文本挖掘技术(2009)

第十四章：

半结构化文本挖掘

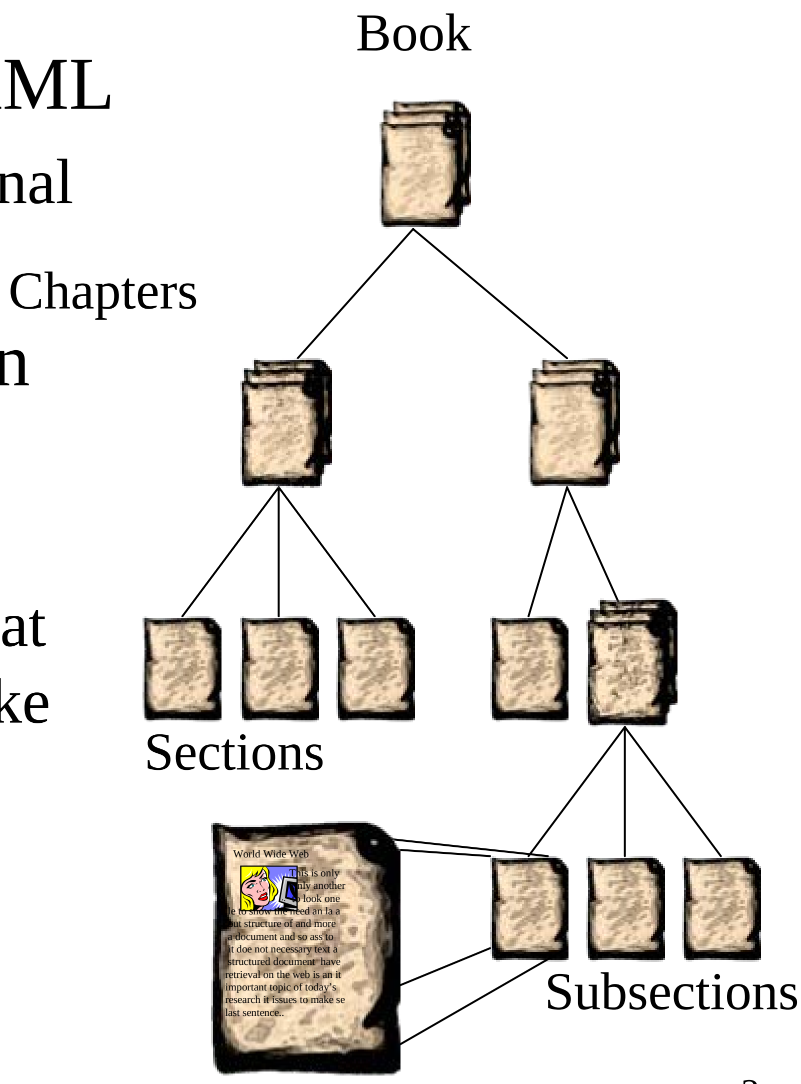
杨建武

北京大学计算机科学技术研究所

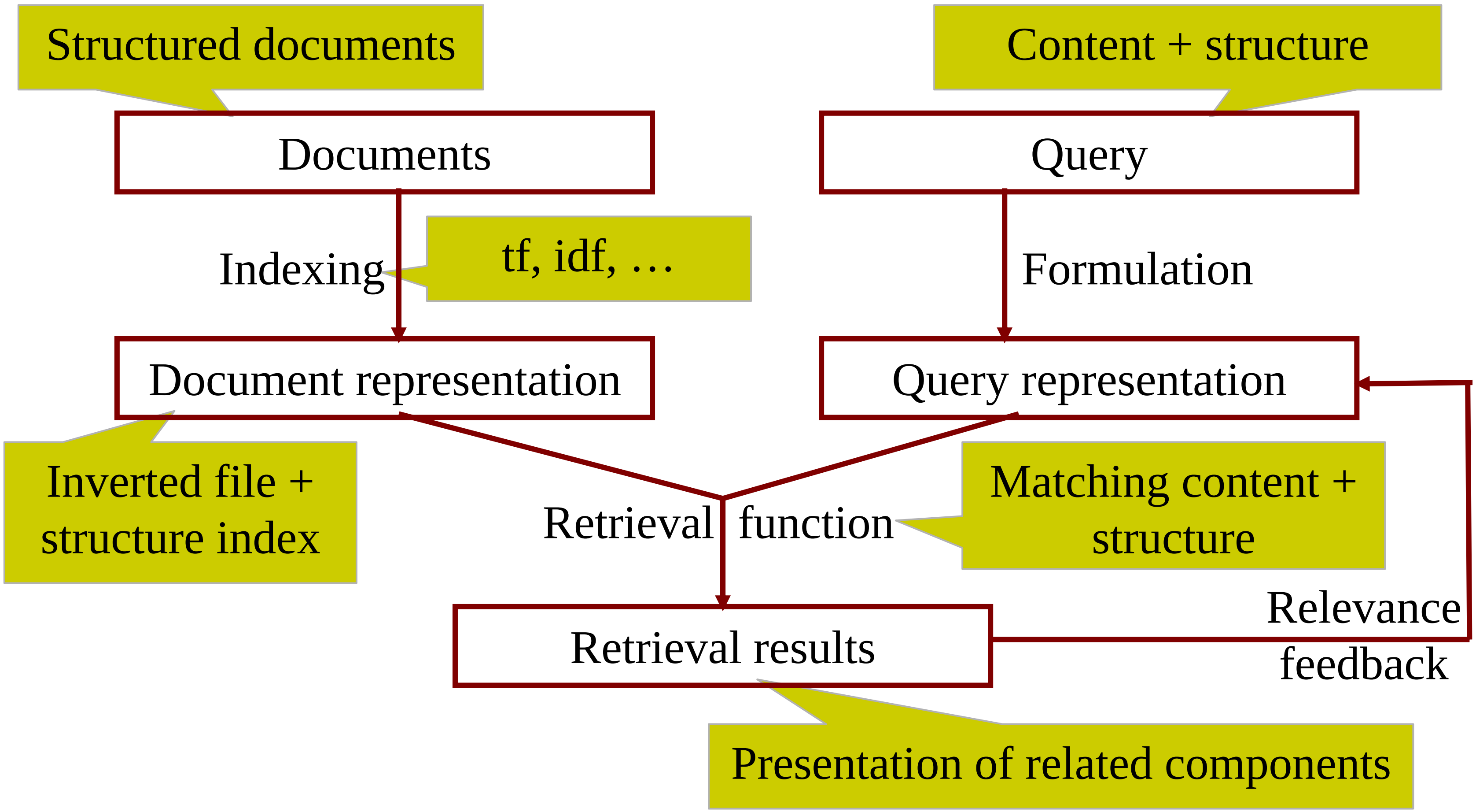
Email:yangjianwu@icst.pku.edu.cn

Text-centric XML retrieval

- Documents marked up as XML
  - E.g., assembly manuals, journal issues …
- Queries are user information needs
  - E.g., give me the Section (element) of the document that tells me how to change a brake light



