CS3245

# Information Retrieval

10

Lecture 10: XML Retrieval

# Last Time

- Evaluating a search engine
  - Benchmarks: 3 components
    Queries, documents and relevance judgments
  - Precision-recall curves
  - Composite, single number summaries
  - A/B Testing
- Creating results summaries
  - Static and dynamic versions

# Today

- Introduction
- Basic XML concepts
- Challenges in XML IR
- Vector space model for XML IR
- Evaluation of XML IR

# IR and relational databases

- IR systems often contrasted with relational databases (RDB).
- Traditionally, IR systems retrieve information from *unstructured text* ("raw" text without markup).
- RDB systems are used for querying *relational data*: sets of records that have values for predefined attributes such as employee number, title and salary.

| | RDB search | unstructured IR |
|---|---|---|
| objects | records | unstructured docs |
| main data structure | table | inverted index |
| model | relational model | vector space & others |
| queries | SQL | free text queries |

- Some structured data sources containing text are best modeled as structured documents rather than relational data (Structured retrieval).

# Structured retrieval

Premise: queries are structured or unstructured; documents are structured.

## Applications of structured retrieval

Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging)

## Example

- Digital libraries: *give me a full-length article on fast fourier transforms*
- Patents: *give me patens whose claims mention RSA public key encryption and that cite US patent 4,405,829*
- Entity-tagged text: *give me articles about sightseeing tours of the Vatican and the Coliseum*

# Why RDB is not suitable in this case

Three main problems

1. An unranked system (like a DB) can return a large set leading to information overload
2. Users often don't precisely state structural constraints – may not know possible structure elements are supported
   - *tours* AND (COUNTRY: *Vatican* OR
   - LANDMARK: *Coliseum*)?
   - t*ours* AND (STATE: *Vatican* OR BUILDING: *Coliseum*)?
3. Users may be unfamiliar with structured search and the necessary advanced search interfaces or syntax

**Solution**: adapt ranked retrieval to structured documents

# Structured Retrieval

## RDB search, Unstructured IR, Structured IR

| | RDB search | unstructured retrieval | structured retrieval |
|---|---|---|---|
| objects | records | unstructured docs | trees with text at leaves |
| main data structure | table | inverted index | ? |
| model | relational model | vector space & others | ? |
| queries | SQL | free text queries | ? |

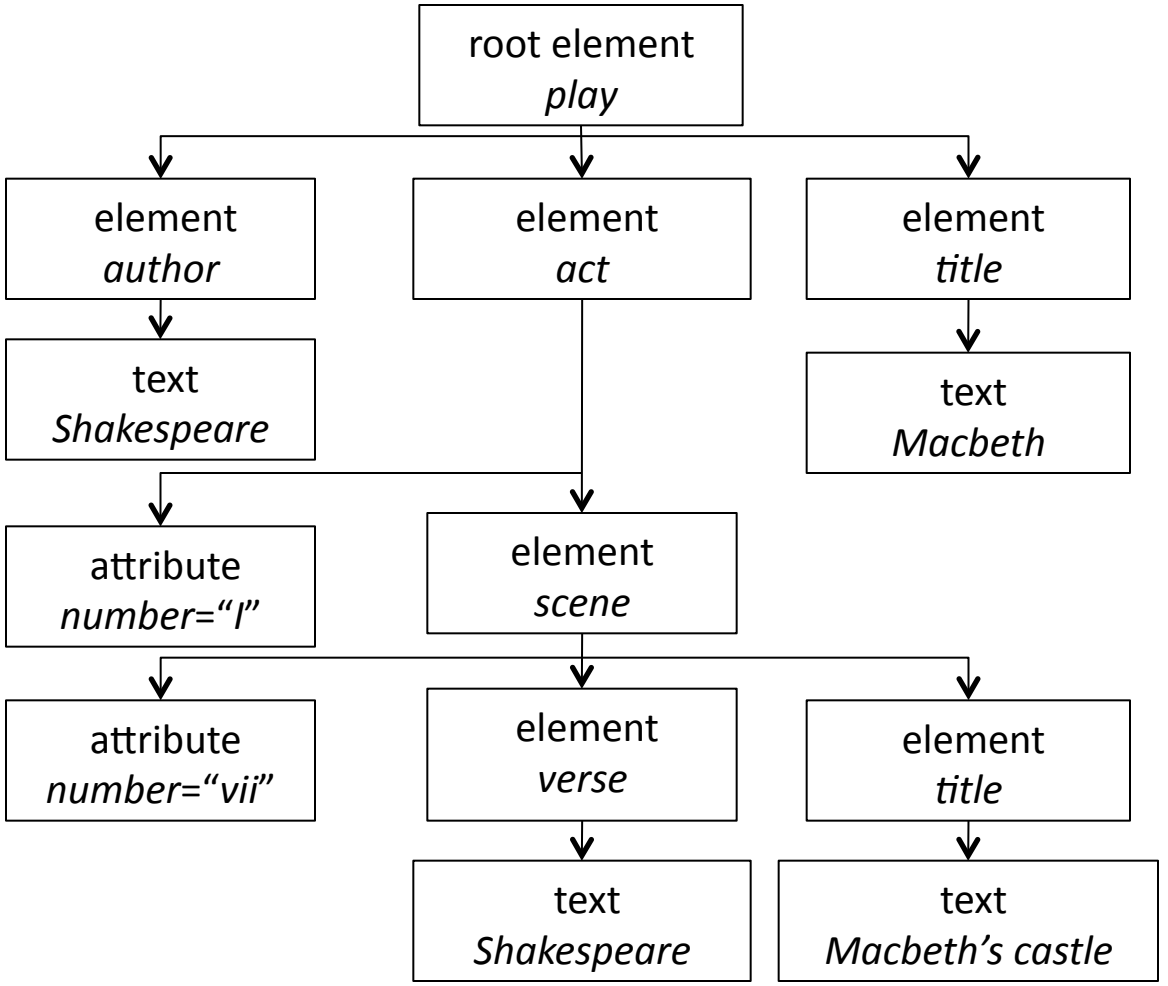
- Standard for encoding structured documents: Extensible Markup Language (XML)
  - structured IR ➔ XML IR
  - also applicable to other types of markup (HTML, SGML, …)

