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

I. Review and Intuition

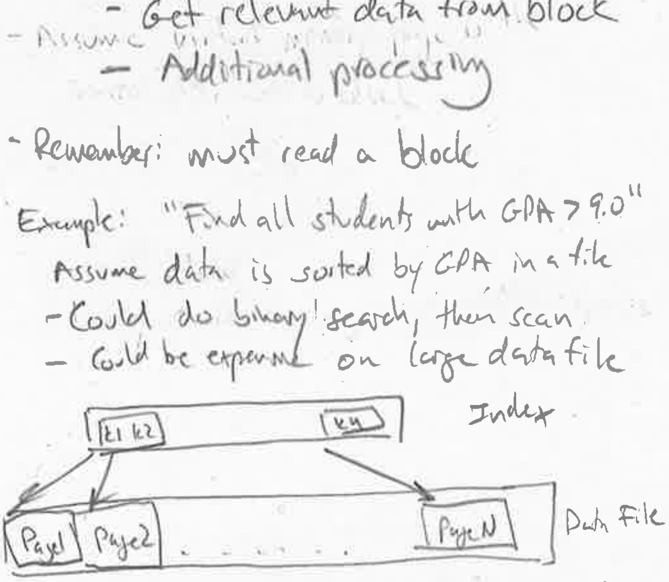
Recall: (logical view) File is a sequence of records.
Records are fixed or variable size.

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

- To answer a query:
  - Find blocks with relevant records
  - Read all blocks into RAM
  - Get relevant data from block
  - Additional processing

- Remember: must read a block

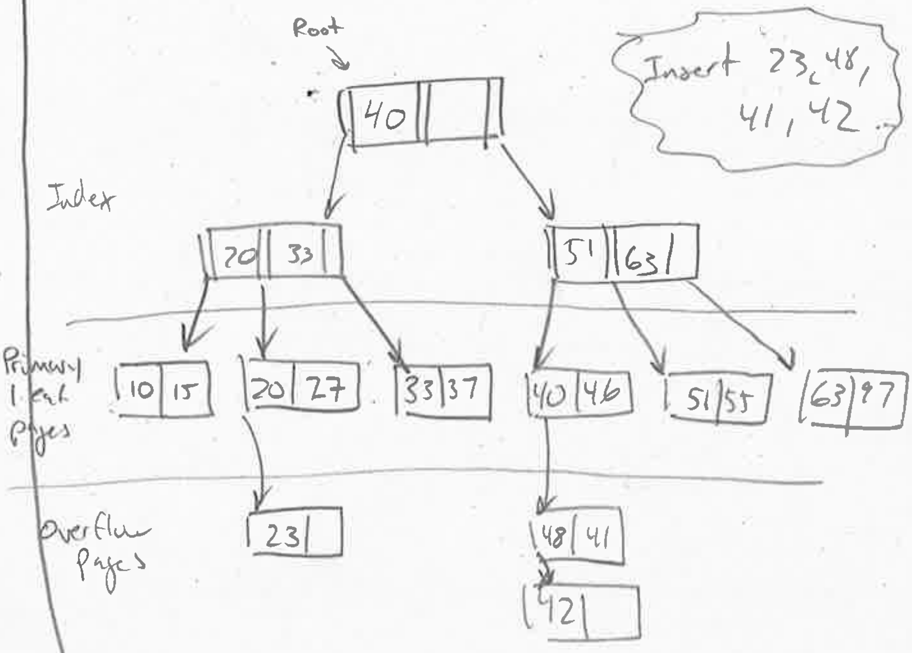
Example: "Find all students with GPA > 9.0"
Assume data is sorted by GPA in a file
- Could do binary search, then scan
- Could be expensive on large data file



- IDEA: CREATE A smaller file: "index"

II ISAM TREE

ISAM = Indexed Sequential Access Method



- Records are stored sequentially
- Indexes are small, can be seached 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 over flow page is used
- over time, overflow gets bigger, access time increases

→ Note if 51 is deleted, index stays the same

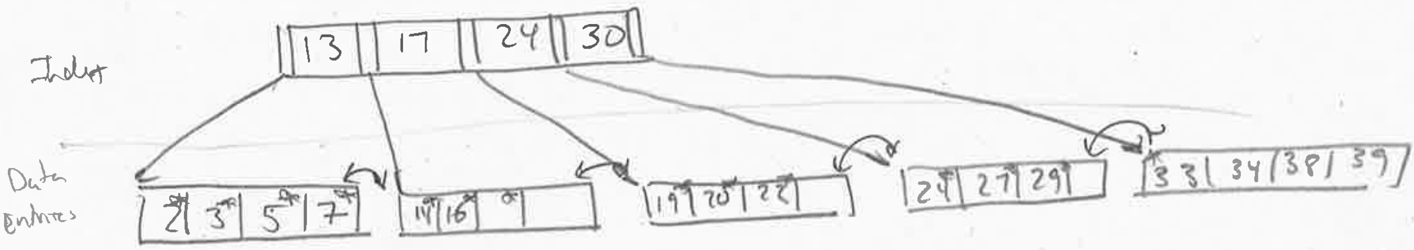
Search: $\log_F N$; F = # entries / index pg
N = # leaf pages.