Conceptual model



Approaches …

divergence from randomness

Bayesian network

fusion

machine learning

vector space model

collection statistics

language model

smoothing

cognitive model

proximity search

tuning

belief model

Boolean model

relevance feedback

phrase

parameter estimation

logistic regression

probabilistic model

component statistics

ontology

term statistics

natural language processing

extending DB model

Language model

element language model
collection language model
smoothing parameter λ

element score

high value of λ leads to increase in size of retrieved elements

element size
element score
article score

rank element

query expansion with blind feedback
ignore elements with ≤ 20 terms

results with λ = 0.9, 0.5 and 0.2 similar

(University of Amsterdam, INEX 2003)

Vector space model

article index → RSV → normalised RSV
abstract index → RSV → normalised RSV
section index → RSV → normalised RSV
sub-section index → RSV → normalised RSV
paragraph index → RSV → normalised RSV

→ merge

tf and idf as for fixed and non-nested retrieval units

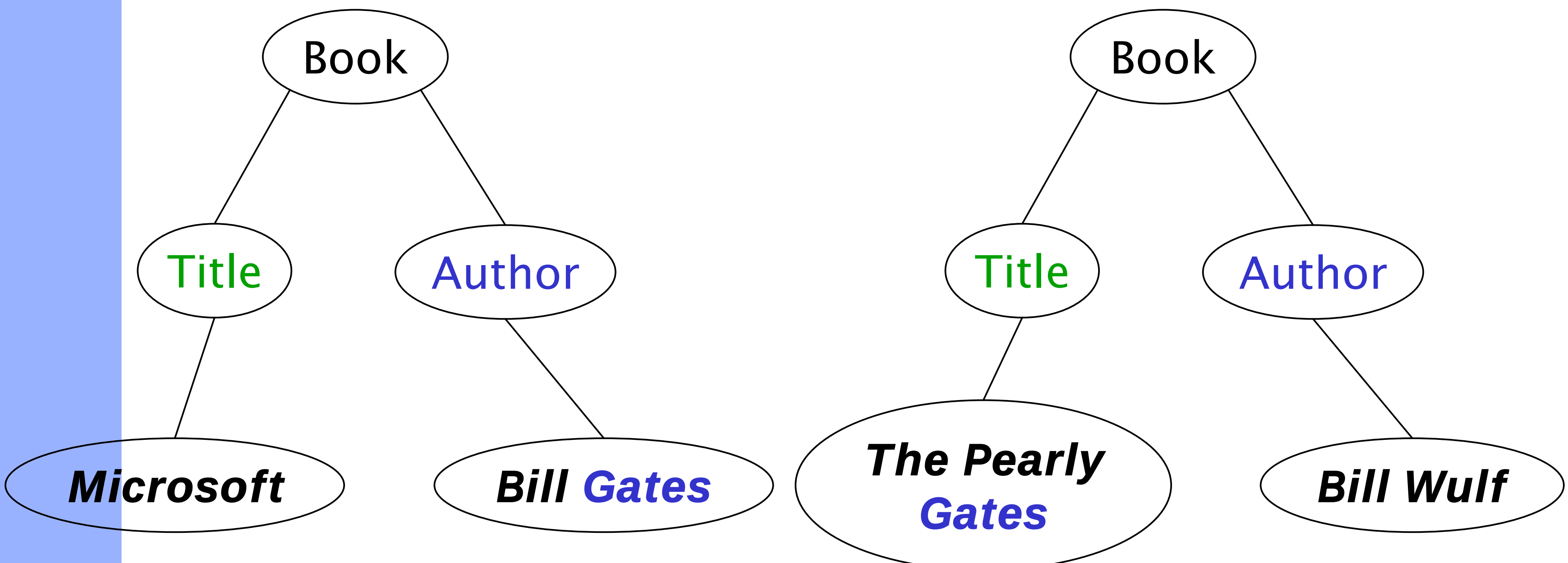
(IBM Haifa, INEX 2003)

Vector spaces and XML

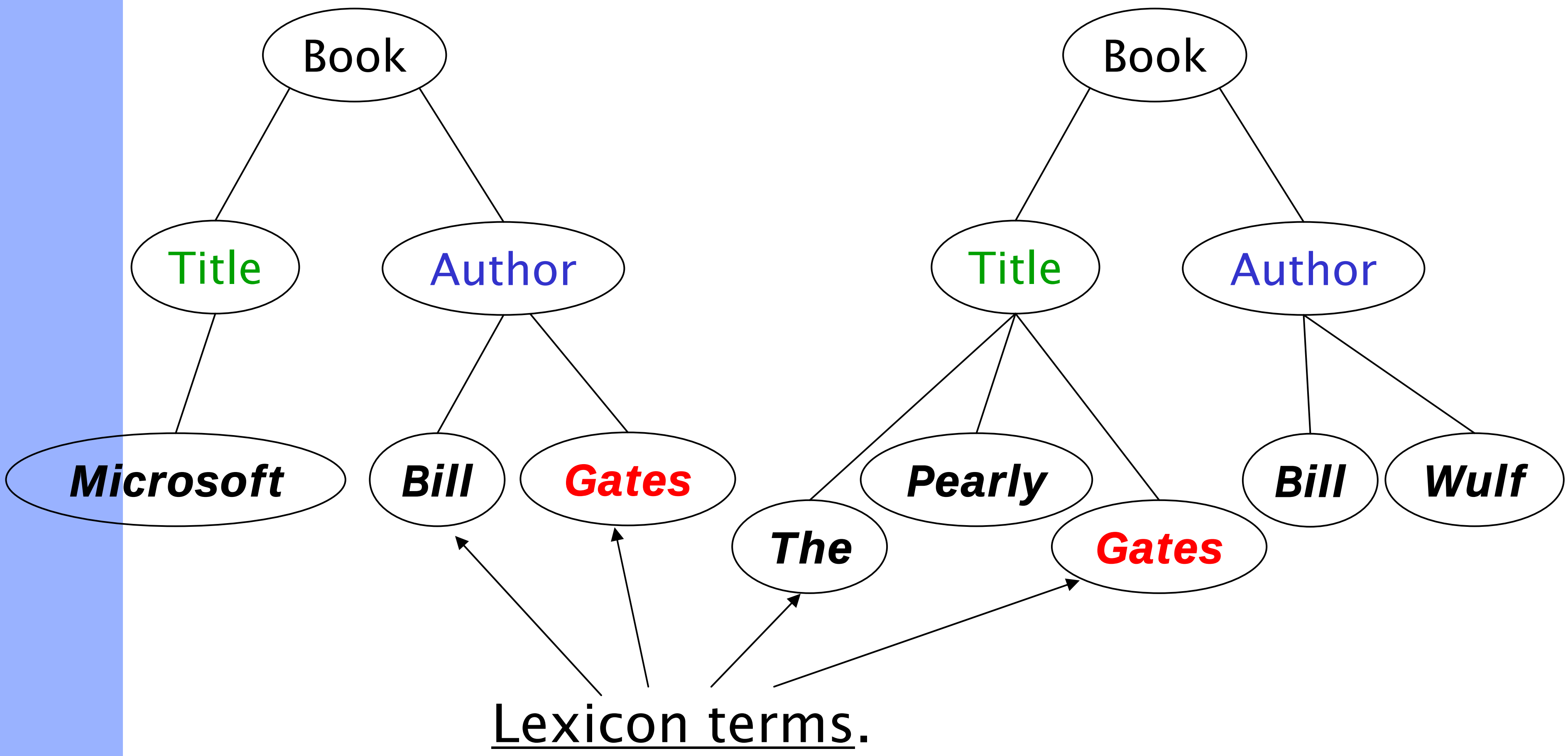
- Vector spaces – tried+tested framework for keyword retrieval
  - Other "bag of words" applications in text: classification, clustering …
- For text-centric XML retrieval, can we make use of vector space ideas?
- Challenge: capture the structure of an XML document in the vector space.

