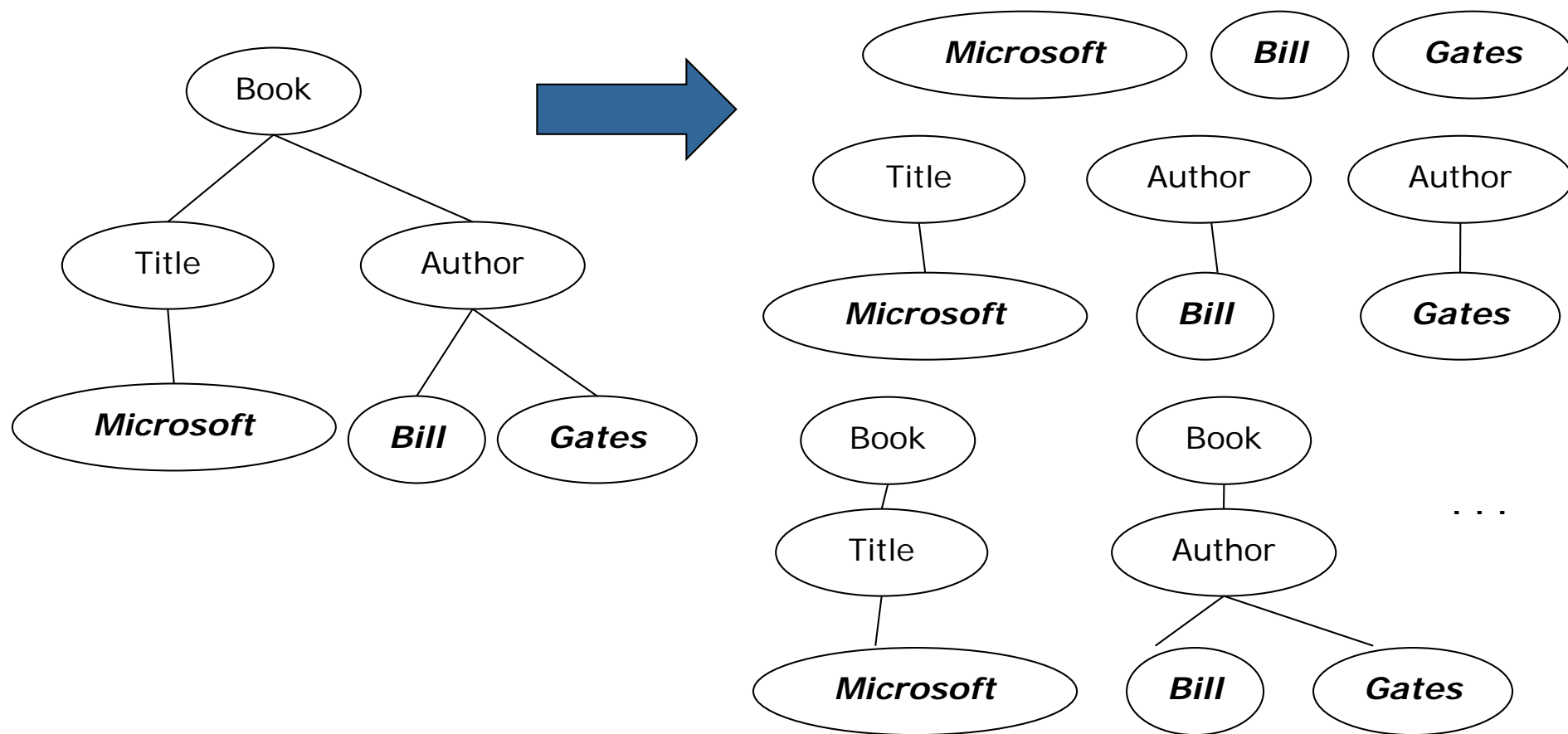
- **目标:** 对基于词项的向量空间模型进行改进, 使其能够有效地支持XML文档检索
- 为什么基于词项的VSM模型需要改进? 其中没有考虑结构信息, 会导致检索效果不佳, 比如: 我们希望书Julius Caesar与 q_1 匹配, 与 q_2 不匹配
- **解决方案:** 对向量空间中的每一维都同时考虑词项及其在XML文档中的上下文