# XML Documents

- Ordered, labeled tree
- Each node of the tree is an XML element, written with an opening and closing XML tag (e.g. <title…>, </title…>)
- An element can have one or more XML attributes (e.g. number)
- Attributes can have values (e.g. vii)
- Attributes can have child elements (e.g. title, verse)

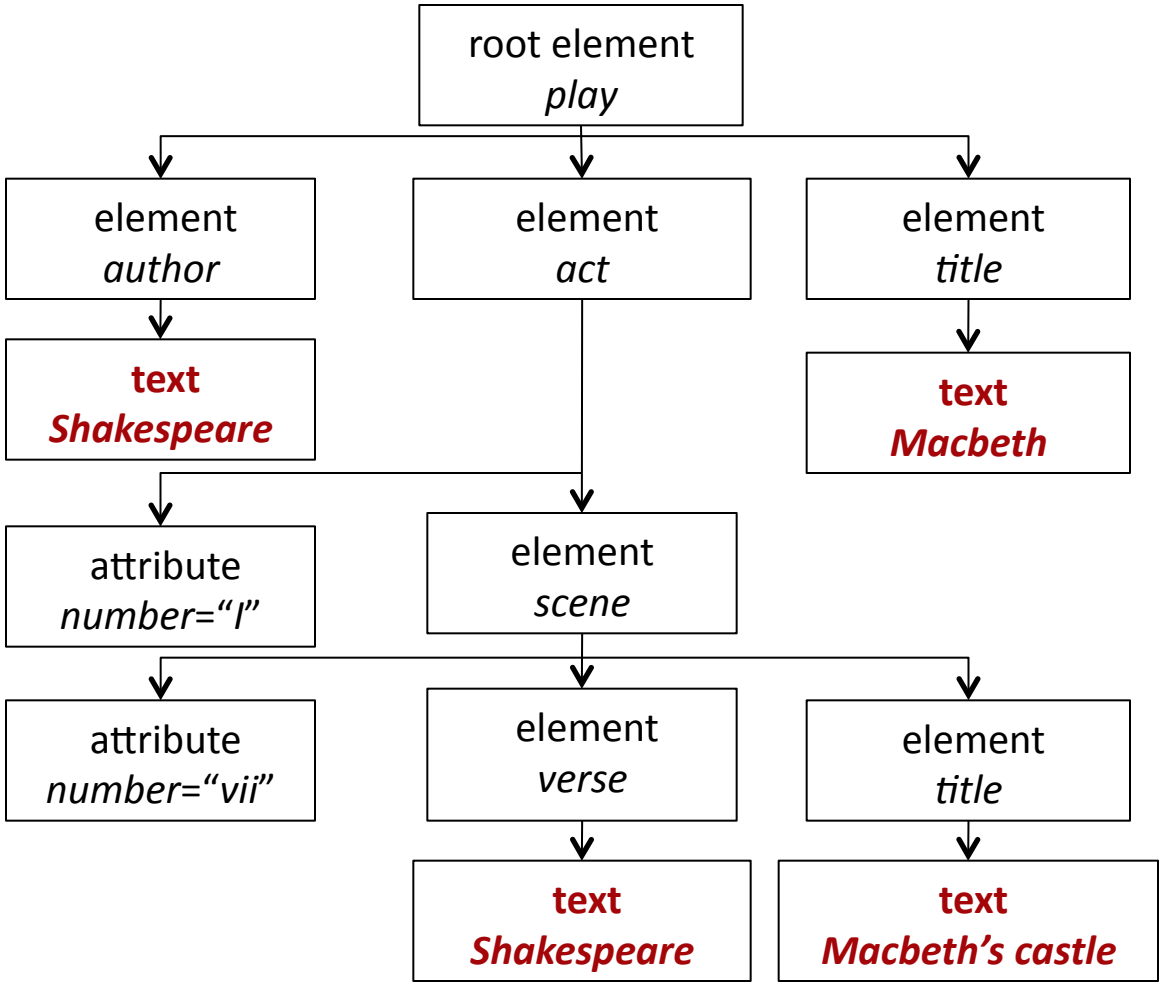
```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene  number=""vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
…</verse>
</scene>
</act>
</play>
```