Vector spaces and XML

- For instance, distinguish between the following two cases



Content-rich XML: representation

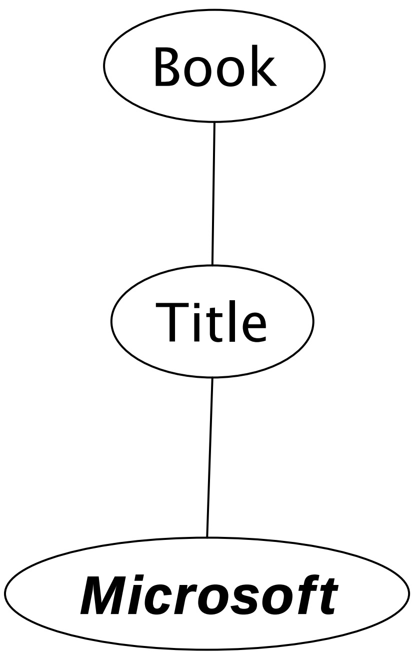


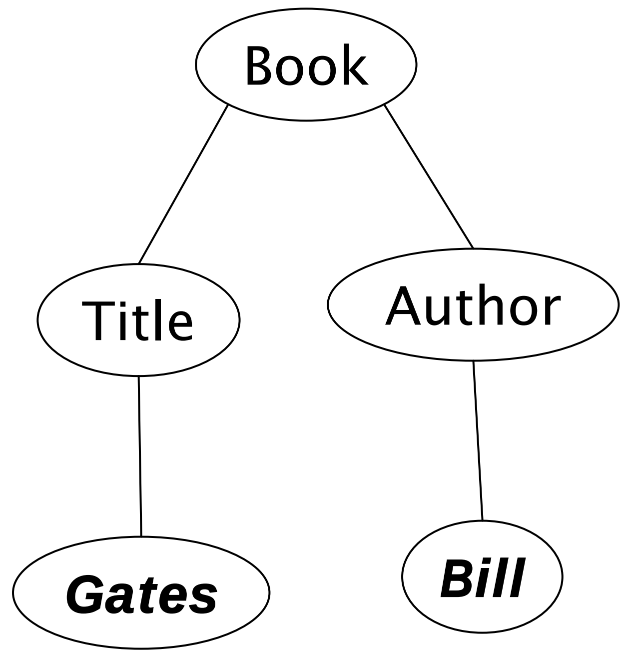
Encoding the Gates differently

- What are the axes of the vector space?
- In text retrieval, there would be a single axis for Gates
- Here we must separate out the two occurrences, under Author and Title
- Thus, axes must represent not only terms, but something about their position in an XML tree

Queries

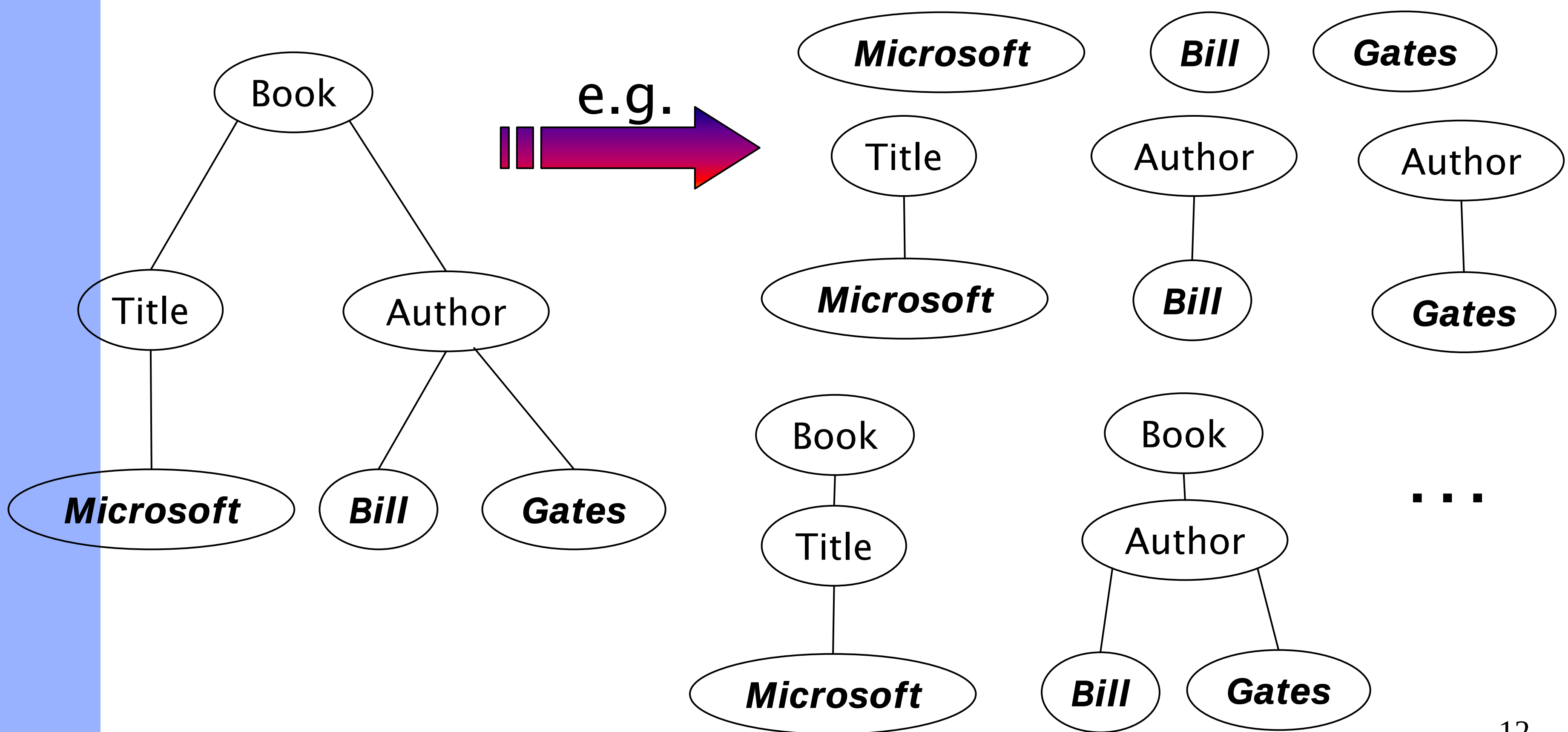
- Before addressing this, let us consider the kinds of queries we want to handle