Insert : Find leaf, put it in

Delete : Find and Remove. if empty overflow, de-allocate

III 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



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

Important properties:

- For any value N, and m ≥ 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 $\log_{\lceil m/2 \rceil}$

- What is the best m?
- In RAM, best m=3. Why? Think of m=N, then sorted sequence. On disk... see other notes.

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

Level	Nodes in narrowest	Nodes in widest
1	1	1
2	2	57
3	58	3,249
4	1,682	185,193
5	48,778	10,556,001
6	1,414,562	
7	41,022,298	

Need: 20,000,000 pointers

For narrowest tree:

If we had 6 levels,

1,414,562 × 29 = 41,022,298

too big! 5 levels, add more children to some nodes

For widest tree

If we had 4 levels:

185,193 × 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

Tuesday 21/4/2015 Tree-structured Indices Pg 4

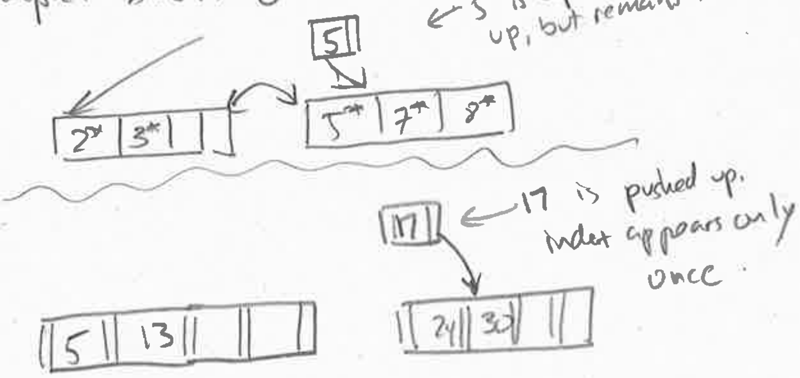
IV Search

- Start at root
- key comparisons direct to leaf
- $O(\log \lceil m/2 \rceil)$

V Insert

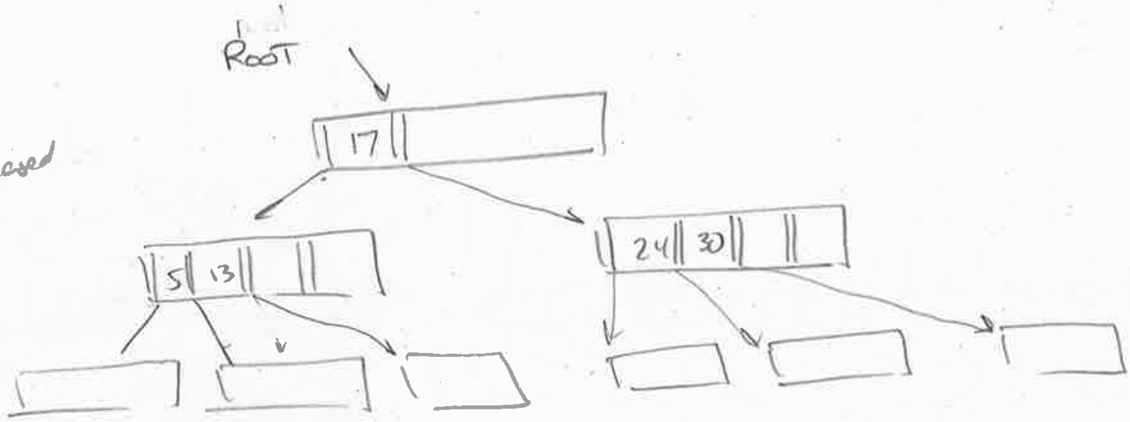
- Find correct leaf L
- Put data into L:
  - If L has enough space, done!
  - Else, must <u>split</u> L (into L and L2)
    - redistribute entries evenly <u>copy up</u> middle key
    - Insert index entry pointing to L2 into parent of L.
  - This can happen recursively
    - To split index node, redistribute entries evenly but <u>push up</u> middle key
  - Split grows tree
  - root split increases heigh

