XML Indexing II

CPS 216
Advanced Database Systems

Announcements (April 6)

♦ Welcome back!
♦ Homework #3 due tonight

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)
Index Fabric: a path-based index

Cooper et al. "A Fast Index for Semistructured Data." VLDB 2001

- Use a label-path encoding for XML
  - Each element is associated with a sequence of labels on the path from the root (e.g., /Invoice/Buyer/Name/ABC Corp.)
  - Encode the label path as a string (e.g., /Invoice/Buyer/Name → αβδ)
- Index all label paths in a Patricia trie
  - And try to make the trie balanced and I/O-efficient

Example of label paths in Index Fabric

-balanced Patricia trie in Index Fabric

- Recall that Patricia trie indexes first point of difference between keys
- Divide trie into blocks
- Build another layer
Searching Patricia trie in Index Fabric

- Start searching in the root layer
- One block access per layer
- Example: "greenbeans"

Refined paths in Index Fabric

- Queries supported by Index Fabric so far:
  - Label paths from the root (e.g., /Invoice/Buyer/Name/)
  - How about //Buyer/Name, or //Buyer/Name|Address?
- Refined paths: frequent queries
  - Just invent labels for these queries and index them in the same Patricia trie
  - Example: find invoices where X sold to Y

Extra refined paths → more space required

ViST: a sequence-based index

Wang et al. "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures."

- 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 \((\text{symbol, prefix})\) pairs, in depth-first order:
  - \((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN)\)

- 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>(Q_1)</td>
<td>((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN))</td>
</tr>
<tr>
<td>(Q_2)</td>
<td>((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN))</td>
</tr>
<tr>
<td>(Q_3)</td>
<td>((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN))</td>
</tr>
<tr>
<td>(Q_4)</td>
<td>((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN))</td>
</tr>
</tbody>
</table>

Matching twigs as sequences

- Data: \((P, \varepsilon), (S, P), (I, PS), (N, PSI), (v_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSI), (v_3, PSIM), (I, PSI), (M, PSI), (v_4, PSIN), (L, PS), (v_5, PSL), (N, PS), (v_6, PSN), (B, P), (L, PB), (v_7, PBL), (N, PB), (v_8, PBN)\)
- Query (Boston seller New York buyer): \((P, \varepsilon), (S, P), (I, PS), (v_5, PSL), (B, P), (L, PB), (v_7, PBL)\)
- Find a (non-contiguous) subsequence of data that matches the query
False alarms

Match sequences for /P/Q(T)/S and /P/(Q/T)/Q/S
Compute the difference between the answers
But what if a document exhibit 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

“Virtual trie” idea

Use (left, size) to encode trie nodes
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 (symbol, prefix)
  - Facilitates prefix matching (checking for ancestor-descendent relationships in documents)
- Leaf nodes point to S-Ancestor B⁺-trees, which further index nodes by (left, size)
  - Facilitates skipping in the trie (checking for ancestor-descendent relationships in the trie)
- Subsequence matching → 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 n traverses n 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 → 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
    - $A(G) = \{ 5 \}, B(G) = \{ 6, 7 \}$
      - Not equal
    - $A(I_1) = \{ 20 \}, B(I_1) = \{ 20 \}$
      - 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

```
  A
 / \
A    
 / \
B    B
 / \
B    B
 / \
B    B
```

- Relax the DFA requirement?

NFA-based structural indexes

- Defined using an equivalence relation (based on the graph structure)
  - Each index node $v$ corresponds to an equivalence class of data nodes in $G$ (denoted $v.extent$)
  - There is a edge from $u$ to $v$ in $I$ iff there exists a edge from a node in $u.extent$ to a node in $v.extent$

- $|I| \leq |G|$ by definition because extents do not overlap; however, the structure is no longer a DFA

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