# XML Document



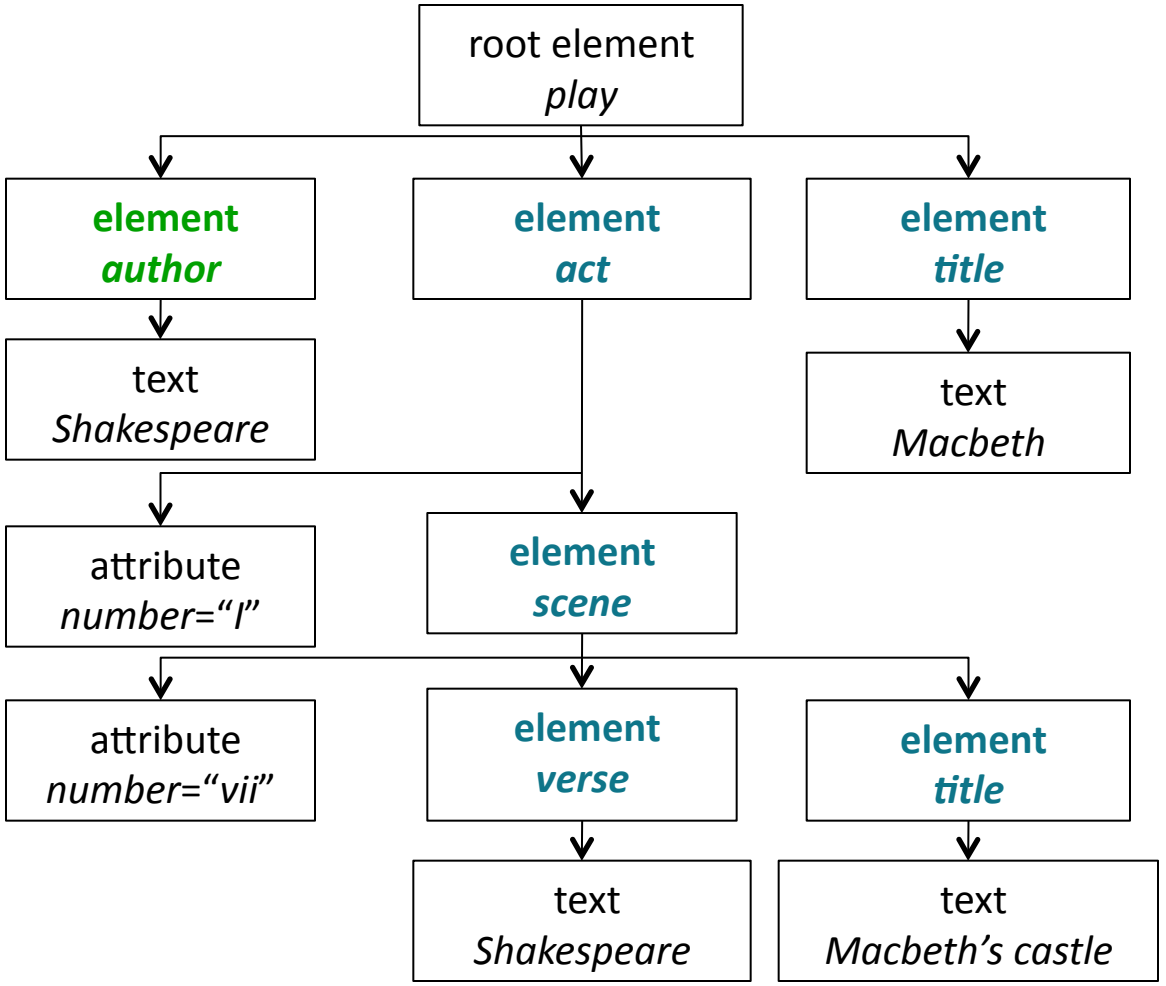
# XML Document

The **leaf nodes** consist of text



# XML Document

The internal nodes encode **document structure** or **metadata** functions



# XML Basics and Definitions

- **XML Document Object Model (XML DOM)**: standard for accessing and processing XML documents
  - The DOM represents elements, attributes and text within elements as nodes in a tree.
  - With a DOM API, we can process an XML documents by starting at the root element and then descending down the tree from parents to children.
- **XPath**: standard for enumerating path in an XML document collection.
  - We will also refer to paths as **XML contexts** or simply **contexts**
- **Schema**: puts constraints on the structure of allowable XML documents. E.g. a schema for Shakespeare's plays: scenes can occur as children of acts.
  - Two standards for schemas for XML documents are: XML DTD (document type definition) and XML Schema.

# CHALLENGES IN XML RETRIEVAL

# First challenge: Document parts to retrieve

- Structured or XML retrieval: users want parts of documents (i.e., XML elements), not the entire thing.

**Example**

If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?

  - In this case, the user is probably looking for the scene.
  - However, an otherwise unspecified search for *Macbeth* should return the play of this name, not a subunit.

- Solution: structured document retrieval principle

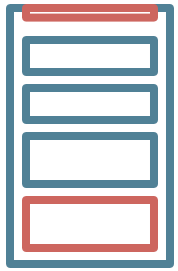
# Structured document retrieval principle



## Structured document retrieval principle

***A system should always retrieve the most specific part of a document answering the query.***

- Motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level.
- Hard to implement this principle algorithmically. E.g. query: title:*Macbeth* can match both the title of the tragedy, *Macbeth*, and the title of Act I, Scene vii, *Macbeth's castle*.
  - In this case, the title (higher node) is preferred.
  - Difficult to decide which level of the tree best satisfies the query.



