Announcements (April 6)
- Welcome back!
- Homework #3 due tonight

XML Indexing II

CPS 216
Advanced Database Systems

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 \( \rightarrow \alpha\beta\delta\) )
- 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

Balancing 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

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_1, PSIN), (M, PSI), (v_2, PSIM), (I, PSI), (M, PSII), (v_3, PSIIM), (I, PS), (N, PSI), (v_4, PSIN), (L, PS), (N, PS), (v_5, PSN), (B, P), (L, PB), (v_6, PBL), (N, PB), (v_6, 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

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

\[
\begin{align*}
D_1 &= (P, T, S, Q) \cup (P, Q, T, S, P) \cup (R, P, U, P, R) \cup (T, P, R) \\
D_2 &= (P, T, Q, S, P) \cup (Q, T, S, P) \\
Q &= (P, S, Q, T, P) \cup (Q, T, S, P)
\end{align*}
\]

\( /P/Q(T)/S \)

- 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

ViST structures

- D-Ancestor B*-tree indexes trie nodes by (symbol, prefix)
  - Facilitates prefix matching (checking for ancestor-descendant 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-descendant 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 \( x \) traverses \( x \) 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 \( p \equiv_c p' \) 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 \( \equiv_c \) and \( \equiv_c \) are the same

Example

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

- 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 \( v . \text{extent} \))
  - There is an edge from \( u \) to \( v \) in \( I \) iff there exists an edge from a node in \( u . \text{extent} \) to a node in \( v . \text{extent} \)
  - \( |I| \leq |G| \) 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 equivalence 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 necessary true
  - \( \Rightarrow \) 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 \( u \) to apply a split operation
  - For each index node \( v \), split it into \( v_1 \) and \( v_2 \) (if both have non-empty extents)
    - \( v_1 . \text{extent} \) contains data nodes in \( v . \text{extent} \) that are children of \( u . \text{extent} \)
    - \( v_2 . \text{extent} \) contains the rest of \( v . \text{extent} \)
  - Repeat split until there is no more change to the index