

B-Trees (4.7 in Weiss)

CSE 373
Data Structures & Algorithms
Ruth Anderson

11/21/2011

1

Today's Outline

- **Admin:**
 - Midterm #2 – Friday Nov 18th, topic list has been posted
 - HW #5 – Graphs, partners due Wed 23 at 11pm, due Thurs Dec 1 at 11pm
- **Graphs**
 - Minimum Spanning Trees
- **Dictionaries**
 - B-Trees

11/21/2011

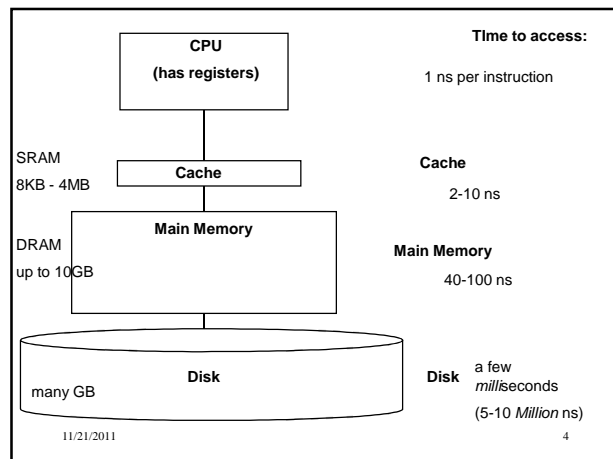
2

Trees so far

- BST
- AVL

11/21/2011

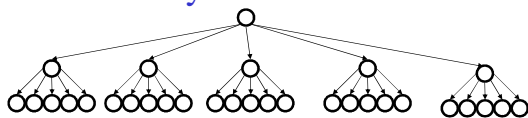
3



11/21/2011

4

M-ary Search Tree



- Maximum branching factor of M
- Complete tree has height =

disk accesses for *find*:

Runtime of *find*:

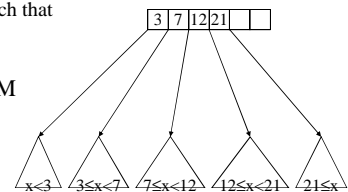
11/21/2011

5

Solution: B-Trees

- specialized M -ary search trees
- Each **node** has (up to) $M-1$ keys:
 - subtree between two keys x and y contains leaves with *values* v such that $x \leq v < y$

- Pick branching factor M such that each node takes one full {page, block} of memory



11/21/2011

6

B-Trees

What makes them disk-friendly?

1. Many keys stored in a node

- All brought to memory/cache in one access!

2. Internal nodes contain *only* keys;

Only leaf nodes contain keys and actual data

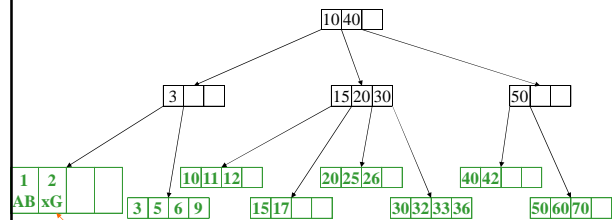
- The tree structure can be loaded into memory irrespective of data object size
- Data actually resides in disk

11/21/2011

7

B-Tree: Example

B-Tree with $M = 4$ (# pointers in internal node)
and $L = 4$ (# data items in Leaf)



Data objects, that I'll ignore in slides

11/21/2011

Note: All leaves at the same depth!

8

B-Tree Properties ‡

- Data is stored at the **leaves**
- All **leaves** are at the same depth and contain between $\lceil L/2 \rceil$ and L data items
- Internal** nodes store up to $M-1$ keys
- Internal** nodes have between $\lceil M/2 \rceil$ and M children
- Root** (special case) has between 2 and M children (or root could be a leaf)

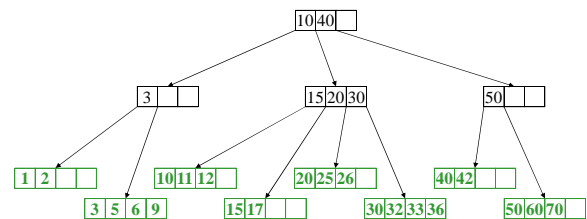
11/21/2011

‡These are technically B⁺-Trees

9

Example, Again

B-Tree with $M = 4$
and $L = 4$



(Only showing keys, but leaves also have data!)