Subtrees and structure

- Consider all subtrees of the document that include at least one lexicon term:



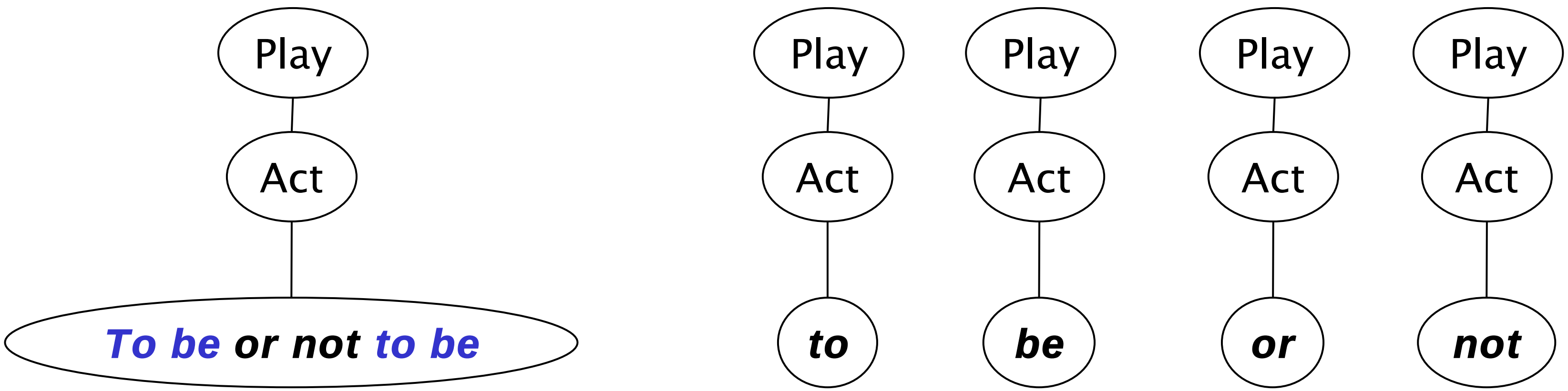
Structural terms: docs+queries

- Call each of the resulting (8+, in the previous slide) subtrees a *structural term*
- Create one axis in the vector space for each distinct structural term
- Each document becomes a vector in the space of structural terms
- A query tree can likewise be factored into structural terms
  - And represented as a vector
  - Allows weighting portions of the query

Structural terms Weight

- Weights based on frequencies for number of occurrences (just as we had *tf*)
- All the usual issues with terms (stemming? Case folding?) remain

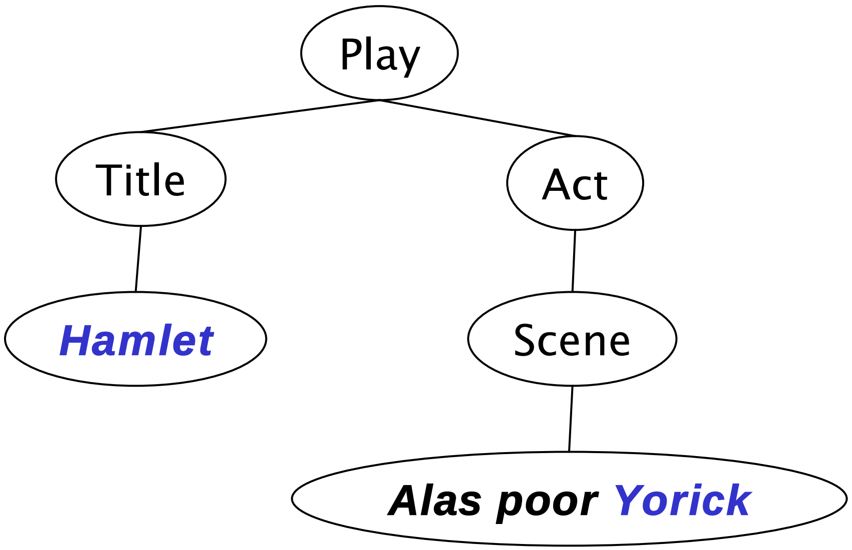
Example of *tf* weighting



How many axes are there in this example?

- Here the structural terms containing ***to*** or ***be*** would have more weight than those that don't

Down-weighting



- For the doc on the left: in a structural term rooted at the node Play, shouldn't ***Hamlet*** have a higher *tf* weight than ***Yorick***?
- Idea: multiply *tf* contribution of a term to a node k levels up by γ^k, for some $\gamma < 1$.

Down-weighting example, γ=0.8

- For the doc on the previous slide, the *tf* of
  - ***Hamlet*** is multiplied by 0.8
  - ***Yorick*** is multiplied by 0.64

in any structural term rooted at Play.

The number of structural terms

- Can be huge! ← Alright, how huge, really?
- Impractical (不切实际的) to build a vector space index with so many dimensions
- Will examine pragmatic (注重实效的) solutions to this shortly; for now, continue to believe …

Restrict structural terms?

