Outline

1. Recall pre / post encoding
2. XPath with //, ancestor, (@, and text)
3. XPath with / and following-sibling:
   - use pre / size / level encoding
4. Optimization through tree-aware join: "Staircase Join"
   - Prune
   - Partition and Skip

Outline - Lectures

1. Introduction to XML, Encodings, Parsers
2. Memory Representations for XML: Space vs Access Speed
3. RDBMS Representation of XML
4. DTDs, Schemas, Regular Expressions, Ambiguity
5. XML Validation using Automata
6. Node Selecting Queries: XPath
7. Tree Automata for Efficient XPath Evaluation, Parallel Evaluation
8. XPath Properties: backward axes, containment test
9. Streaming Evaluation: how much memory do you need?
10. XPath Evaluation using RDBMS
11. XSLT – stylesheets and transform
12. XQuery – XML query language
13. Wrap up, Exam prep, misc.

Outline - Assignments

2. SAX Parse into memory structure: Tree and DAG
3. Map XML into
4. XPath evaluation
5. XPath into SQL Translation → 31st May

Questions

What is the benefit of storing the LEVEL of a node in the table? Which accessor functions become easier?

→ It can also be useful to store the size of the subtree at a node in the table. Do you see advantages of doing that?
Later in this course, we will use the PRE/POST encoding again.

We will find a systematic way to map queries on XML (XPath) into XQL queries.

Assignment 5 is about programming this mapping.

Assignment 3 generate tables, send to DB, run query & print results

XML document

1. Recall: Pre/Post Encoding

→ Add post order

CREATE VIEW descendant AS
SELECT r1.pre, r2.pre FROM doc_tbl r1, R r2
WHERE r1.pre < r2.pre
AND r1.post > r2.post

“structural join”

XML Database – Table Storage

Attributes:
- attributes are “owned” by their elements → no table storage
- another table field is introduced which references attribute arrays

XPath Accelerator encoding

XML fragment f and its skeleton tree

Pre/post encoding of f: table accel

XML Database – Table Storage
Our Tables: Assignment 3 + extra root node

XML document

<book isbn="1-2345-6789-0" year="1994">
<title>TCP/IP Illustrated</title>
<author><last>Stevens</last><first>John</first></author>
<publisher>Addison-Wesley</publisher>
<price currency="USD">65.95</price>
</book>

book_tbl:
0 13 null null null
1 12 0 book null
2 2 1 title null
3 1 2 null TCP/IP Illustrated
4 7 1 author null
5 4 4 last null
6 3 5 null Stevens
7 6 4 first null
8 5 7 null John
9 9 1 publisher null
10 8 9 null Addison-Wesley
11 11 2 price null
12 10 3 null 65.95

book_attr:
1 isbn 1-2345-6789-0
2 year 1994
11 currency USD

More flexible, of course, to use TYPE-field with values {elem, text, root, com, PI .. }

Add root node with
pre = 0
post = #nodes+1
parent = null
tag = null
text = null

ancestor (pr, po) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre<pr
AND r1.post>po
ORDERED BY r1.pre

ancestor (pr, po) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre<pr
AND r1.post>po
ORDERED BY r1.pre

pre=0
pre=1
pre=2

ancestor (pr, po) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre<pr
AND r1.post>po
ORDERED BY r1.pre

XPath evaluation: need to operate wrt a set of context nodes, stored in intermediate table.

2. XPath with //, ancestor, @, and text()

XPath path expression s/s/s...s
Each step si is of the form axis:node/test
Each axis results in context node sequence for step si
Other axes define efficiently computable subsets of the four major axes

Evaluation in DBMS
Apply each location step to all nodes in context (bulk-oriented query processing)
Preserve document order and not produce duplicate nodes

Initial focus on major axis steps
Major, ancestor, descendant, preceding, and following
Other axes define efficiently computable subsets of the four major axes

XPath data model
root node
pre=0
Watch out:
Must be "extra" root node
→ XPath data model/
Formally: do you remember the correct definition of the abbreviation "//"?