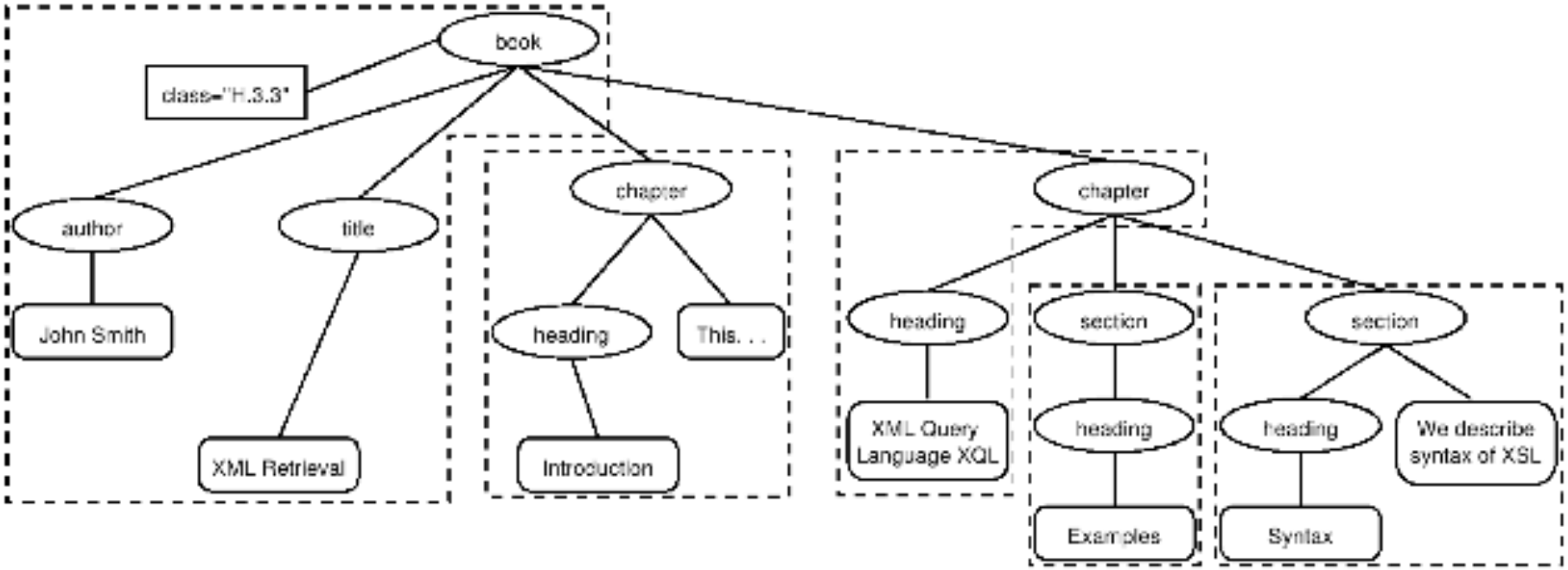
# Second challenge: Document part to index



- Central notion for indexing and ranking in IR: documents unit or **indexing unit**.
  - In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages, etc.
  - In structured retrieval, there are four main different approaches to defining the indexing unit:
    1. Non-overlapping pseudo-documents
    2. Top down
    3. Bottom up
    4. All units

# 1) Non-overlapping pseudodocuments

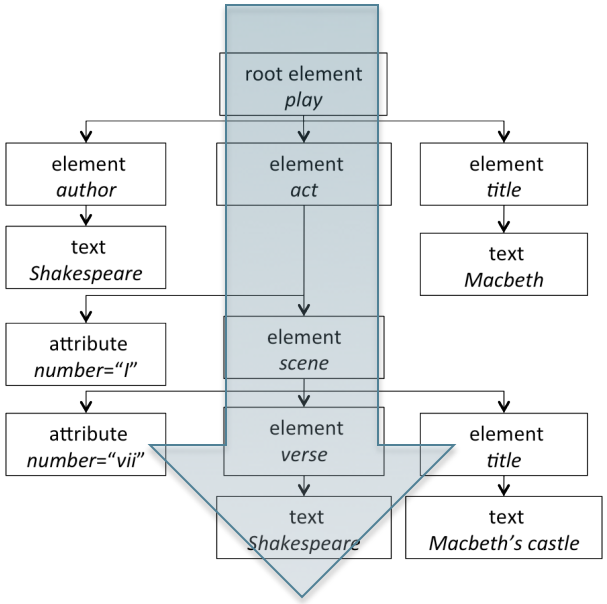
Group nodes into non-overlapping subtrees




- Indexing units: books, chapters, section, but without overlap.
- Disadvantage: pseudodocuments may not make sense to the user because they are not coherent units.

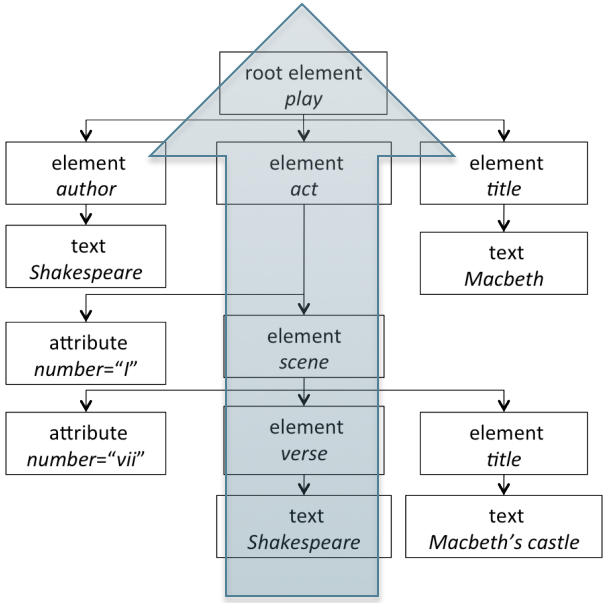
# 2) Top down

- A 2-stage process:
  1. Start with one of the largest elements as the indexing unit, e.g. the **<book>** element in a collection of books
  2. Then postprocess search results to find for each book the subelement that is the best hit.
- This two-stage process often fails to return the best subelement
  - the relevance of a whole book is often not a good predictor of the relevance of subelements within it.



