More on Indexes

B-Trees

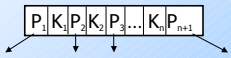
Source: our textbook,
slides by Hector Garcia-Molina

B-Trees

- Automatic multi-level index
- Features:
  - automatically adjust number of levels of indexes as size of data file changes
  - storage on blocks is managed to keep every block between half full and full => no overflow blocks needed
- We'll actually study B+ trees

B-Tree Structure

- A balanced search tree: every root-to-leaf path has same length
- each node (vertex) in the tree contains search keys $K_1 < K_2 ... < K_n$ and P_{n+1} pointers



- parameter *n* is the max number of keys in a node so that *n+1* pointers and *n* keys fit in one block
  - Ex: In practice choose n to be large. If block size is 4096 bytes, keys are 4 bytes, and pointers are 8 bytes, then *n* = 340.

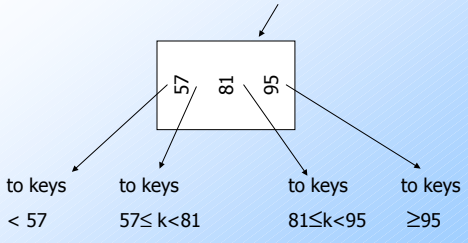
Constraints on B-Tree Nodes

- Keys in leaf nodes are in sorted order (copy keys from the data file)
- The Root contains between 2 and *n+1* index node pointers
- Each internal node must contain at least $\lceil (n+1)/2 \rceil$ keys and $\lceil (n+1)/2 \rceil + 1$ pointers (ceiling function $\lceil x \rceil$ rounds up)
- P_i points to an index node with keys K such that $K < K_i$

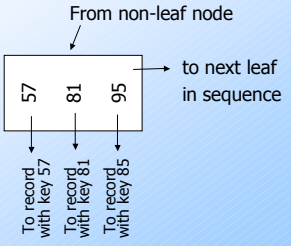
Leaf Node Constraints

- Each leaf contains at least $\lfloor (n+1)/2 \rfloor$ and at most *n data record* keys and pointers (floor function $\lfloor x \rfloor$ rounds down)
- Each leaf also has a "next leaf" pointer and possibly a "prev leaf" pointer

Sample non-leaf



Sample leaf node:

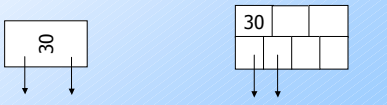


7

In textbook's notation n=3

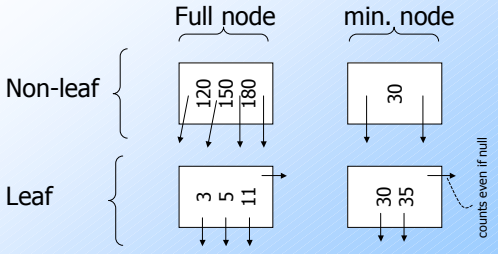
Leaf:

Non-leaf:



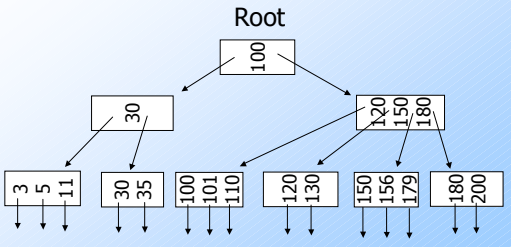
8

n=3



9

B-Tree Example n=3



10

Insert into B+tree

(a) simple case
- space available in leaf

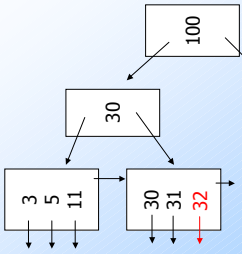
(b) leaf overflow

(c) non-leaf overflow

(d) new root

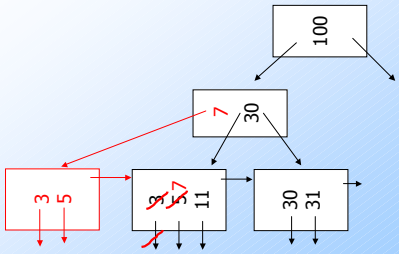
11

(a) Insert key = 32 n=3



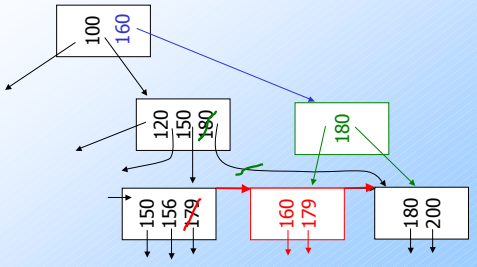
12

(a) Insert key = 7 n=3



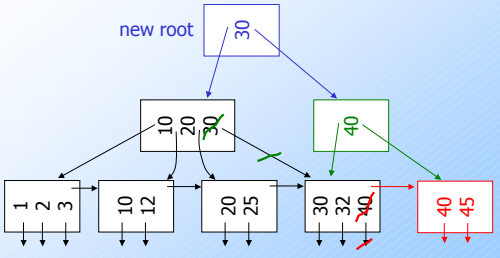
13

(c) Insert key = 160 n=3



14

(d) New root, insert 45 n=3



15

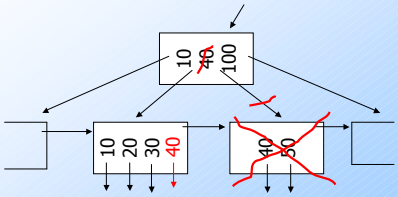
Deletion from B-tree

(a) Simple case - no example
(b) Coalesce with neighbor (sibling)
(c) Re-distribute keys
(d) Cases (b) or (c) at non-leaf

16

(b) Coalesce with sibling n=4

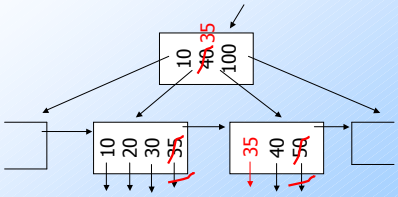
- Delete 50



17

(c) Redistribute keys n=4

- Delete 50

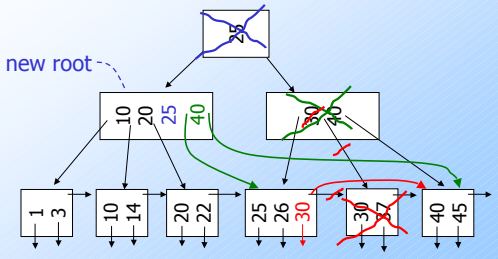


18

(d) Non-leaf coalese

- Delete 37

n=4



19

B-tree deletions in practice

- Often, coalescing is not implemented
  - Too hard and not worth it!
- Does SimpleDB implement coalesing?--No

20

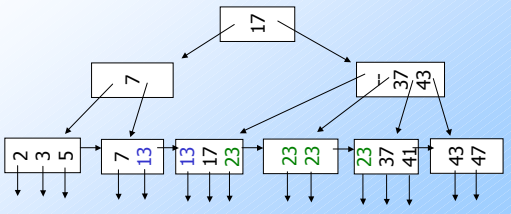
B-Trees with Duplicate Keys

Change definition of B-tree:

- If key *K* appears in an internal node, then *K* is the smallest "new" key in the subtree *S* rooted at the pointer that follows *K* in the node
- "New" means *K* does not appear in the part of the B-tree to the left of *S* but it does appear in *S*
- Allow null key in certain situations

21

Example B-Tree with Duplicates



22

Lookup in B-Trees

- Assume no duplicate keys.
- Assume B-tree is a dense index.
- To find the record with key *K*, search starting at the root and ending at a leaf:
  - if current node is not a leaf and has keys K_1, K_2, …, K_n, find the smallest key, K_i, in the sequence that is $\leq K$.
  - follow the (i+1)-st pointer to a node at the next level and repeat
  - when a leaf node is reached, find the key with value *K* and follow the associated pointer to the data record

23

Range Queries with B-Trees

- Range query: a query in which a range of values is sought. Examples:
  - SELECT * FROM R WHERE R.k > 40;
  - SELECT * FROM R WHERE R.k >= 10 AND R.k <= 25;
- To find all keys in the range [*a,b*]:
  - Do a lookup on *a*: leads to leaf where *a* could be
  - Search the leaf for all keys ≥ *a*
  - If we find a key > *b*, we are done
  - Else follow next-leaf pointer and continue searching in the next leaf
  - Continue until finding a key > *b* or no more leaves

24

Efficiency of B-Trees

- B-trees allow lookup, insertion and deletion of records with very few disk I/Os
- Number of disk I/Os is number of levels in the B-tree plus cost of any reorganization
- If n is at least 10, then splitting/merging blocks will be rare and usually limited to the leaves
- For typical sizes of keys, pointers, blocks and files, 3 levels suffice (see next slide)
- Also can keep root block of B-tree in memory

25

Size of B-Tree

- Assume
  - 4096 bytes per block
  - 4 bytes per key (e.g., integer)
  - 8 bytes per pointer
  - no header info in the block
- Then n = 340 (can keep n keys and n+1 pointers in a block)
- Assume on average a block has 255 pointers
- Count:
  - one node at level 1 (the root)
  - 255 nodes at level 2
  - 255*255 = 65,025 nodes at level 3 (leaves)
  - each leaf has 255 pointers, so total number of records is more than 16 million

26