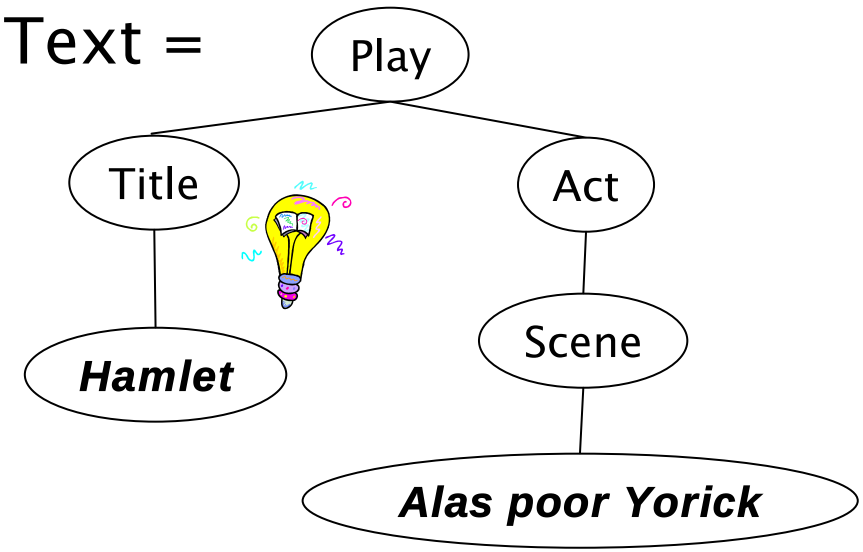
- Depending on the application, we may restrict the structural terms
  - E.g., may never want to return a Title node, only Book or Play nodes
  - So don't enumerate/index/retrieve/score structural terms rooted at some nodes
- Two solutions
  - Query-time materialization of axes
  - Restrict the kinds of subtrees to a manageable set

Query-time materialization

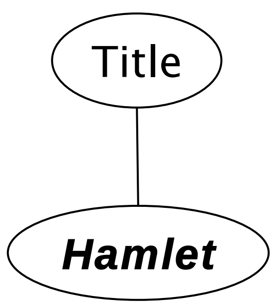
- Instead of enumerating all structural terms of all docs (and the query), enumerate only for the query
  - The latter is hopefully a small set
- Now, we're reduced to checking which structural term(s) from the query match a subtree of any document
- This is tree pattern matching: given a *text tree* and a *pattern tree*, find matches
  - Except we have many text trees
  - Our trees are labeled and weighted

Example

Text =



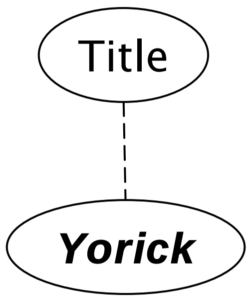
Query =



- Here we seek a doc with ***Hamlet*** in the title
- On finding the match we compute the cosine similarity score
- After all matches are found, rank by sorting

(Still infeasible (不可实行的))

- A doc with ***Yorick*** somewhere in it:
- Query =

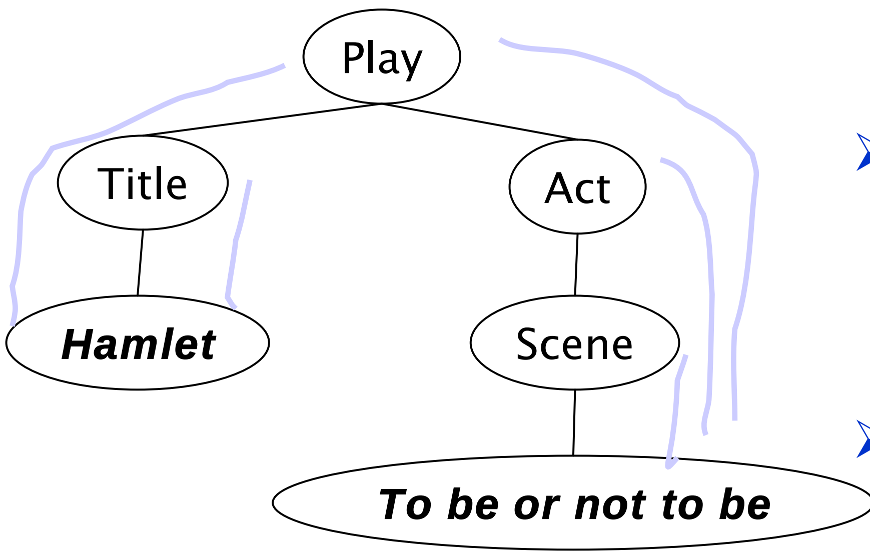


- Will get to it …

Restricting the subtrees

- Enumerating all structural terms (subtrees) is prohibitive, for indexing
  - Most subtrees may never be used in processing any query
- Can we get away with indexing a restricted class of subtrees
  - Ideally – focus on subtrees likely to arise in queries

JuruXML (IBM Haifa)

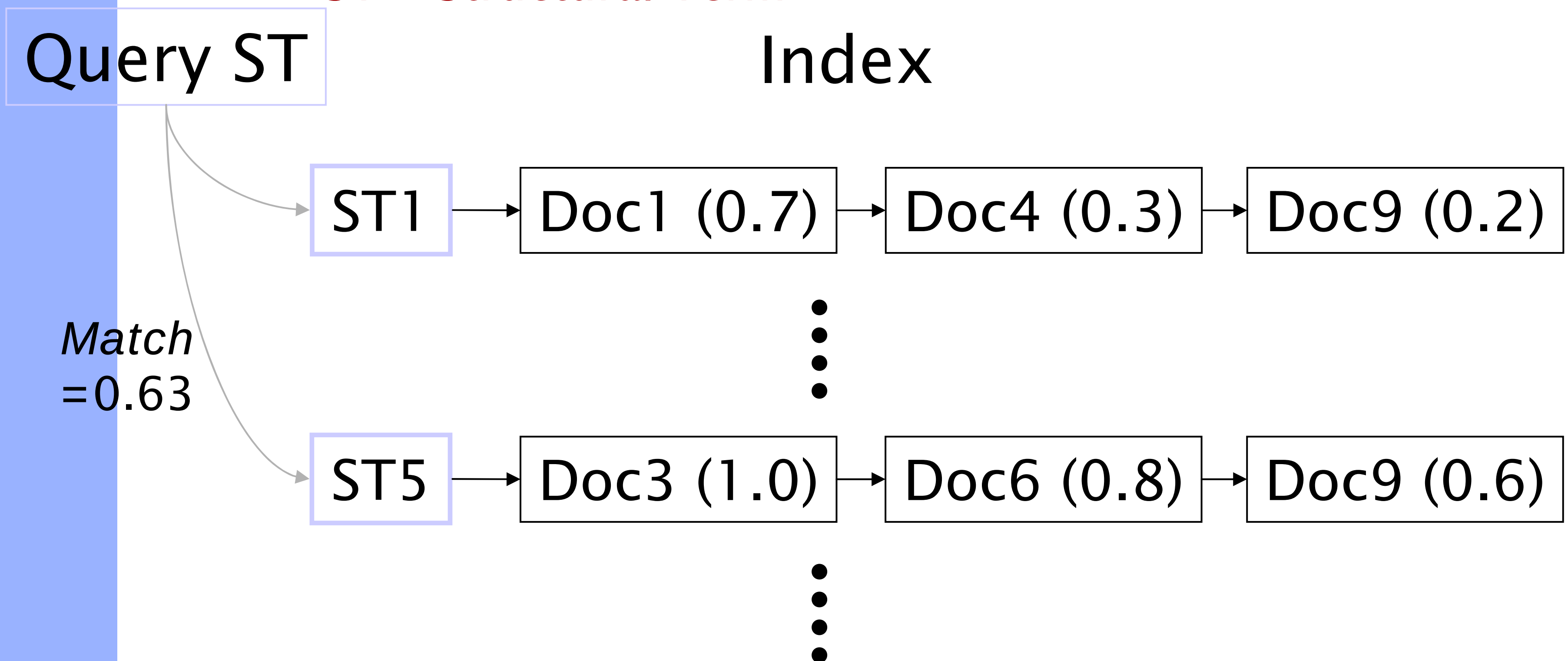


- Only paths including a lexicon term
- In this example there are only 14 (why?) such paths
- Thus we have 14 structural terms in the index

Why is this far more manageable?
How big can the index be as a function of the text?