# 3) Bottom Up

- We can search all leaves, select the most relevant ones and then extend them to larger units in postprocessing (bottom up).
- Similar problem as top down: the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in.



# 4) Index all elements

- The least restrictive approach, but also problematic:
  - Many XML elements are not meaningful search results, e.g., an ISBN number, bolded text
  - Indexing all elements means that search results will be highly redundant.

**Example**

For the query *Macbeth's castle* we would return all of the *play*, *act*, *scene* and *title* elements on the path between the root node and *Macbeth's castle*. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

- **Nested elements are** elements that are contained within each other. Returning redundant nested elements is not very user-friendly.

# Third challenge: Nested elements



- Due to the redundancy of nested elements, it is common to restrict the set of elements eligible for retrieval.

- Restriction strategies include:
  - discard all small elements
  - discard all elements that **users** do not look at (working XML retrieval system logs)
  - discard all elements that **assessors** generally do not judge to be relevant (when relevance assessments are available)
  - only keep elements that a **system designer or librarian** has deemed to be useful

- In most of these approaches, result sets will still contain nested elements.

# Third challenge: Nested elements



Further techniques:

- remove nested elements in a postprocessing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

## Highlighting

- Gain 1: enables users to scan medium-sized elements (e.g., a section); thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.
- Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

# Nested elements and term statistics

- Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf ).

**Example**

The term *Gates* under the node *author* is unrelated to an occurrence under a content node like *section* if used to refer to the plural of *gate*. It makes little sense to compute a single document frequency for *Gates* in this example.

- Solution: compute idf for XML-context term pairs.
- Sparse data problems (many XML-context pairs occur too rarely to reliably estimate df)
- Compromise: consider the parent node **x** of the term and not the rest of the path from the root to **x** to distinguish contexts.