11/21/2011

10

B-trees vs. AVL trees

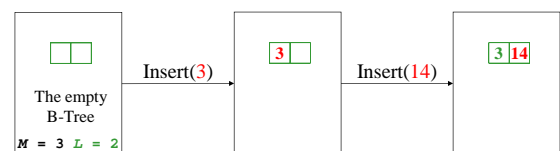
Suppose we have 100 million items (100,000,000):

- Depth of AVL Tree
- Depth of B+ Tree with $M = 128$, $L = 64$

11/21/2011

11

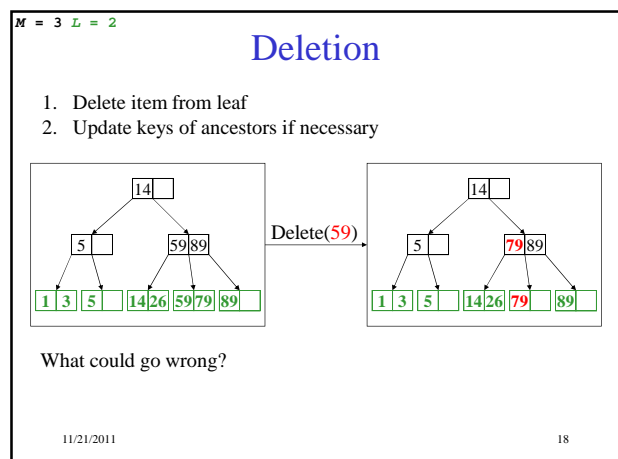
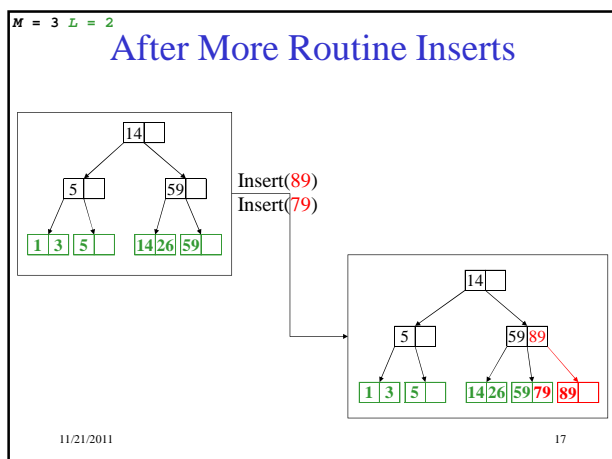
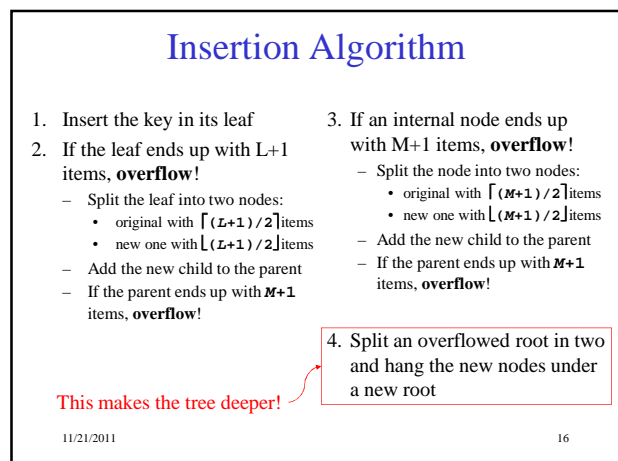
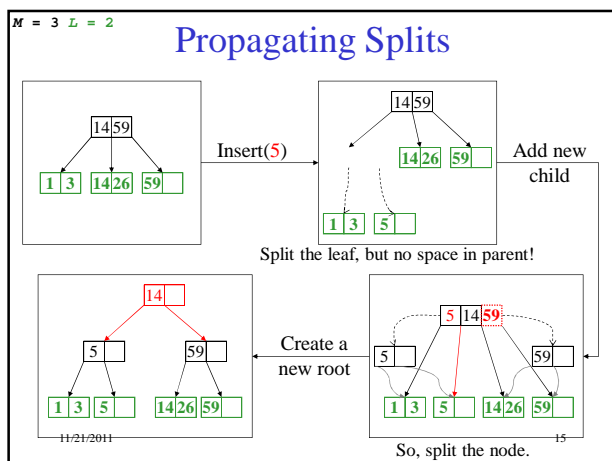
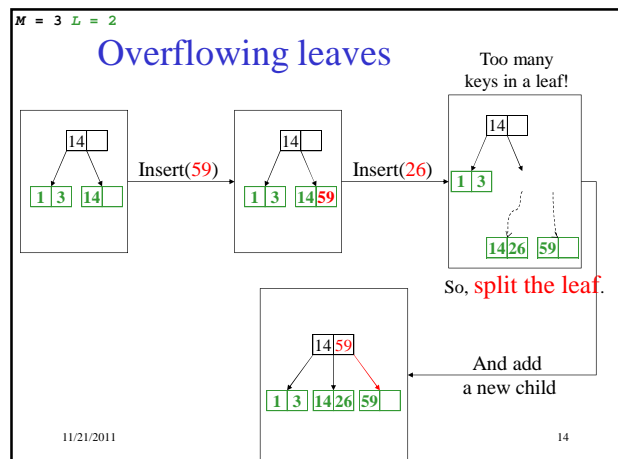
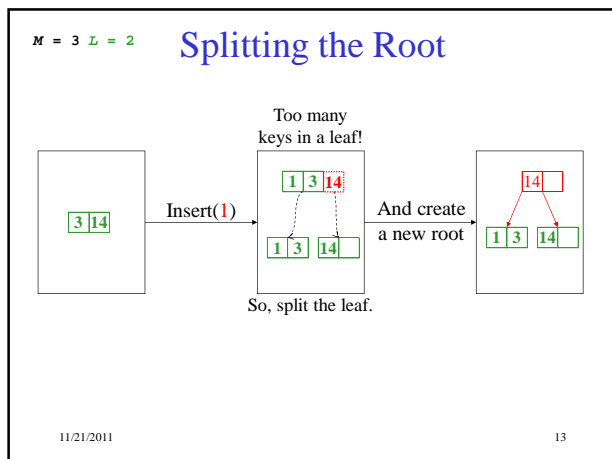
Building a B-Tree