Example of a retrieval step

ST = Structural Term



Now rank the Doc's by cosine similarity;
e.g., Doc9 scores 0.578.

XQuery

XQuery

- SQL for XML
- Usage scenarios
  - Human-readable documents
  - Data-oriented documents
  - Mixed documents (e.g., patient records)
- Relies on
  - XPath
  - XML Schema datatypes

XQuery

- The **principal forms** of XQuery expressions are:
  - path expressions
  - element constructors
  - FLWR ("flower") expressions
  - list expressions
  - conditional expressions
  - quantified expressions
  - datatype expressions
- Evaluated with respect to a context

FLWR

- FOR $p IN document("bib.xml")//publisher LET $b := document("bib.xml")//book[publisher = $p] WHERE count($b) > 100 RETURN $p
- FOR generates an ordered list of bindings of publisher names to $p
- LET associates to each binding a further binding of the list of book elements with that publisher to $b
- at this stage, we have an ordered list of tuples of bindings: ($p,$b)
- WHERE filters that list to retain only the desired tuples
- RETURN constructs for each tuple a resulting value

Queries Supported by XQuery

- Location/position (“chapter no.3”)
- Simple attribute/value
  - /play/title contains “hamlet”
- Path queries
  - title contains “hamlet”
  - /play//title contains “hamlet”
- Complex graphs
  - Employees with two managers
- Subsumes: hyperlinks
- What about relevance ranking?

XQuery : Order By Clause

- Order by clause only allows ordering by “overt” criterion
  - Say by an attribute value
- Relevance ranking
  - Is often proprietary
  - Can’t be expressed easily as function of set to be ranked
  - Is better abstracted out of query formulation (cf. www)

XQuery: Order By Clause

```
for $d in document("depts.xml")//deptno
let $e := document("emps.xml")//emp[deptno = $d]
where count($e) >= 10
order by avg($e/salary) descending
return <big-dept> { $d,
    <headcount>{count($e)}</headcount>,
    <avgsal>{avg($e/salary)}</avgsal> } </big-dept>
```

XML Indexing

Native XML Database

- Uses XML document as logical unit
- Should support
  - Elements
  - Attributes
  - PCDATA (parsed character data)
  - Document order
- Contrast with
  - DB modified for XML
  - Generic IR system modified for XML

XML Indexing and Search

- Most native XML databases have taken a DB approach
  - Exact match
  - Evaluate path expressions
  - No IR type relevance ranking
- Only a few that focus on relevance ranking

Data vs. Text-centric XML

- Data-centric XML:
  - mainly a recasting of relational data
  - used for messaging between enterprise applications
- Content-centric XML:
  - used for annotating content
  - Rich in text
  - Demands good integration of text retrieval functionality
  - E.g., find me the ISBN #s of Books with at least three Chapters discussing cocoa production, ranked by Price

Data structures for XML retrieval

Data structures for XML retrieval

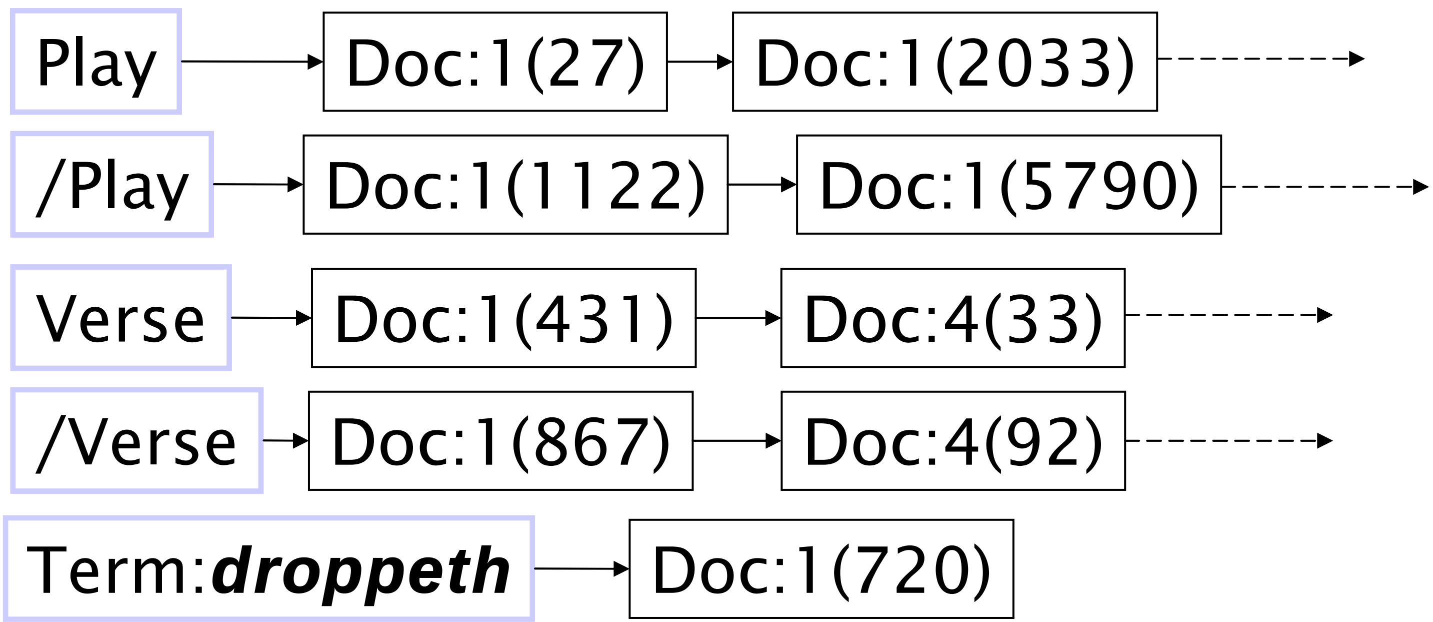
- What are the primitives (原始的) we need?
- Inverted index: give me all elements matching text query *Q*
  - We know how to do this – treat each element as a document
- Give me all elements (immediately) below any instance of the Book element
- Combination of the above