基于向量空间模型的XML检索

- **词汇化子树：** 是XML文档DOM模型中的至少包含词汇表中一个词项的一棵子树
 - 这里，DOM中包含多词的每个文本节点（叶节点）被 分裂成多个节点，每个节点对应一个词。 例如，将 Bill Gates 分裂成 Bill 和Gates
- **词汇化子树空间：** 向量空间的每一维定义为文档的词汇化子树
- **基于VSM的XML检索：** 将查询和文档表示成这些词汇化子树空间上的向量，并根据前面的向量相似度公式进行相似度计算

词汇化子树举例



向量空间维度和检索精度之间的平衡

□ 向量空间的可能维度

- 如果每棵词汇化子树都对应空间一维，那么空间的维数会变得太大
- 如果将每一维限制为词汇表中的词项，那么得到的是一个标准的向量空间检索系统。这种系统下得到的很多文档在结构上并不与查询匹配 (例如，title中的Gates和author中的Gates)

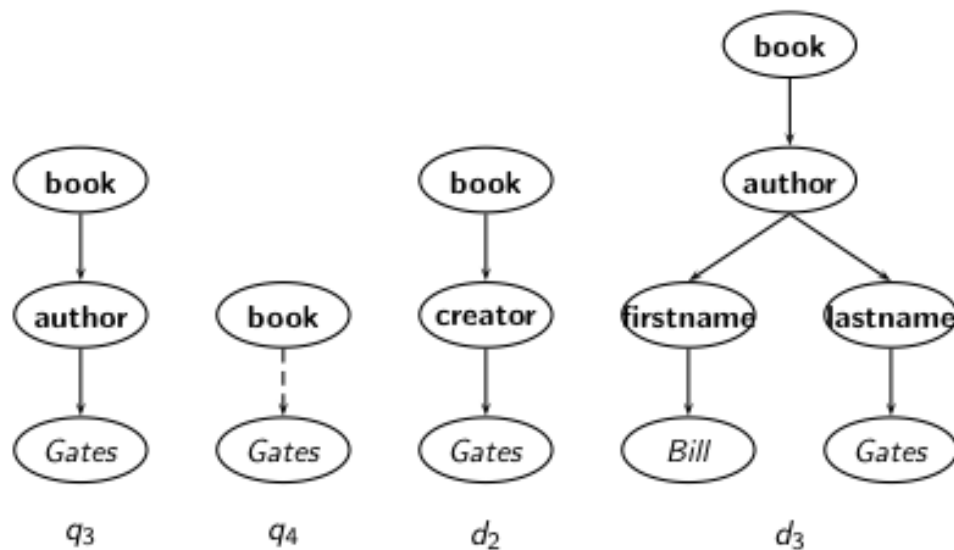
□ **解决方案:** 对所有的最终以单个词项结束的路径建立索引，即对所有XML上下文/词项对建立索引。这种XML上下文/词项对被称为结构化词项 (structural term)，记为 $\langle c, t \rangle$: 其中c是XML上下文，t是词项

上下文相似度 (Context resemblance)

$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{如果 } c_q \text{ 和 } c_d \text{ 匹配} \\ 0 & \text{如果 } c_q \text{ 和 } c_d \text{ 不匹配} \end{cases}$$

其中 $|c_q|$ 和 $|c_d|$ 分别是查询路径和文档路径中的节点数目，并且当且仅当可以通过插入额外的节点使 c_q 转换成 c_d 时， c_q 和 c_d 才能匹配。