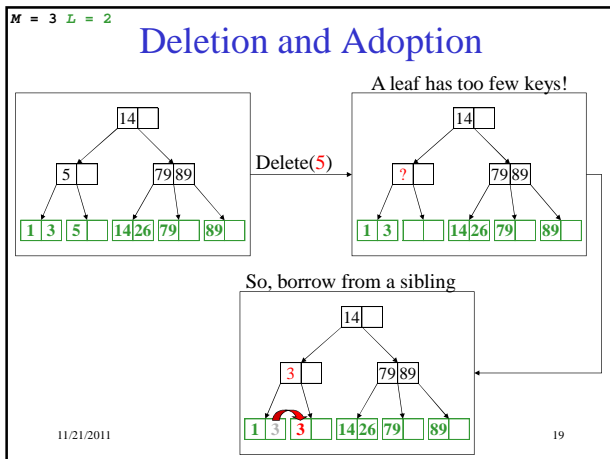


Now, Insert(1)?

11/21/2011

12



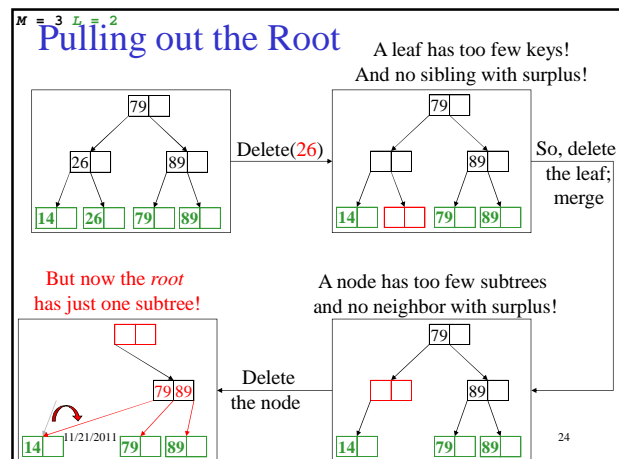
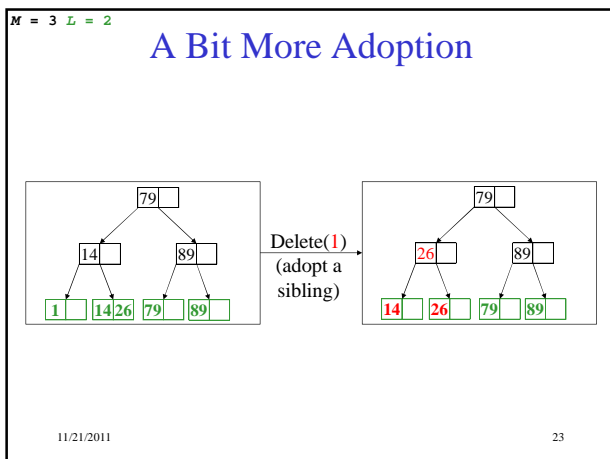
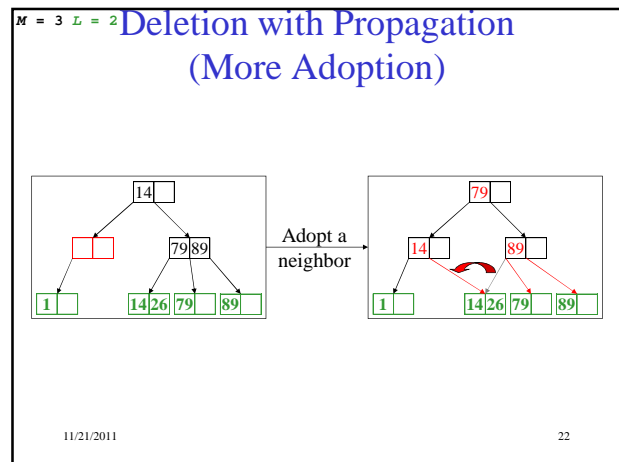
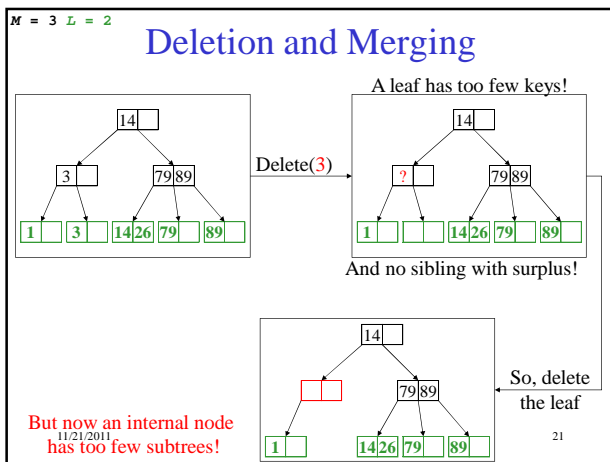


Does Adoption Always Work?

- What if the sibling doesn't have enough for you to borrow from?

e.g. you have $\lceil L/2 \rceil - 1$ and sibling has $\lceil L/2 \rceil$?

11/21/2011 20



$M = 3$ $L = 2$

Pulling out the Root (continued)

The *root* has just one subtree!

Simply make the one child the new root!

11/21/2011 25

Deletion Algorithm

1. Remove the key from its leaf
2. If the **leaf** ends up with fewer than $\lceil L/2 \rceil$ items, **underflow!**
 - Adopt data from a sibling; update the parent
 - If adopting won't work, delete node and merge with neighbor
 - If the parent ends up with fewer than $\lceil M/2 \rceil$ items, **underflow!**

11/21/2011 26

Deletion Slide Two

3. If an **internal** node ends up with fewer than $\lceil M/2 \rceil$ items, **underflow!**
 - Adopt from a neighbor; update the parent
 - If adoption won't work, merge with neighbor
 - If the parent ends up with fewer than $\lceil M/2 \rceil$ items, **underflow!**
4. If the root ends up with only one child, make the child the new root of the tree

This reduces the height of the tree!

11/21/2011 27

Thinking about B-Trees

- B-Tree **insertion** can cause (expensive) splitting and propagation
- B-Tree **deletion** can cause (cheap) adoption or (expensive) deletion, merging and propagation
- Propagation is rare if M and L are large (*Why?*)
- If $M = L = 128$, then a B-Tree of height 4 will store at least 30,000,000 items

11/21/2011 28

Tree Names You Might Encounter

FYI:

- B-Trees with $M = 3$, $L = x$ are called **2-3 trees**
 - Nodes can have 2 or 3 pointers
- B-Trees with $M = 4$, $L = x$ are called **2-3-4 trees**
 - Nodes can have 2, 3, or 4 pointers

11/21/2011 29

Determining M and L for a B-Tree

1 Page on disk = 1 KByte
 Key = 8 bytes, Pointer = 4 bytes
 Data = 256 bytes per record (includes key)

M =

L =

11/21/2011 30