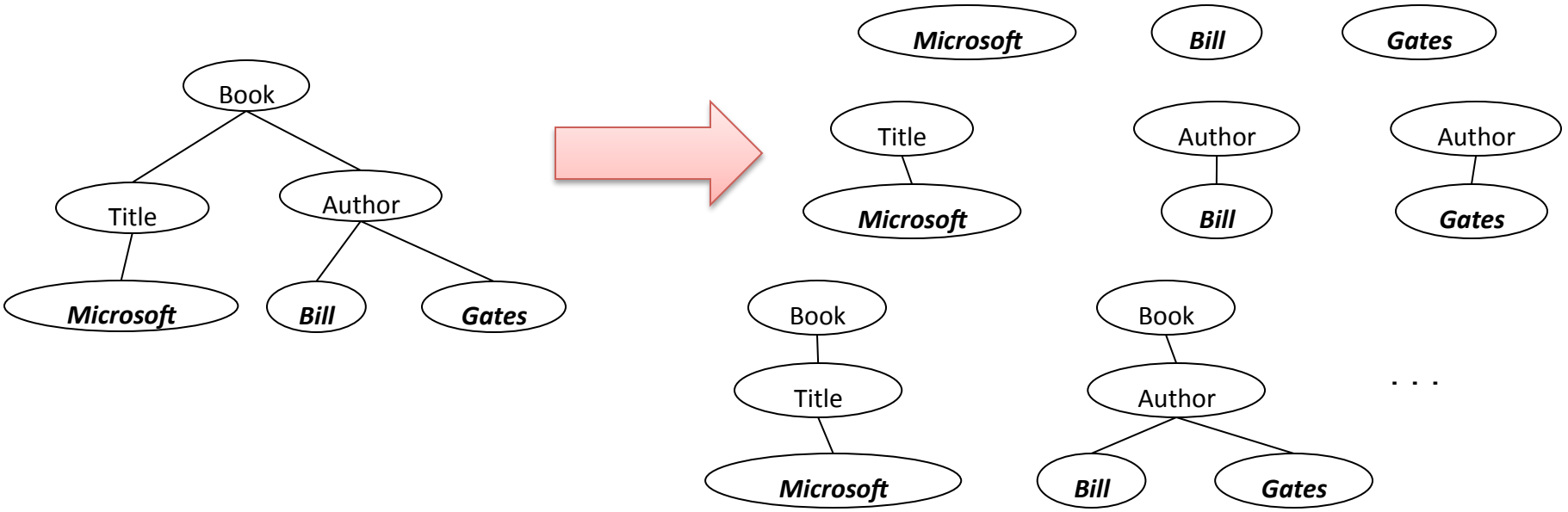
NUS National University of Singapore

# VECTOR SPACE MODEL FOR XML IR

# Main idea: lexicalized subtrees

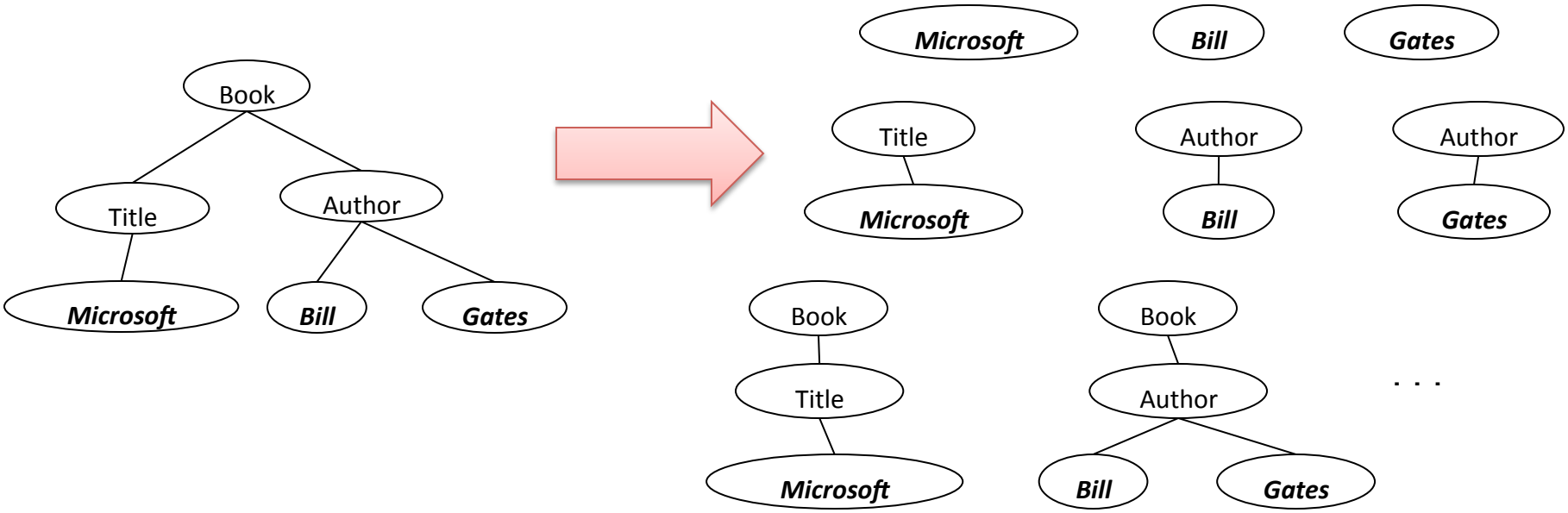
- Aim: to have each dimension of the vector space encode a word together with its position within the XML tree.
- How: Map XML documents to lexicalized subtrees.

"With words"



# Creating lexicalized subtrees

- Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split **Bill Gates** into **Bill** and **Gates**
- Define the dimensions of the vector space to be lexicalized subtrees of documents – subtrees that contain at least one vocabulary term.



# Lexicalized subtrees

- We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them,
- e.g. using the vector space formalism.

## Vector space formalism in unstructured vs. structured IR

The **main difference** is that the dimensions of vector space in unstructured retrieval are **vocabulary terms** whereas they are **lexicalized subtrees** in XML retrieval.

# Structural term

- There is a tradeoff between the dimensionality of the space and the accuracy of query results.
  - Feast or Famine
    - If we restrict dimensions to vocabulary terms, then the VSM retrieval system will retrieve many documents that do not match the structure of the query (e.g., Gates in the title as opposed to the author element).
    - If we create a separate dimension for each lexicalized subtree in the collection, the dimensionality becomes too large.
- Compromise: index all paths that end in a single vocabulary term (i.e., all XML-context term pairs). We call such an XML-context term pair a structural term and denote it by **<c, t>**: a pair of XML-context **c** and vocabulary term **t**.

# Context resemblance

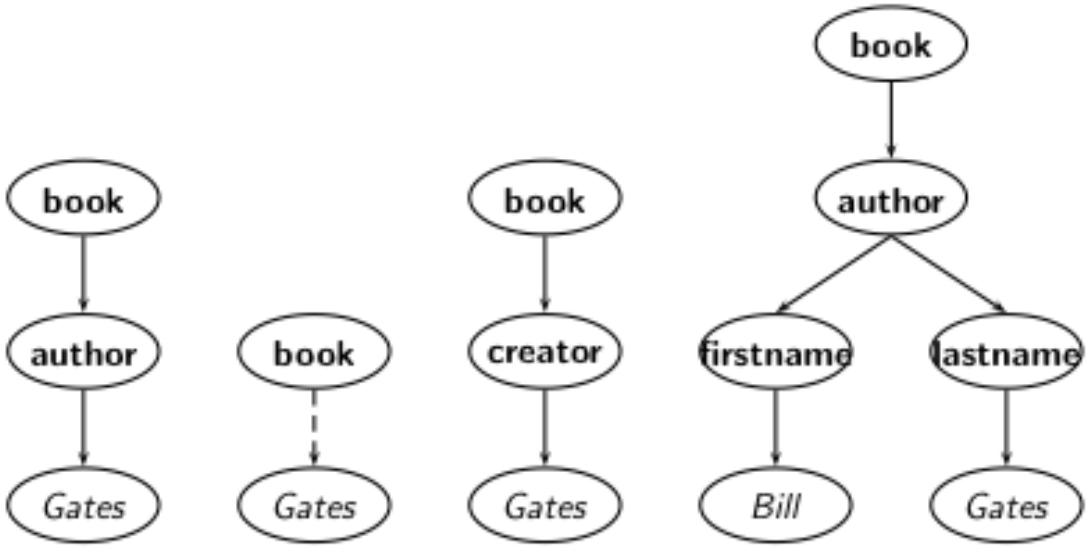
- A simple measure of the similarity of a path $c_q$ in a query and a path $c_q$ in a document is the following ***context resemblance*** function CR:

$$\mathrm{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$|c_q|$ and $|c_d|$ are the number of nodes in the query path and document path, respectively

- $c_q$ matches $c_d$ **iff** we can transform $c_q$ into $c_d$ by inserting additional nodes.