Parent/child links

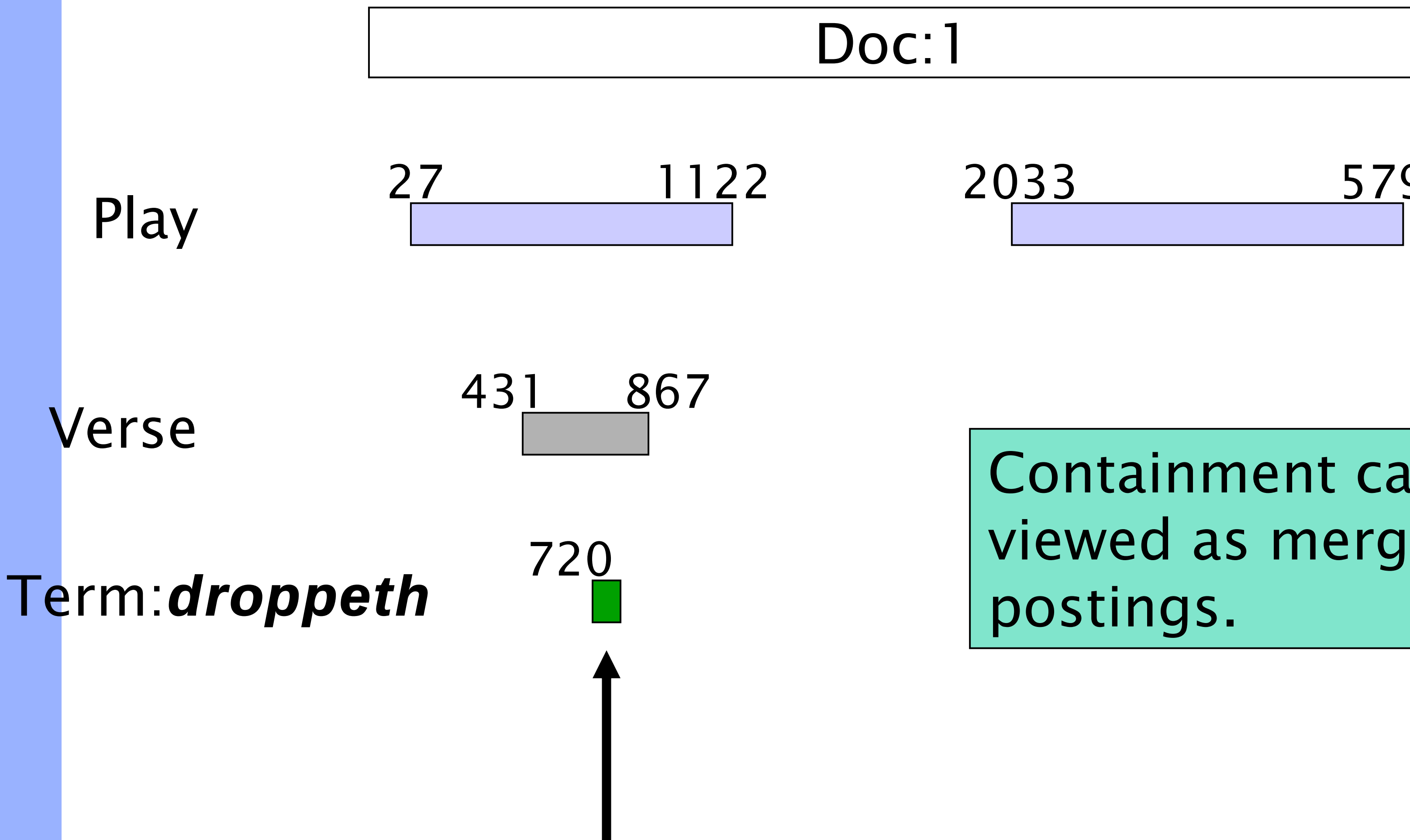
- Number each element
- Maintain a list of parent-child relationships
  - E.g., Chapter:21 ← Book:8
  - Enables immediate parent
- But what about “the word ***Hamlet*** under a Scene element under a Play element?

General positional indexes

- View the XML document as a text document
- Build a positional index for each *element*
  - Mark the beginning and end for each element, e.g.,



Positional containment



Containment can be viewed as merging postings.

droppeth under Verse under Play.

Summary of data structures

- Path containment etc. can essentially be solved by positional inverted indexes
- Retrieval consists of "merging" postings
- All the compression tricks etc. from 276A are still applicable
- Complications arise from insertion/deletion of elements, text within elements
  - Beyond the scope of this course

INEX: a benchmark for text-centric XML retrieval

INEX

- INitiative for the Evaluation of XML Retrieval
- http://www.inex.otago.ac.nz
- Sponsored by
  - DELOS Network of Excellence for Digital Libraries
  - IEEE Computer Society
- Benchmark for the evaluation of XML retrieval
- Consists of:
  - Set of XML documents
  - Collection of retrieval tasks

INEX

- Task
  - ad hoc
  - Document mining
    - http://xmlmining.lip6.fr/
  - Multimedia
  - Entity Ranking
  - Book searching
  - Document collection interlinking (Link the Wiki)

INEX-- ad hoc task

- Each engine indexes docs
- Engine team converts retrieval tasks into queries
  - In XML query language understood by engine
- In response, the engine retrieves not docs, but elements within docs
  - Engine ranks retrieved elements

INEX corpus

- INEX 2006 (IEEE Computer Society publications)
  - 12,107 articles
  - 494 Megabytes
  - Average article:1,532 XML nodes
    - Average node depth = 6.9
- INEX 2007 (Wikipedia documents)
  - the XML full-texts of 659,388 articles
  - more than 60 GB (4.6GB without images)
  - 30 million elements
  - article: 161.35 nodes; depth: 6.72

INEX topics

- Each topic is an information need, one of two kinds:
  - Content Only (CO)
    - free text queries
  - Content and Structure (CAS)
    - explicit structural constraints,
      - e.g., containment conditions.