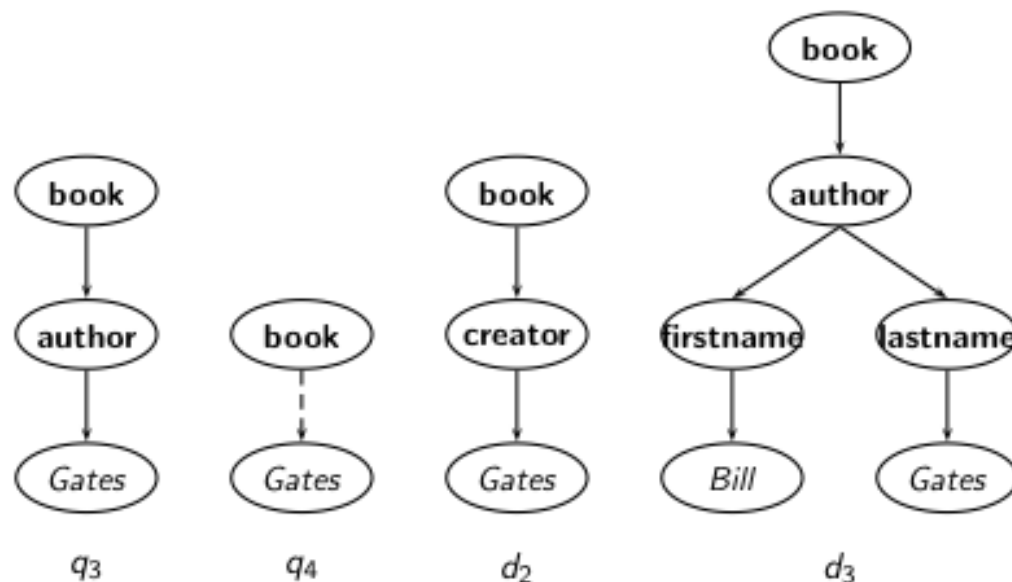
上下文相似度计算的例子



$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{如果 } c_q \text{ 和 } c_d \text{ 匹配} \\ 0 & \text{如果 } c_q \text{ 和 } c_d \text{ 不匹配} \end{cases}$$

$CR(c_{q4}, c_{d2}) = 3/4 = 0.75$. 如果 q 和 d 相等, 那么 $CR(c_q, c_d) = 1.0$

上下文相似度计算的例子



$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{如果 } c_q \text{ 和 } c_d \text{ 匹配} \\ 0 & \text{如果 } c_q \text{ 和 } c_d \text{ 不匹配} \end{cases}$$

$$CR(c_{q4}, c_{d3}) = 3/5 = 0.6.$$

文档相似度计算方法

$$\text{SIMNOMERGE}(q, d) =$$

$$\sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q, t, c_k) \frac{\text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}}$$

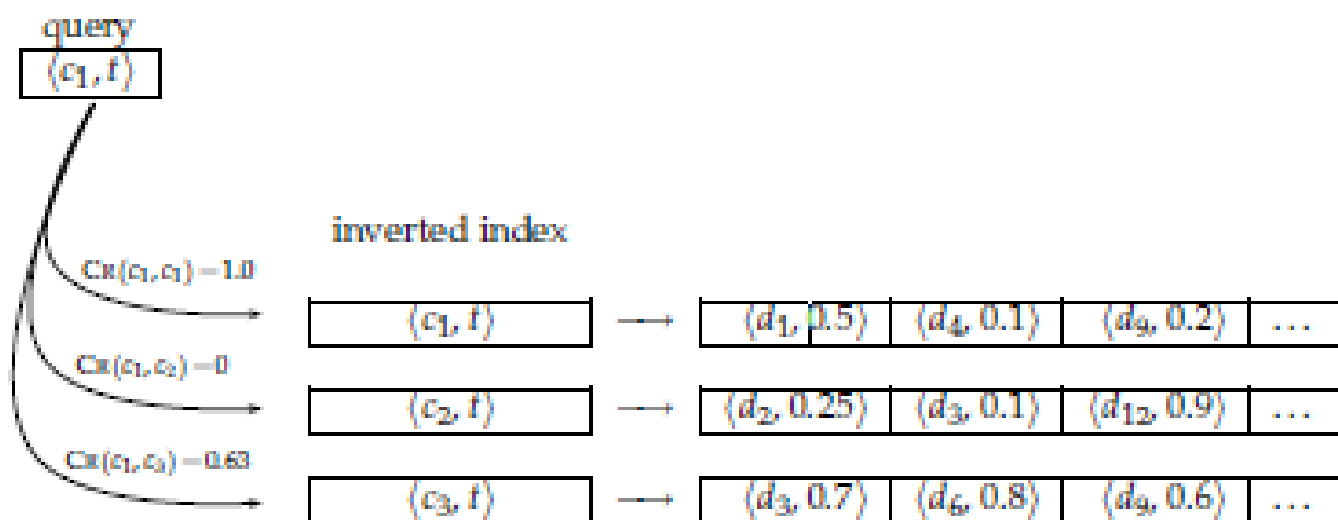
- V 非结构化词项的词汇表， B 是所有XML上下文的集合
- $\text{weight}(q, t, c)$ 和 $\text{weight}(d, t, c)$ 分别是词项 t 在查询 q 和文档 d 的上下文 c 中的权重 (可以采用 $\text{idf}_t \times \text{wf}_{t,d}$, 逆文档频率 idf_t 的值取决于 df_t 计算时我们所利用的元素)
- $\text{SIMNOMERGE}(q, d)$ 并不是一个真正的余弦相似度计算函数，因为它的值可能会超过1.0