Example: Insert 8*



21/4/2015 TREE-STRUCTURED INDEXES pg 5

After insert, root split height increased



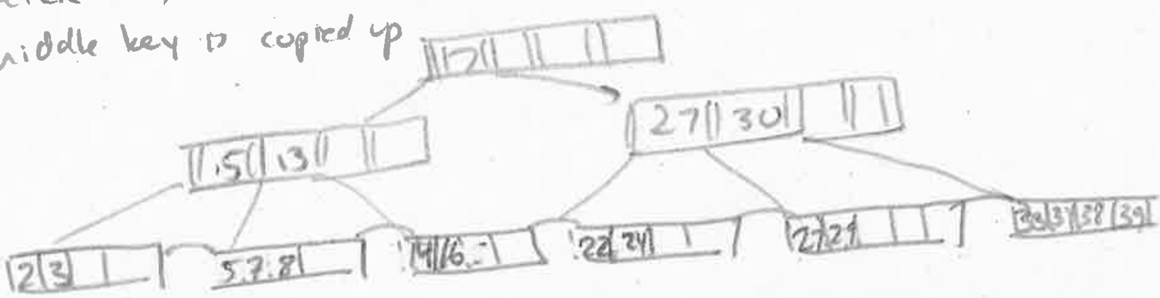
VI Delete

$d = \lceil m/2 \rceil$
called the "order" of the tree

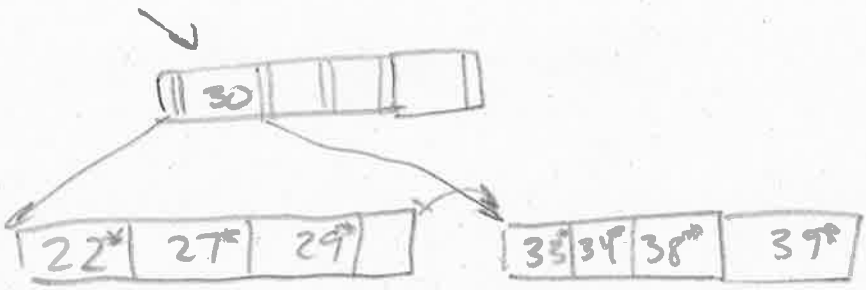
- Start at root, find leaf L where entry belongs
- Remove the entry
  - If L is at least half full, done!
  - If L has only d-1 entries:
    - try to re-distribute, borrow from sibling (adjacent node with same parent)
    - If re-distribution fails, merge L and sibling
- If merge occurred, must delete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

Example Delete 19*, 20*

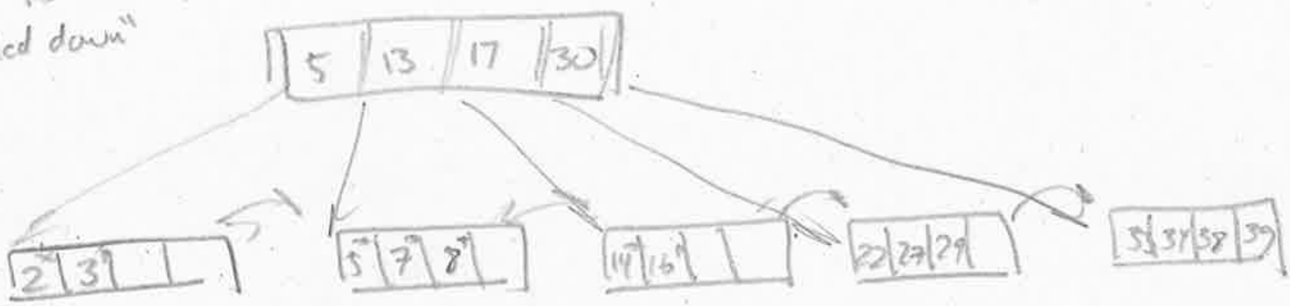
- Delete 19, easy
- Delete 20, re-distribution
- middle key is copied up



- Delete 24

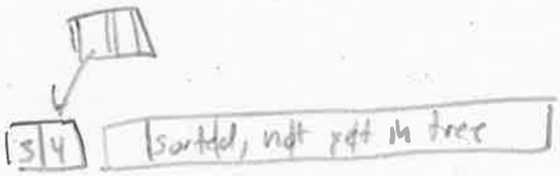


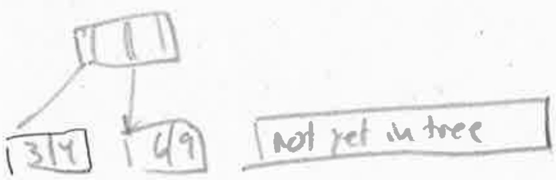
Must merge
index entry is "tossed"
and index entry is "pulled down"



VII Bulk loading

1. Sort all data entries
   / insert pointer to first leaf in a new root





- may "split" as you go up
- entries always into right-most indexed page just above leaf level
- much faster than repeated inserts.