// is abbreviation for /descendant-or-self::node/ 
//book is abbreviation for /descendant-or-self::node/book 

descendant :: book

Cave
Not fully correct!
Works only if author-nodes have a single text node, as first child…

Correct: returns result nodes in document order.
XPath with //, ancestor, @, and text()

```xml
//book/author[@name="Knuth"]
```

```
SELECT r3.pre FROM book_tbl r1, r2, r3,
book_attr a
WHERE r1.pre=0
AND r2.pre>r1.pre
AND r2.post<r1.post
AND r2.tag="book"
AND r3.pre>r2.pre
AND r3.post<r2.post
AND r3.tag="author"
AND r3.pre=a.pre
AND a.attr="name"
AND a.value="Knuth"
ORDERED BY r3.pre
```

Correct: returns result nodes in document order.

Question

Can there be duplicates produced by this query?

What about duplicates? E.g. //author/ancestor::*

```
DISTINCT
SELECT r3.pre FROM book_tbl r1, r2, r3,
book_attr a
WHERE r1.pre=0
AND r2.pre>r1.pre
AND r2.post<r1.post
AND r2.tag="book"
AND r3.pre>r2.pre
AND r3.post<r2.post
AND r3.tag="author"
AND r3.pre=a.pre
AND a.attr="name"
AND a.value="Knuth"
ORDERED BY r3.pre
```

What about other axes? E.g., child, parent etc??
XPath with `/` and following-sibling

firstChild(pr, po) = left-most node, below and to the right of (pr, po)

Not good!

Needs

```
SELECT min(pre) FROM ...
```

But easy:

```
AND r2.pre=r1.pre+1
AND r2.post<r1.post
```

How to select ALL children of a node?

maybe

descendant, i.e., larger pre and smaller post,
AND no other descendant inbetween

can be done in SQL, but is expensive!

```
... AND NOT EXIST (SELECT ...)
```

e.g., not c- and d-node
(because b-node is inbetween...)

firstChild(pr, po) = left-most node, below and to the right of (pr, po)

nextSibling(pr, po) =

left-most node (pr2, po2),

→ to the right
→ up

such that there is no node
with post value > po and < po2.

... AND NOT EXIST (SELECT ...)

e.g., not c- and d-node
(because b-node is inbetween...)
firstChild \( (pr, po) \) = left-most node, below and to the right of \((pr,po)\)

following-sibling

Can be done similar as before, using a complicated SQL query.

Æ Expensive

Use other encoding!!

PRE / SIZE / LEVEL

```
  a
  b
  c
  d
```

Can you compute the post-value of a node, from its \( \{ \text{pre, size, level} \} \) values?

```
  a
  b
  c
  d
```

Useful relationships & approximations

THUS \[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

\[ \text{pre}(n) = \text{x} + \text{level}(n) \]
\[ \text{post}(n) = \text{x} + \text{size}(n) \]
THUS \[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

number \( x \) of nodes

\[ \text{pre}(n) = x \times \text{level}(n) \]
\[ \text{post}(n) = x \times \text{size}(n) \]
\[ \text{pre}(n) - \text{post}(n) = \text{level}(n) - \text{size}(n) \]

Pre / Size / Level Encoding
Useful relationships & approximations

\[ \text{descendant}(\text{pr}, \text{si}, \text{le}) = \text{child}(\text{pr}, \text{si}, \text{le}) \]
\[ \text{descendant}(\text{pr}, \text{si}, \text{le}) = \text{queries} \]
\[ \text{ancestor}(\text{pr}, \text{si}, \text{le}) = \text{queries} \]

\[
\text{descendant}(\text{pr}, \text{si}, \text{le}) = \text{SELECT ri.pre FROM doc_tbl ri WHERE ri.pre = \text{pr} AND ri.pre+ri.size >= \text{pr+si}}
\]

\[
\text{ancestor}(\text{pr}, \text{si}, \text{le}) = \text{SELECT ri.pre FROM doc_tbl ri WHERE ri.pre > \text{pr} AND ri.pre+r1.size <= \text{pr+si}}
\]

\[
\text{child}(\text{pr}, \text{si}, \text{le}) = \text{SELECT ri.pre FROM doc_tbl ri WHERE ri.pre < \text{pr} AND ri.pre+ri.size >= \text{pr}}
\]
Sometimes even store Parent with Pre / Size / Level

Level not needed, if we have parent.
4. Staircase Join

Most of the following slides are by Maurice van Keulen (Univ. of Twente, The Netherlands)

See http://edbtss04.dia.uniroma3.it/VanKeulen.pdf

### SQL Query

```sql
SELECT DISTINCT r2.pre
FROM context c,
     doc_tbl r1, r2
WHERE r1.pre  > c.pre
AND r1.post > c.post
AND r2.pre  > r1.pre
AND r2.post < r1.post
ORDERED BY r2.pr
```

### IBM DB2 Query Plan

1. `ixscanpre/post` doc_tbl r2
2. `ixscanpre/post` doc_tbl r1
3. `context c`
4. `sortpre` doc_tbl r1
5. `ixscanpre/post` doc_tbl r2

**Problem 1**

might have MANY duplicates here!

