XML Indexing I

CPS 216

Advanced Database Systems

Announcements (March 25)

- Course project milestone 2 due next Tuesday
- Homework #3 due on April 6
- Recitation session this Friday
  - XML API's
- No classes next week
  - Make up during reading period

XML indexing overview

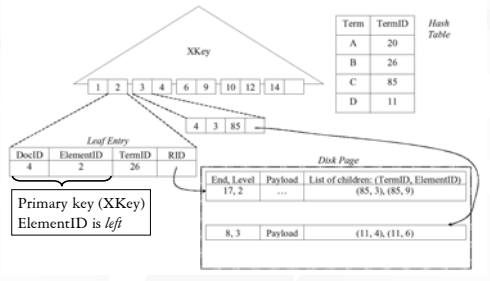
- It is a jungle out there
  - Different representation scheme lead to different indexes
  - Will we ever find the "One Tree" that rules them all?
- Building blocks: B^+-trees, inverted lists, tries, etc.
- Indexes for node/edge-based representations (graph)
- Indexes for interval-based representations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

Warm-up: indexes in Lore (review)

- Label index: (child, label) → parent
  - B^+-tree
- Edge index: label → (parent, child)
  - B^+-tree
- Value index: (value, label) → Node
  - B^+-tree
- Path index: path expression → node
  - Structural index: DataGuide (more in next lecture)

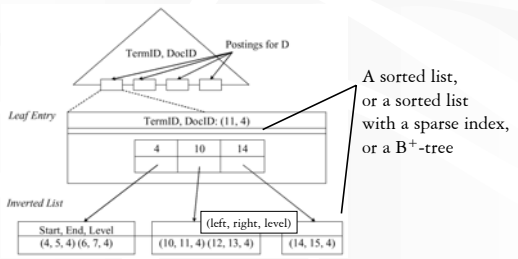
Niagara: data manager index

- A combination of node/edge-based and interval-based representations using B^+-tree



Niagara: index manager index

- Essentially an inverted-list index for tag names with entries in each list sorted by XKey



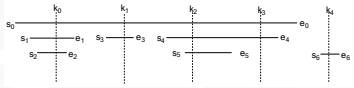
XR-tree

Stands for XML Region Tree (Jiang et al., *ICDE* 2002)

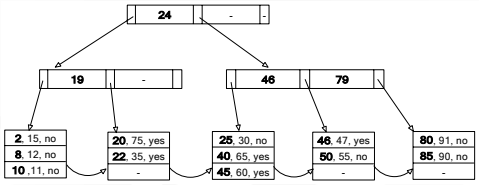
- Intended for interval-based representation
- Based on B^+-tree
- Nice property: given an element, all its ancestors/descendents can be identified very efficiently

XR-tree structure

- Backbone is a B^+-tree with *left* as the index key
- Each internal index node *n* maintains a stab list *SL*(*n*)
  - An element is in *SL*(*n*) if it is
    - "Stabbed" by at least one key in *n*, i.e., that key in contained in the element's (*left*, *right*)
    - Not stabbed by any key in *n*'s ancestor
- For each key within an internal node *n*, also store (*first_left*, *first_right*), from the first element in *SL*(*n*) stabbed by this key but not by any previous keys in *n*
  - Example: (s_0, e_0) for k_0; (s_4, e_4) for k_2; (nil, nil) for k_3

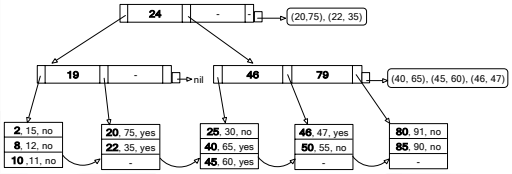


The backbone B^+-tree



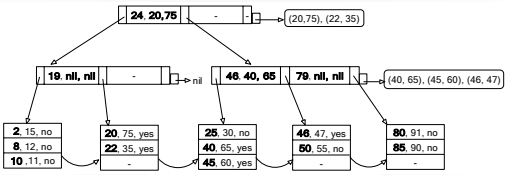
- Entries in leaf index nodes have the form (*left*, *right*, *InStabList*, *pointer_to_record*)
  - InStabList is set to true iff the entry can be found some stab list

Stab lists



- Each internal node maintains a stab list
- An element can be in at most one stab list
- Some internal nodes may have empty stab lists (nil)

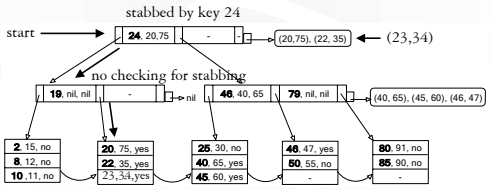
(*first_left*, *first_right*) fields



- Note that keys 19, 79 have nil (*first_left*, *first_right*)

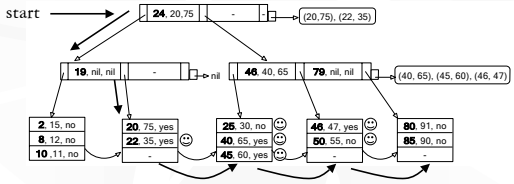
XR-tree insertion example

- Insert (23, 24)



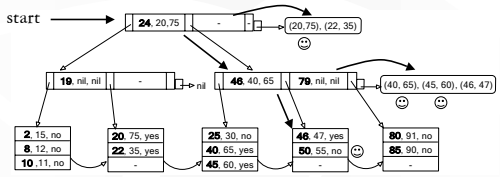
Looking up descendents

- Basically a range query over the backbone B$^+$-tree
- Example: descendents of (21, 74)



Looking up ancestors

- Go down the tree and check stab lists and the leaf
- Example: ancestors of (51, 52)
  - Just look for all intervals stabbed by 51
  - Need to check 52?
  - Need to check stab lists on other paths?

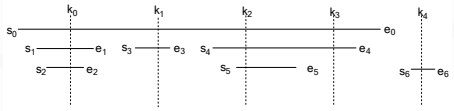


Note that leaves with "yes" are ignored

Stop.

Stab list checking in more detail

- Visually, the stab list for an internal index node can be seen as stacks of intervals, one stack for each key in the node



- If *left* falls in between k_i and k_{i+1}, only need to check from the first to the $(i+1)$-th stack (why?)
- For each stack, check bottom-up, and stop whenever the interval is no longer stabbed by *left* (why?)
- (*start_left*, *start_end*) ensures that no stack is checked unnecessarily

Performance of XR-tree

- Space: linear in the size of the XML document
- Time
  - h_{tree}: B$^+$-tree height; R: result size; B: block size
  - Looking up descendents: $O(h_{tree}+R/B)$ in the worst case
  - Looking up ancestors: $O(h_{tree}+R)$ in the worst case
    - Loss of $1/B$ factor is worrisome
    - R in this case can be up to h_{xml}, the height of the XML tree
  - Insert/delete: $O(h_{tree}+c)$, amortized

Discussion on XR-tree

- Plain B$^+$-tree works fine for descendents
- Lots of machineries just to find all ancestors
  - Maintaining back pointers allow ancestors to be retrieved in h_{xml} I/O's, matching the bound for XR-tree!
    - Perhaps XR-tree works better on the average case?
  - It should be possible to answer stabbing queries in $O(h_{tree}+R/B)$ time and beat XR-tree and back pointers, even with arbitrary intervals