SIMNOMERGE 算法的伪代码

SCOREDOCUMENTSWITHSIMNOMERGE($q, B, V, N, \text{normalizer}$)

```
1  for  $n \leftarrow 1$  to  $N$ 
2  do  $\text{score}[n] \leftarrow 0$ 
3  for each  $\langle c_q, t \rangle \in q$ 
4  do  $w_q \leftarrow \text{WEIGHT}(q, t, c_q)$ 
5    for each  $c \in B$ 
6    do if  $\text{CR}(c_q, c) > 0$ 
7      then  $\text{postings} \leftarrow \text{GETPOSTINGS}(\langle c, t \rangle)$ 
8      for each  $\text{posting} \in \text{postings}$ 
9      do  $x \leftarrow \text{CR}(c_q, c) * w_q * \text{weight}(\text{posting})$ 
10       $\text{score}[\text{docID}(\text{posting})] + = x$ 
11 for  $n \leftarrow 1$  to  $N$ 
12 do  $\text{score}[n] \leftarrow \text{score}[n] / \text{normalizer}[n]$ 
13 return  $\text{score}$ 
```


文档相似度计算举例



在本例中，排名最高的文档是 d_9 ，其相似度为
 $1.0 * 0.2 + 0.63 * 0.6 = 0.578$

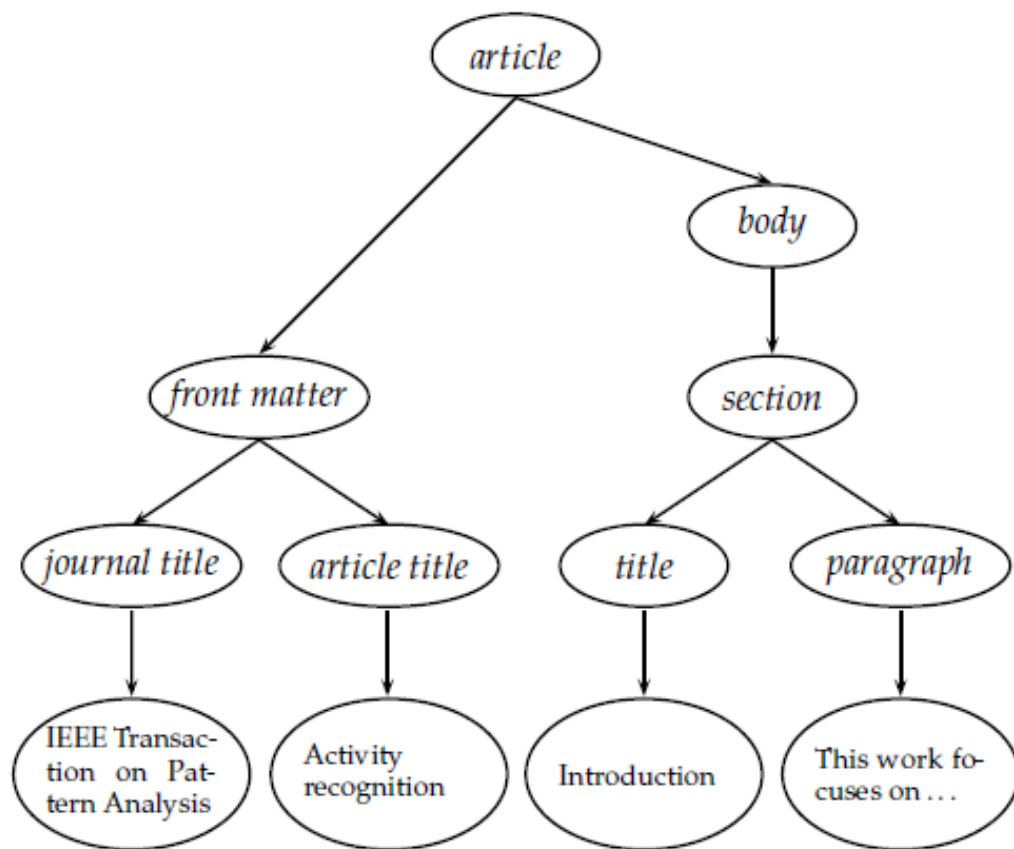
XML检索评价平台：INEX

- **INEX:** XML检索研究中的首要评测平台和会议，它通过协作产生参考文档集、查询集及相关性判断。INEX 2002文档集包含大概12000篇来自IEEE期刊的文章。
- 文档的相关性判定主要通过人工判断来完成

INEX 2002 文档集统计信息

12,107	文档数目
494 MB	规模
1995—2002	文章发表年份
1,532	平均每篇文档中的XML节点个数
6.9	平均每个节点的深度
30	CAS主题的数目
30	CO主题的数目

INEX 文档 schema



INEX 主题

- 完全基于内容(content-only 或 CO)的主题:和非结构化信息检索中一样的常规关键词查询
- 内容结构相结合 (content-and-structure 或 CAS)的主题: CAS主题在关键词基础上增加了结构化限制

由于CAS查询同时包含结构信息和内容信息，其相关性判断就比非结构化中的相关性判断要复杂得多

INEX相关性判断

部件覆盖度

评价的是返回元素在结构上是否正确，分为四个等级：