### Technique 1

Avoiding Duplicates

Select lowest independent nodes (not ancestor of each other)
4. Staircase Join

Technique 1 Pruning

→ How to Prune depends on axis

C: set of context nodes
ax: XPath axis

(1) If ax=ancestor, then
Prune(C,ax) = lowest (=bottom-most) independent nodes in C

(2) If ax=descendant, then
Prune(C,ax) = highest (=top-most) independent nodes in C

Hint: For any two context nodes N1 and N2:
following-nodes(N1) ⊆ following-nodes(N2)
OR
following-nodes(N2) ⊆ following-nodes(N1)

Why is that?
Consider arbitrary N1, N2. Either one is a descendant of the other or…

Qu: give pruning rule for
(3) ax = following
4. Staircase Join

Technique 2: Partitioning & Skipping

It scans doc and context tables sequentially.
2. It scans both tables only once for an entire context node sequence.
3. It never delivers duplicate nodes.
4. Result nodes are produced in document order.
5. Input for staircase join can be any node sequence.

\[ (3) \land (4) \Rightarrow \text{no post-processing (unique/sort) is needed to comply to XPath semantics} \]

Example //a/b:

First context: root node.
Result: all a-nodes.

Second context: all a-nodes.
Axis = descendant.
Therefore, consider all top-most a-nodes only.
Result: all their b-descendants.

\[ \Rightarrow \quad \text{improvement: we could consider only top-most a-nodes in the first step.} \]

\[ \Rightarrow \quad \text{imagine there are many a-nodes, but only a single b-leaf.} \]
In this case, bottom-up evaluation would be best.
(need selectivity estimation)

4. Staircase Join

Technique 2: Partitioning & Skipping

\[
\text{context}(\text{pre}, \text{post}) \begin{align*}
\text{begin} & \quad \text{set}\{\text{new table}\{\text{pre}, \text{post}\}\} \\
& \quad / * \text{partition } c_{\text{first}}, c_{\text{last}} * / \\
& \quad c_{\text{first}} = \text{first node in context}; \\
& \quad \text{while} (c_{\text{first}}(\text{next node in context}) \text{ do} \\
& \quad \text{if } c_{\text{last}}(\text{post}) < c_{\text{first}}(\text{post}) \text{ then} \\
& \quad \text{else} \\
& \quad \text{scanpartition_desc}(c_{\text{first}}, \text{pre}+1, c_{\text{last}}, \text{pre}+1, \\
& \quad c_{\text{first}}(\text{post})); \\
& \quad c_{\text{first}} = c_{\text{last}}; \\
& \quad \text{scanpartition_desc}(c_{\text{last}}, \text{pre}+1, \text{post}, c_{\text{last}}(\text{post})); \\
& \quad \text{return result}; \\
& \quad \text{end} \\
& \quad \text{scanpartition_desc}(\text{pre}, \text{post}) \text{ end} \\
& \quad \text{for } i \text{ from } \text{pre} \text{ to } \text{post} \text{ do} \\
& \quad \text{if } \text{dec}(i \text{ post}) < \text{post} \text{ then} \\
& \quad \text{append } \text{doc}(i) \text{ to result}; \\
& \quad \text{end} \\
\end{align*}
\]

CAVE: only true if there are no value comparisons in filters...

4. Staircase Join

More...
Skip empty regions in the plane...
4. Staircase Join

For any two nodes ‘a’ and ‘b’

(1) Nodes a and b relate to each other on the ancestor-descendant axis.

Why? \( b \) cannot be a descendant of a preceding node of \( a \).

(2) Nodes a and b relate to each other on the preceding-following axis.

Why? \( b \) cannot be a descendant of a preceding node of \( a \).

More... Skip empty regions in the plane.

Æ a’s following nodes are also following nodes of b.

Æ a and b do not have common descendants.

Experiments:

effect of skipping

Experiments: Tree-unaware vs. Tree-aware
END
Lecture 10