XML Indexing II

CPS 216
Advanced Database Systems

Announcements (April 14)

- Homework #3 will be graded by next Tuesday
- Reading assignment due next Monday
  - Selinger paper on query optimization

XML indexing overview (review)

- It is a jungle out there
  - Different representation scheme lead to different indexes
  - Will we ever find the “One Tree” that rules them all?
- Building blocks: B+-trees, inverted lists, tries, etc.
- Indexes for node/edge-based representations (graph)
- Indexes for interval-based representations (tree)
- Indexes for path-based representations (tree)
- Indexes for sequence-based representations (tree)
- Structural indexes (graph)

ViST: a sequence-based index


- Use a sequence-based encoding for XML
- Turn twig queries to subsequence matches
- Index sequences in a virtual trie using interval-based encoding

Sequence representation of XML

- A sequence of (symbol, prefix) pairs, in depth-first order:
  - (P, ε), (S, P), (I, PS), (N, PSI), (V, PSI), (M, PSM), (L, PSL), (N, PBN)
- What is the worst-case storage requirement?
- Would listing symbols in depth-first order be sufficient?

Sequence representation of twigs

- Twigs can be represented sequences as well

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Structure-Encoded Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁: (Purchase)[Sold] (Item) (Manufacture)</td>
<td>(P × (S, P), L, PSI, M, PSI)</td>
</tr>
<tr>
<td>Q₂: (Purchase) (Item) [name = n₁] (Order) [far = n₂]</td>
<td>(P × (S, P), L, PSI, M, PSI, (R, P), (L, PBL), (N, PB), (B, P), (N, PB), (B, PBN))</td>
</tr>
<tr>
<td>Q₃: (Purchase) (Item) [name = n₁]</td>
<td>(P × (L, PBL), P × (B, PBN))</td>
</tr>
<tr>
<td>Q₄: (Purchase) (Item) (Manufacture) [name = n₁]</td>
<td>(P × (M, PBL), P × (BBN))</td>
</tr>
</tbody>
</table>
Matching twigs as sequences

- Data: \( (P, \varepsilon), (S, P), (I, PS), (N, PSI), (M, PSI), (I, PSI), (M, PSI), (I, PSI), (N, PSI) \)
- Query (Boston seller New York buyer): \( (P, \varepsilon), (S, P), (I, PS), (N, PSI) \)
- Find a (non-contiguous) subsequence of data that matches the query

False alarms

- /P/Q[T]/S
- Match sequences for /P/Q[T]/S and /P/[Q/T]/Q/S
- Compute the difference between the answers
- But what if a document exhibits both structures?

Indexing sequences with a trie

- Just insert sequences into a trie
- Search the trie for subsequences matching the query
- Expensive because subsequences do not need to be contiguous
- \( D_{\text{twigs}}: (P, S, P), (I, PS), (N, PSI), (M, PSI) \)
- \( D_{\text{twigs}}: (P, S, P), (I, PS), (N, PSI) \)
- \( Q: (P, S, P), (I, PS), (N, PSI) \)
- \( Q: (P, S, P), (I, PS), (N, PSI) \)

Virtual trie” idea

- Use \( \text{left}, \text{size} \) to encode trie nodes
- \( \text{size} = \text{right} - \text{left} \)
- Supports efficient “skipping”
- Index in a regular \( B^+ \)-tree
- No need to store the trie itself

ViST structures

- D-Ancestor \( B^+ \)-tree indexes trie nodes by \( \text{symbol}, \text{prefix} \)
  - Facilitates prefix matching (checking for ancestor-descendent relationships in documents)
- Leaf nodes point to S-Ancestor \( B^+ \)-trees, which further index nodes by \( \text{left}, \text{size} \)
  - Facilitates skipping in the trie (checking for ancestor-descendent relationships in the trie)
- Subsequence matching \( \rightarrow \) repeated index lookups

Lore’s DataGuide: a structural index


- Given an XML data graph \( G \), a DataGuide is an index graph \( I \) with the following properties
  - Every label path in \( G \) also occurs in \( I \)
    - Complete coverage
  - Every label path in \( I \) also occurs in \( G \)
  - Accurate coverage
  - Every label path in \( I \) (starting from a particular object) is unique (i.e., \( I \) is a DFA)
    - Efficient search: a label path of length \( v \) traverses \( v \) edges and ends at one node
  - Each index node in \( I \) points to its extent: a set of data nodes in \( G \)
    - Label path query on \( G \) \( \rightarrow \) label path query on \( I \)
Strong DataGuide

- Let $p$, $p'$ be two label path expressions and $G$ a graph; define $p \equiv_G p'$ if $p(G) = p'(G)$
  - That is, $p$ and $p'$ are indistinguishable on $G$
- $I$ is a strong DataGuide for a database $G$ if the equivalence relations $\equiv_I$ and $\equiv_G$ are the same

**Example**
- $I_1$ is strong; $I_2$ is not
  - $A.C(I_1) = \{ 5 \}$, $B.C(I_2) = \{ 6, 7 \}$
  - Not equal
  - $A.C(I_2) = \{ 20 \}$, $B.C(I_2) = \{ 20 \}$
  - Equal

Size of DataGuides

- If $G$ is a tree, then $|I| \leq |G|
  - Linear construction time
- In the worst case, the size of a strong DataGuide may be exponential in $|G|$ because of the DFA requirement

1-index

- Milo & Suciu, "Index Structures for Path Expressions." ICDT, 1997
- “Perfect” equivalence relation: two data nodes are equivalent iff they are not distinguishable by label path expressions
  - That is, the sets of label path expressions that can reach them are the same
  - Too expensive to compute in practice
- 1-index uses a less perfect equivalent relation, bisimilarity, which is easier to compute
  - If two nodes are bisimilar, then they are not distinguishable by label path expressions
  - The converse is not necessarily true
  - May result in larger indexes

1-index construction

- Initialize the index
  - Data nodes with the same label go into the same index node
- Pick an index node $v$ to apply a split operation
  - For each index node $v$, split it into $v_1$ and $v_2$ (if both have non-empty extents)
    - $v_1$.extent contains data nodes in $v$.extent that are children of $v$.extent
    - $v_2$.extent contains the rest of $v$.extent
- Repeat split until there is no more change to the index