- ① **精确覆盖 (E)**：所需求的信息是部件的主要主题，并且该部件是一个有意义的信息单位
- ② **覆盖度太小 (S)**：所需求的信息是部件的主要主题，但是该部件不是一个有意义（自包含）的信息单位
- ③ **覆盖度太大 (L)**：所需求的信息在部件中，但不是主要主题
- ④ **无覆盖 (N)**：所需求的信息不是部件的主题

INEX相关性判断

□ **主题相关性：** 强相关 (3)、较相关 (2)、弱相关 (1) 和不相关 (0)

部件覆盖度和主题相关性的组合

- 每个部件在覆盖度和主题相关性两个方面都要进行判断，然后将判断结果组合成一个数字—字母编码，2S表示一个比较相关的部件，但是其覆盖度太小。而3E表示高度相关并具有精确覆盖的一个部件。
- 理论上说，经过组合，对一个部件的评价有16种可能，但是实际评价中很多组合并不会出现。
- 比如，编码为3N的组合显然是不可能的。

INEX 相关性判断

- 相关度—覆盖度组合可以采用如下量化方法:

$$Q(rel, cov) = \begin{cases} 1.00 & \text{if } (rel, cov) = 3E \\ 0.75 & \text{if } (rel, cov) \in \{2E, 3L\} \\ 0.50 & \text{if } (rel, cov) \in \{1E, 2L, 2S\} \\ 0.25 & \text{if } (rel, cov) \in \{1S, 1L\} \\ 0.00 & \text{if } (rel, cov) = 0N \end{cases}$$

- 检索结果集合A中相关部件的数目可以定义为:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} Q(rel(c), cov(c))$$

INEX的评价指标

- 非结构化IR中定义的正确率、召回率及 F 值的标准定义都可以近似地应用到上式所示的相关部件数目上来。和前面的细微差别是，这里计算的是评分等级之和而不是二值相关性之和

缺点

没有考虑到重合现象，搜索结果中的元素的多重嵌套问题加剧了这一点

近年来INEX的焦点：提出结果无冗余的检索算法和评估指标并对结果进行恰当评价

参考资料

- ❑ 《信息检索导论》第10章
- ❑ Amer-Yahia, Sihem, and Mounia Lalmas. 2006. XML search: Languages, INEX and scoring. *SIGMOD Record* 35(4):16-23. DOI: <http://doi.acm.org/10.1145/1228268.1228271>
- ❑ Harold, Elliotte Rusty, and Scott W. Means. 2004. *XML in a Nutshell*, 3rd edition. O' Reilly
- ❑ INEX网站: <http://www.inex.otago.ac.nz/>

课后作业

□ 见课程网页:

<http://10.76.3.31>