# Context resemblance example

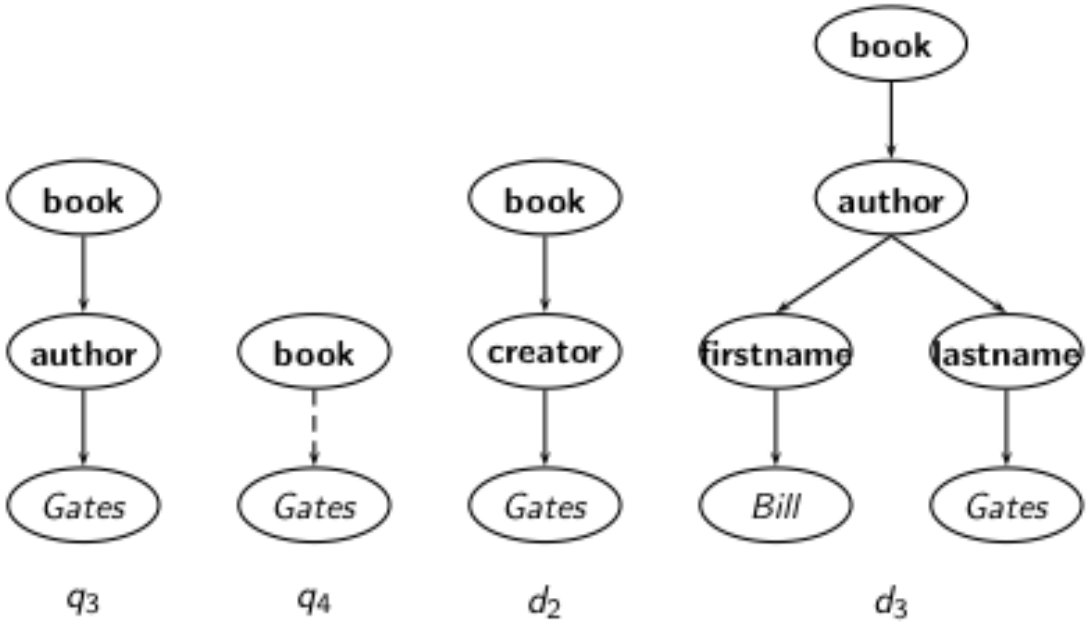



$$C_R(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

- $C_R(c_{q4}, c_{d2}) = 3/4 = 0.75$.
  The value of $C_R(c_q, c_d)$ is 1.0 if $q$ and $d$ are identical.

*Blanks on slides, you may want to fill in*

# Context resemblance example




$$C_R(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

- $C_R(c_{q?}, c_{d?}) = C_R(c_q, c_d) = 3/5 = 0.6$.

# Document similarity measure

- The final score for a document is computed as a variant of the cosine measure, which we call SIMNOMERGE.
- SIMNOMERGE($q$, $d$) =

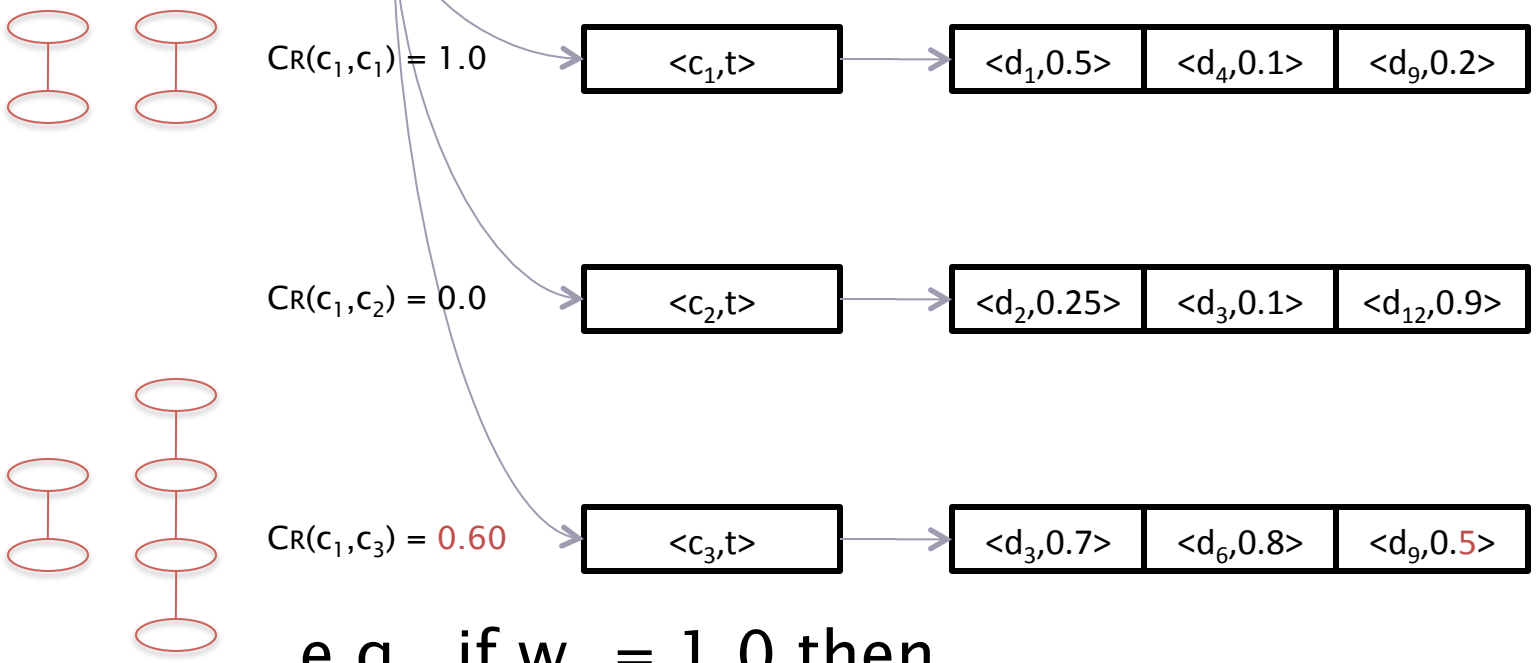
$$\sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q, t, c_k) \frac{\text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}}$$

