Introduction to Data Management

Lecture 20
(Storage and Indexing, cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

It’s time again for....

Friday Nights with Databases...!

Brought to you by...
Announcements

- Midterm #2 is **Wednesday (5/22) at 5 PM**
  - Relational languages (see syllabus!)
  - Sample exam from last year is available
  - Assigned seating, similar to last time
- HW #6 is due on **Monday at 7 PM**
  - One late “day” (**22 hours**) will be available
  - Solution coming Tuesday right after **5 PM** (**really**)?
- Today’s lecture plan
  - More about database indexes
  - (Not on Midterm #2, of course)

Tree-Structured Indexes: Over(re)view

- **As for any index, 3 alternatives for data entries (k*)**:
  - Record with key k
  - <k, rid of record with key k>
  - <k, list of rids of records with key k>
- This data entry choice is orthogonal (⊥) to the indexing technique used to locate the data entries.
- Tree-structured indexing techniques can support both **range searches** and **equality searches**.
- **ISAM**: static structure; **B+ tree**: dynamic, adjusts gracefully under inserts and deletes.
Example ISAM Tree

- Suppose each node can hold 2 entries (really more like 200, since nodes are disk pages!)

After Inserting 23*, 48*, 41*, 42* ...

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
... Then Deleting 42*, 51*, 97*

Note that 51* still appears in index levels, but not in leaf!

B+ Tree: Most Widely Used Index!

- Insert/delete at \( \log_F N \) cost; keep tree height-balanced. (\( F = \) fanout, \( N = \) # leaf pages)
- Minimum 50% occupancy (except for root).
  Each node contains \( d \leq m \leq 2d \) entries.
  The (mythical) \( d \) is called the order of the B+ tree.
- Supports equality and range-searches efficiently.
Example B+ Tree

- Search begins at root, and key comparisons direct the search to a leaf (as in ISAM).
- Ex: Search for 5*, 15*, all data entries >= 24*, ...

Based on the search for 15*, we know it is not in the tree!

B+ Tree: Most Widely Used Index!

- Insert/delete at log \(_F\) N cost; keep tree height-balanced. (\(F = \text{fanout}, N = \# \text{leaf pages}\)
- Minimum 50% occupancy (except for root).
  Each node contains \(d \leq m \leq 2d\) entries.
  The (mythical) \(d\) is called the order of the B+ tree.
- Supports equality and range-searches efficiently.
Example B+ Tree

- Search begins at root, and key comparisons direct the search to a leaf (as in ISAM).
- Ex: Search for 5*, 15*, all data entries >= 24*, ...

Based on the search for 15*, we know it is not in the tree!

Example B+ Tree (a clarification)

- Search begins at root, and key comparisons direct the search to a leaf (as in ISAM).
- Ex: Search for 5*, 15*, all data entries >= 24*, ...

Based on the search for 15*, we know it is not in the tree!
Inserting a Data Entry into a B+ Tree

- Find correct leaf \( L \) (by searching for the new \( k \)).
- Put new data entry (\( k^* \), a.k.a. \( (k, I(k)) \)) in leaf \( L \).
  - If \( L \) has enough space, done! (Most likely case!)
  - Else, must split \( L \) (into \( L \) and a new node \( L_2 \))
    - Redistribute entries evenly and copy up middle key.
    - Insert new index entry pointing to \( L_2 \) into parent of \( L \).
- This can happen recursively.
  - To split an index node, redistribute entries evenly but push up the middle key. (Contrast with leaf splits!)
- Splits “grow” tree; root split increases its height.
  - Tree growth: gets wider or one level taller at top.

Inserting \( 8^* \) into Example B+ Tree

- Observe how minimum occupancy is guaranteed in both leaf and index pg splits.
- Note difference between copy-up and push-up; be sure you understand the reasons for this!

Entry to be inserted in parent node. (Note that 5 is copied up and continues to appear in the leaf.)

Entry to be inserted in parent node. (Note that 17 is pushed up and only appears once in the index. Contrast this with a leaf split.)
Example B+ Tree After Inserting $8^*$

- Notice that root was split, leading to increase in height.
- In this example, could avoid split by redistributing entries; however, not usually done in practice. (Q: Why is that?)

Let's Go Live…! (Demo Time!)

Note (see Piazza): Very cool online B+ tree viz tool available (😊)
- [https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html](https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html)
- Only slight differences from our defs (e.g., key 13 above $\rightarrow$ 14)
- Their “Max. Degree” is our $2d+1$ (limit of 5 pointers/node above)
Deleting a Data Entry from a B+ Tree

- Start at root, find leaf $L$ where entry belongs.
- Remove the entry.
  - If $L$ is still at least half-full, done!
  - If $L$ has only $d-1$ entries,
    - Try to redistribute, borrowing from sibling (adjacent node with same parent as $L$).
    - If re-distribution fails, merge $L$ and sibling.
- If merge occurred, must delete search-guiding entry (pointing to $L$ or sibling) from parent of $L$.
- Merge could propagate to root, decreasing height.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

Example Tree After (Inserting 8*, Then) Deleting 19* and 20* ...

- Deleting 19* is easy.
- Deleting 20* is done with redistribution. Notice how middle key is copied up.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
... And Then Deleting 24*

- Must merge.
- Observe “toss” of index entry (on right), and “pull down” of index entry (below).