Tuesday 21/4/2015: Tree-Structured Indexing

Lecture Topics
I. Review and Intuition
II. ISAM
III. B+ Trees
IV. Search
V. Insert
VI. Delete
VII. Bulk Loading

II. ISAM Trees

ISAM = Indexed Sequential Access Method

- Records are stored sequentially
- Indexes are small, can be searched quickly
- Older systems stored pointers to other data within the records
- Index nodes are fixed: do not change with insertion or deletion
- If insert exceeds node capacity, then overflow page is used
- Over time, overflow gets bigger, access time increases

> Note: if 51 is deleted, index stays the same

Search: \( \log N \); \( F = \# \text{entries} / \text{index page} \)
\( N = \# \text{leaf pages} \)

Insert: Find leaf, put it in
Delete: Find and remove, if empty overflow, de-allocate

I. Review and Intuition

Recall: (logical view) File is a sequence of records. Records are fixed or variable size.

(physical view) File is a sequence of blocks/page. Blocks are fixed size, non-contiguous

- To answer a query:
  - Read blocks with relevant records
  - Read all blocks into RAM
  - Get relevant data from block (additional processing)

- Remember: must read a block

Example: "Find all students with GPA ≥ 9.0"
Assume data is sorted by GPA in a file
- Could do binary search, then scan
- Could be sorted on larger data file

- JOB: Create a smaller file: "index"
### B+ Trees

- Generalizes 2-3 trees
- It is rooted
- It is directed (order of children matter)
- All paths from root to leaves are the same length (balanced)
- For some parameter $m$
  - All internal nodes have between \( \lceil m/2 \rceil \) and $m$ children
  - The root has between 2 and $m$ children
- Internal nodes contain keys and pointers

**Diagram:**

```
      13 17 24 30
    /  \  /  \
   5   7 16  9
  / \  / \  / \  /
3  5 11 19 22 27 29
```

- Note: 2-3 tree is a B+ tree with $m = 3$

### Important properties:

- For any value $N$, and $m \geq 3$, there is always a B+ tree storing $N$ pointers in the leaves
- Possible to maintain above for insert/delete
- For such operations, only the depth of the tree need be manipulated

- Depth is $\log_{m/2} N$

**What is the best $m$?**

- In RAM, best $m = 3$. Why? Think of $m = N$, then sorted sequence. On disk...
Example:
- File with 20,000,000 records
- \( m = 57 \)
- Dense index, unclustered file.
- How big is the tree?
  - Root: 2-57 pointers
  - Non-Root: 29-57 pointers

Narrowest tree: every node has 29 children

Widest tree: every node has 57 children

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes in narrowest</th>
<th>Nodes widest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>57</td>
</tr>
<tr>
<td>3</td>
<td>58</td>
<td>3,249</td>
</tr>
<tr>
<td>4</td>
<td>1,482</td>
<td>185,193</td>
</tr>
<tr>
<td>5</td>
<td>48,778</td>
<td>10,556,001</td>
</tr>
<tr>
<td>6</td>
<td>1,414,562</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>41,022,298</td>
<td></td>
</tr>
</tbody>
</table>

Need: 20,000,000 pointers

For narrowest tree:
If we had 6 levels,
\( 1,414,562 \times 29 = 41,022,298 \)
Too big! 5 levels, add more children to some nodes

For widest tree:
If we had 4 levels:
\( 185,193 \times 57 = 10,556,001 \)
So we need 5 levels

Note: these will not always be the same

If we want to find 10 (random) records?
- 6 block accesses per record
- 60 block accesses for all 10
  - if sorted, may be better
IV Search
- Start at root
- Key comparisons direct to leaf
- $O(\log \frac{m}{\log m - 1})$

IV Insert
- Find correct leaf $L$
- Put data into $L$
  - If $L$ has enough space, done!
  - Else, must split $L$ (into $L_1$ and $L_2$)
    - Redistribute entries evenly
    - Copy up middle key
    - Insert index entry pointing to $L_2$
      into parent of $L$
    - This can happen recursively
    - To split index node,
      redistribute entries evenly
      but push up middle key
- Split grows tree
- Each split increases height

Example: Insert 8

```
5
| 2 | 3 |
```

5 is copied up, but remains in leaf

```
2
5 17 23
```

17 is pushed up.
Index appears only once
**Delete**

\[ d = \frac{m}{2} \]

called the 'order' of the tree

- Start at root, find leaf \( L \) where entry belongs
- Remove the entry
  - If \( L \) is at least half full, done!
  - If \( L \) has only \( d-1 \) entries:
    - try to re-distribute, borrow from sibling (adjacent node with same parent)
    - If re-distribution fails, merge \( L \) and sibling
  - If merge occurred, must delete entry (point to \( L \) or sibling) from parent.
- Merge could propagate, decrease the height.

**Example Delete** \( 19^*, 20^* \)

- Delete \( 19 \), easy
- Delete \( 20 \), re-distribution
  - middle key is copied up
- Delete 24

Must merge:
- Index entry is "tossed"
- Index entry is "pulled down"

VII Bulk loading

1. Sort all data entries
2. Insert pointer to first leaf in a new root

- may "split" as you go up
- Entries always into rightmost indexed page just above leaf level
- Much faster than repeated inserts