- $V$ is the vocabulary of non-structural terms
- $B$ is the set of all XML contexts
- weight ($q$, $t$, $c$), weight($d$, $t$, $c$) are the weights of term t in XML context $c$ in query $q$ and document $d$, resp. (standard weighting e.g. $\text{idf}_t$ x $\text{wf}_{t,d}$, where $\text{idf}_t$ depends on which elements we use to compute $\text{df}_t$.)
- SIMNOMERGE($q$, $d$) is not a true cosine measure since its value can be larger than 1.0.

*Blanks on slides, you may want to fill in*

# SIMNOMERGE example

example **slightly different** from book




e.g., if $w_q = 1.0$ then
$sim(q,d_9) = 1.0\ (1.0 \times 0.2 + 0.6 \times 0.5 = .5$

"No Merge" because each context is separately calculated

# SIMNOMERGE algorithm



NUS National University of Singapore

SCOREDOCUMENTSWITHSIMNOMERGE($q$, $B$, $V$, $N$, *normalizer*)

```
1  for n ← 1 to N
2  do score[n] ← 0
3  for each ⟨c_q, t⟩ ∈ q
4  do w_q ← WEIGHT(q, t, c_q)
5      for each c ∈ B
6      do if CR(c_q, c) > 0
7            then postings ← GETPOSTINGS(⟨c, t⟩)
8                 for each posting ∈ postings
9                 do x ← CR(c_q, c) * w_q * weight(posting)
10                   score[docID(posting)] += x
11 for n ← 1 to N
12 do score[n] ← score[n]/normalizer[n]
13 return score
```

NUS National University of Singapore

# XML IR EVALUATION

# Initiative for the Evaluation of XML retrieval (INEX)

INEX: standard benchmark evaluation (yearly) that has produced test collections (documents, sets of queries, and relevance judgments).
Based on IEEE journal collection (since 2006 INEX uses the much larger English Wikipedia test collection).
The relevance of documents is judged by human assessors.

| INEX 2002 collection statistics | |
|---|---|
| 12,107 | number of documents |
| 494 MB | size |
| 1995—2002 | time of publication of articles |
| 1,532 | average number of XML nodes per document |
| 6.9 | average depth of a node |
| 30 | number of CAS topics |
| 30 | number of CO topics |

# INEX Topics

- Two types:
  1. content-only or **CO topics**: regular keyword queries as in unstructured information retrieval
  2. content-and-structure or **CAS topics**: have structural constraints in addition to keywords
- Since CAS queries have both structural and content criteria, relevance assessments are more complicated than in unstructured retrieval

# INEX relevance assessments

- INEX 2002 defined component coverage and topical relevance as orthogonal dimensions of relevance.

**Component coverage**

Evaluates whether the element retrieved is "structurally" correct, i.e., neither too low nor too high in the tree.

- We distinguish four cases:

1. Exact coverage (E): The information sought is the main topic of the component and the component is a meaningful unit of information.
2. Too small (S): The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information.
3. Too large (L): The information sought is present in the component, but is not the main topic.
4. No coverage (N): The information sought is not a topic of the component.

# INEX relevance assessments

- The **topical relevance** dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0).

## Combining the relevance dimensions

Components are judged on both dimensions and the judgments are then combined into a digit-letter code, e.g. **2S** is a fairly relevant component that is too small. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination **3N** is not possible.

# INEX relevance assessments

- The relevance-coverage combinations are quantized as follows:

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

- This evaluation scheme takes account of the fact that binary relevance judgments are not appropriate for XML retrieval. The quantization function **Q** instead allows us to grade each component as partially relevant. The number of relevant components in a retrieved set *A* of components can then be computed as:

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

# INEX evaluation measures

- As an approximation, the standard definitions of precision and recall can be applied to this modified definition of relevant items retrieved, with some subtleties because we sum graded as opposed to binary relevance assessments.

**Drawback**

Overlap is not accounted for. Accentuated by the problem of multiple nested elements occurring in a search result.

- Recent INEX focus: develop algorithms and evaluation measures that return non-redundant results lists and evaluate them properly.

# Summary

- Structured or XML IR: effort to port unstructured IR know-how to structured (DB-like) data
- Specialized applications such as patents and digital libraries

Resources

- IIR Ch 10
- http://inex.is.informatik.uni-duisburg.de/