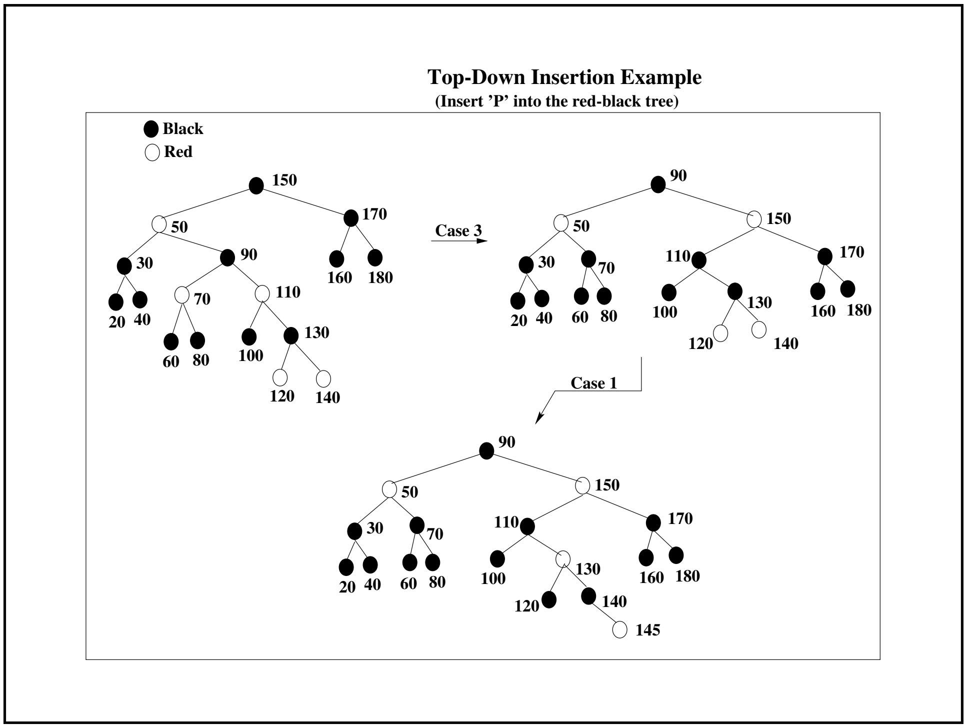
CMSC 341
Lecture 17

Announcements

Exam Monday
- same format as midterm
- material through RedBlack trees



Top Down Insertion

Make adjustments on way down

- if X has two red children, make X red and children black.
  - If P red, perform rotation to fix (P's sibling can't be red)

Insert as red node at leaf

- since parent was black, no further adjustments necessary

Top Down Deletion

Make adjustments on way down

- assure that leaf to be deleted is red

Remove node

Top Down Delete

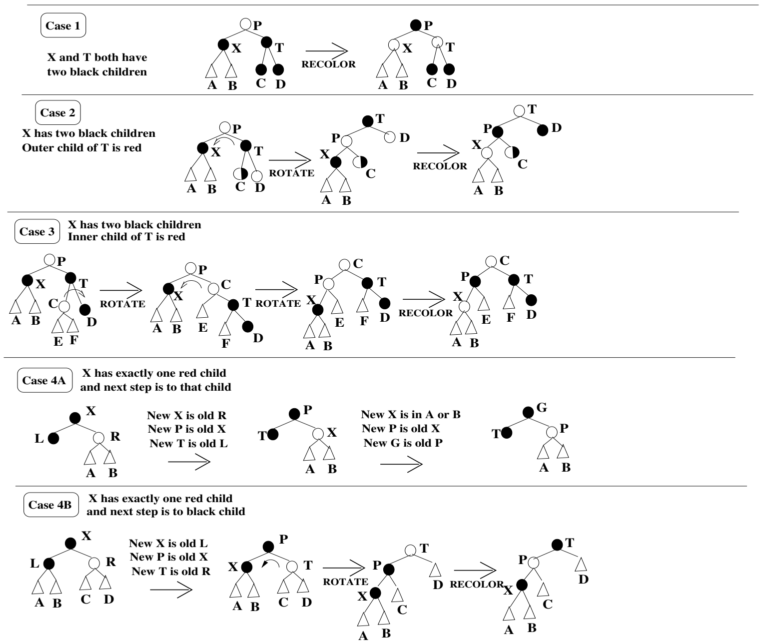
Cases:

- 1: X and T both have two black children
- 2: X has two black children, outer child of T is red
- 3: X has two black children, inner child of T is red
- 4a: X has one red child, on path to leaf
- 4b: X has one red child, away from path to leaf

Start at root

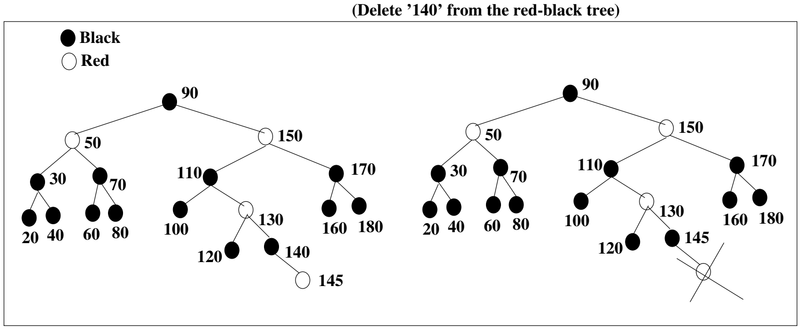
- if children are same, color root red and children black
- else treat root as X and use appropriate case

Top-Down Deletion Cases for Red-Black Trees



Top-Down Deletion Example

(Delete '140' from the red-black tree)



Proof: # Internal Nodes

Theorem 1: Any red-black tree, with root x, has at least n = $2^{bh(x)}-1$ internal nodes, where bh(x) is the black-height of node x.

Proof: by induction on height of x

- Base: x is a leaf, height is zero, bh(x) = 0, $2^0 - 1 = 0$
- Induction: Let x be an internal node with two children. If a child of x is red, its black height is bh(x). If the child is black, its black height is bh(x)-1. In any event, the IH holds for the children since their height is less than that of x. Therefore, each child subtree has at least $2^{bh(x)-1}-1$ internal nodes. Therefore, the tree rooted at x has at least

  $$2(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$$

  internal nodes

Proof: # Black

Theorem 2: In a red-black tree, at least half of the nodes on any path from root to a leaf must be black.

Proof: Since no red node can have a red child, the maximum number of red nodes in any path from root to leaf will be every other node.

Proof: Path Length

Theorem 3: In a red-black tree, no path from any node N to a leaf is more than twice as long as any other path from N to any other leaf.

Proof: By definition, every path from a node to a leaf contains the same number of black nodes. By Theorem 2, at least half of the nodes on any such path must be black. Therefore, there can be no more than twice as many red nodes on any path from a node to a leaf. Therefore, the length of every path is no more than twice as long as that of any other path.

Proof: Height

Theorem 4: A red-black tree with n internal nodes has height $n \le 2 \lg(n+1)$

Proof: Let h be the height of the red-black tree. By theorem 2, $bh(x) \ge h/2$. Therefore,

$$n \ge 2^{h/2}-1$$

$$n - 1 \ge 2^{h/2}$$

$$\lg(n-1) \ge h/2$$

$$2 \lg(n-1) \ge h$$

Large Trees

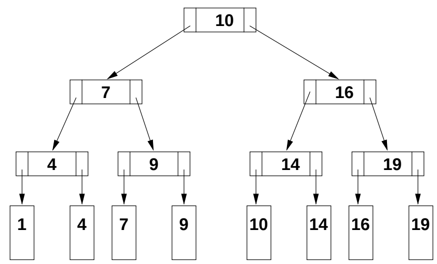
Tailored toward applications where tree doesn't fit in memory

- operations much faster than disk accesses
- want to limit levels of tree (because each new level requires a disk access)
- keep root and top level in memory

Alternative BST Model

Store data only at leaves; all leaves at same level

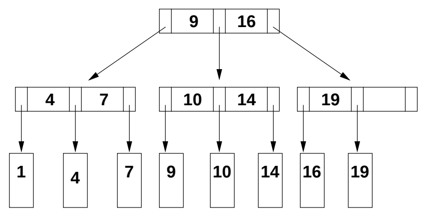
- interior and exterior nodes have different structure
- interior nodes store one key and two subtree pointers
- all search paths have same length: $\lceil \lg n \rceil$
- can store multiple data elements in a leaf