Sample INEX CO topic

<**Title**> computational biology </**Title**>

<**Keywords**> computational biology, bioinformatics, genome, genomics, proteomics, sequencing, protein folding </**Keywords**>

<**Description**> Challenges that arise, and approaches being explored, in the interdisciplinary field of computational biology</**Description**>

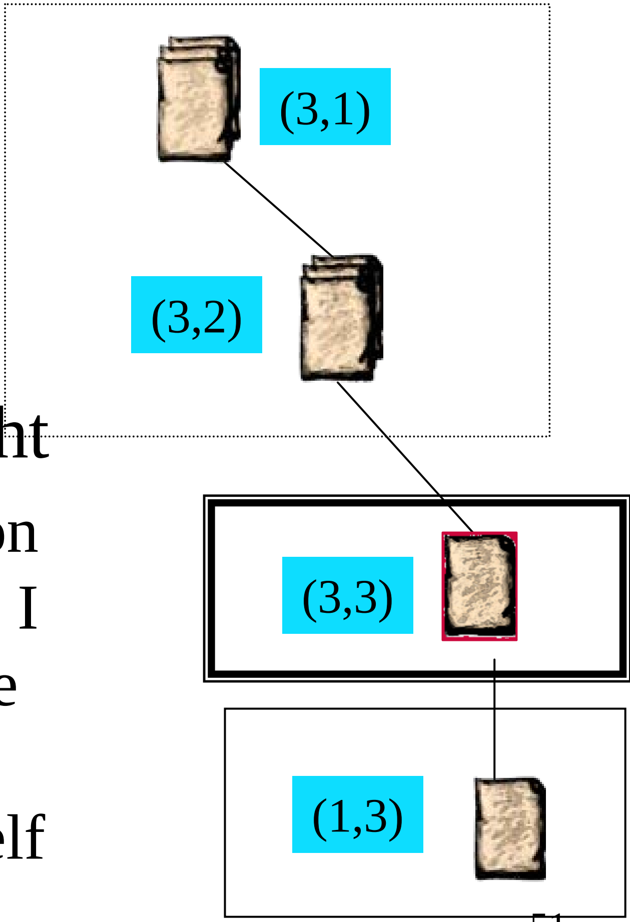
<**Narrative**> To be relevant, a document/component must either talk in general terms about the opportunities at the intersection of computer science and biology, or describe a particular problem and the ways it is being attacked. </**Narrative**>

INEX assessment

- Each engine formulates the topic as a query
  - E.g., use the keywords listed in the topic.
- Engine retrieves one or more elements and ranks them.
- Human evaluators assign to each retrieved element relevance and coverage scores.

INEX assessment

- <u>Relevance</u>
  - how relevant is the retrieved element
- <u>Coverage</u>
  - is the retrieved element too specific, too general, or just right
    - E.g., if the query seeks a definition of the Fast Fourier Transform, do I get the equation (too specific), the chapter containing the definition (too general) or the definition itself



Assessments

- Relevance assessed on a scale from Irrelevant (scoring 0) to Highly Relevant (scoring 3)
- Coverage assessed on a scale with four levels:
  - No Coverage (N: the query topic does not match anything in the element
  - Too Large (The topic is only a minor theme of the element retrieved)
  - Too Small (S: the element is too small to provide the information required)
  - Exact (E).
- These assessments are turned into composite precision/recall measures, $\{0,1,2,3\} \times \{N,S,L,E\}$

Combining the assessments

- Define scores:

$$f_{strict}(rel,cov) = \begin{cases} 1 & \text{if } rel,cov = 3E \\ 0 & \text{otherwise} \end{cases}$$

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

XML Mining Track

- INEX Document mining track
- 2005，2006，2007，2008
- http://xmlmining.lip6.fr
- Two main tasks:
  - Categorization
  - Clustering

XML Mining Track 2007

- Corpus
  - 96,000 documents extracted from the Wikipedia XML Corpus
    - The training part composed of 50% of the documents,
    - The testing part composed of the 50% remaining documents.
  - 21 categories

	Total	Train	Test
Number of documents	96,611	48,306	48,305
Number of internal nodes	$\approx$ 9 M	4,505,141	4,487,819
Number of distinct words	446,916 (depending on the preprocessing)		
Number of words	33,944,462	17,261,996	16,682,466
Mean length of the documents	351.4	357.3	345.5
Number of distinct tags	$\approx$ 5,800		
Size of the corpus	$\approx$ 720 Mbytes	$\approx$ 360 Mbytes	$\approx$ 360 Mbytes

Clustering Results

Article	Micro-average purity	Macro-average purity	Number of clusters
Yao et al.	44.4 %	44.7 %	5
Yao et al.	53.4 %	60.2 %	10
Hagenbuchner et al.	25.1 %	26.6 %	10
Tran et al	38.9 %	40.4 %	10
Kutty et al.	25.0 %	24.9 %	10
Yao et al.	53.6	59.1	15
Yao et al.	57.9	67.2	21
Tran et al.	37.6	39.9	21
Hagenbuchner et al.	26.4	26.9	21
Kutty et al.	25.0	25.0	21
Yao et al.	59.5	66.3	30
Tran et al.	38.9	40.4	30

Categorization (2006)

	F1 micro	F1 macro
NB	0.59	0.605
Structure model	0.619	0.622
SVM TF-IDF	0.534	0.564
Fisher kernel	**0.661**	**0.668**
Discriminant learning	0.575	0.600

Structure helps in finding the category of a document !

Categorization (2007)

Article	Micro-average recall	Macro-average recall
Yang et al.	87.2 %	83.9 %
Campos et al.	78.9 %	76.0 %
Murugeshan et al.	78.0 %	75.7 %

Yang et al. in INEX 2007

```
<article>
   <title>Ontology Enabled Web Search</name>
   <author>John</author>
   <conference>Web Intelligence</conference>
</article>
```

$$\Delta_x = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{ccc} \text{title} & \text{author} & \text{conference} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{Ontology} \\ \text{Enabled} \\ \text{Web} \\ \text{Search} \\ \text{John} \\ \text{Intelligence} \end{array}$$

Yang et al. in INEX 2007

$$\begin{array}{rcl} & \begin{array}{ccc} \text{title} & \text{author} & \text{conference} \end{array} & \\ \widetilde{\Delta}_x = & \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0.5 & 0 & \sqrt{2}/2 \\ 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \sqrt{2}/2 \end{bmatrix} & \begin{array}{l} \text{Ontology} \\ \text{Enabled} \\ \text{Web} \\ \text{Search} \\ \text{John} \\ \text{Intelligence} \end{array} \end{array}$$

$$sim(doc_x, doc_y) = \sum_{j=1}^{m} \sum_{i=1}^{m} M_{e(i,j)} \cdot (\widetilde{\Delta}_{x(i)}{}^{T} \bullet \widetilde{\Delta}_{y(j)})$$

$$k(x_i, x_j) = sim(doc_x, doc_y) = \sum_{j=1}^{m} \sum_{i=1}^{m} M_{e(i,j)} \cdot (\widetilde{\Delta}_{x(i)}{}^{T} \bullet \widetilde{\Delta}_{y(j)})$$

小结

- XML文档特点
- XQuery
- XML Indexing
- INEX:
  - Ad hoc
  - XML Mining

Any Question?