M-way Tree

Similar to previous BST model

- each interior node has M-1 keys and M subtrees pointers
- as M increases, height decreases: $\lceil \lg_M n \rceil$
- perfect M-way tree of height h has M^h leaves



Search in M-way Tree

Different from standard BST search

- search always terminates at a leaf node
- might need to scan more than one element at a leaf
- might need to scan more than one key at an interior node

Trade-offs

- tree height decreases as M increases
- computation at each node during search increases as M increases

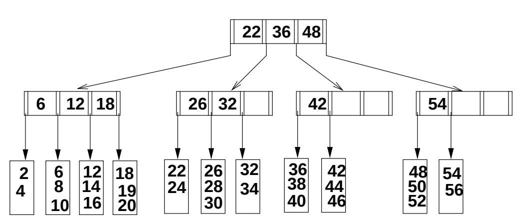
B-Trees

A B-tree of order M is an M-way tree with the following contraints:

- the root is either a leaf or has between 2 and M subtrees
- all interior nodes (except possibly root) have between $\lceil M/2 \rceil$ and M subtrees (each node is at least half full)
- all leaves are at the same level. A leaf may store between $\lceil L/2 \rceil$ and L data elements, where L is a fixed constant, $L \geq 1$

Choosing constants

- L might be chosen by number of records that fit in a disk block
- M chosen to fit M-1 keys and M pointers fit in a disk block



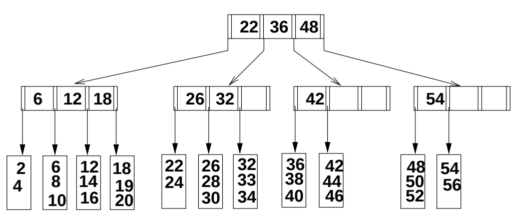
B-tree Insertion

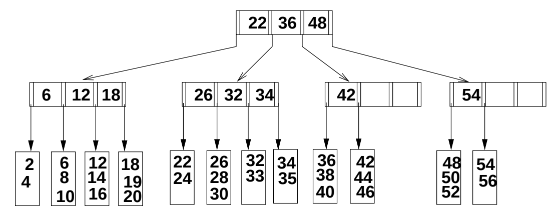
Search to find which leaf X belongs in.

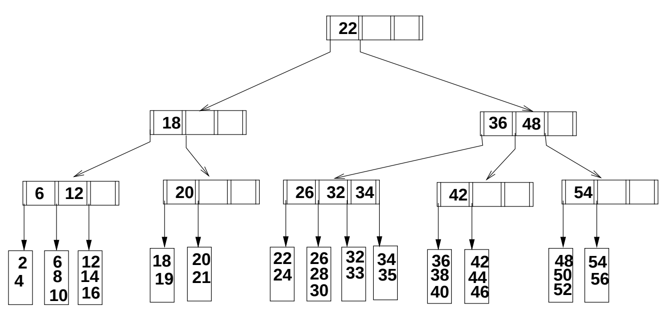
If leaf has room, add it (and write back to disk).

If leaf full, split into two leaves, each with half of elements.
(write new leaves to disk)

– if parent was already full, split in same manner
– splits may propagate all the way to the root, in which case, the root is split (this is how the tree grows in height)




22 36 48
6 12 18
26 32 34
42
54
2 4
6 8 10
12 14 16
18 19 20
22 24
26 28 30
32 33
34 35
36 38 40
42 44 46
48 50 52
54 56


22
18
36 48
6 12
20
26 32 34
42
54
2 4
6 8 10
12 14 16
18 19
20 21
22 24
26 28 30
32 33
34 35
36 38 40
42 44 46
48 50 52
54 56

B-tree Deletion

Find leaf containing element to be deleted.

If that leaf is still full enough, write it back to disk without that element.

If leaf is now too empty, borrow an element from a neighbor.

- If neighbor would be too empty, combine two leaves into one.
- Continue to combine nodes as long as necessary.