B-Trees
(4.7 in Weiss)

CSE 373
Data Structures & Algorithms
Ruth Anderson

12/01/2010   1

Trees so far

- BST

- AVL

12/01/2010   2

CPU
(has registers)

TIme to access:

1 ns per instruction

SRAM
8KB - 4MB

Cache

Cache

2-10 ns

Main Memory

DRAM
up to 10GB

Main Memory

40-100 ns

Disk
many GB

Disk

a few
*milli*seconds
(5-10 *Million* ns)

12/01/2010   3

M-ary Search Tree

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

disk accesses for *find*:

Runtime of *find*:

12/01/2010   4

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 $x \le v < y$

  | 3 | 7 | 12 | 21 | |

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

  x<3   3≤x<7   7≤x<12   12≤x<21   21≤x

12/01/2010   5
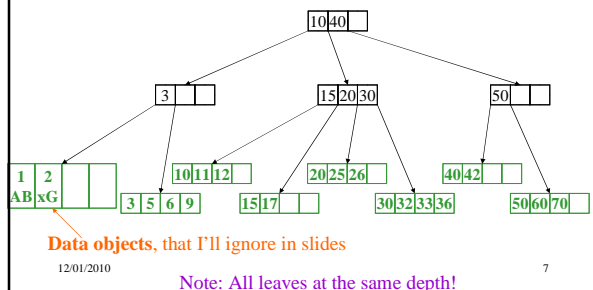

B-Trees

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

12/01/2010   6

1

B-Tree: Example

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

Data objects, that I'll ignore in slides

Note: All leaves at the same depth!

12/01/2010

7

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)

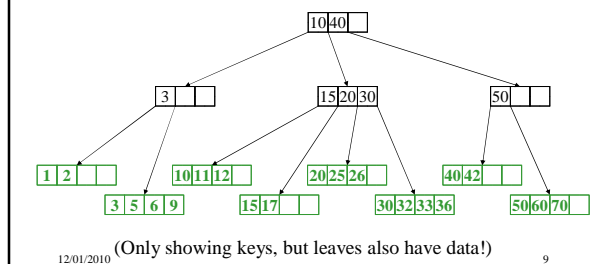12/01/2010   [‡]These are technically B[+]-Trees   8

Example, Again

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

(Only showing keys, but leaves also have data!)
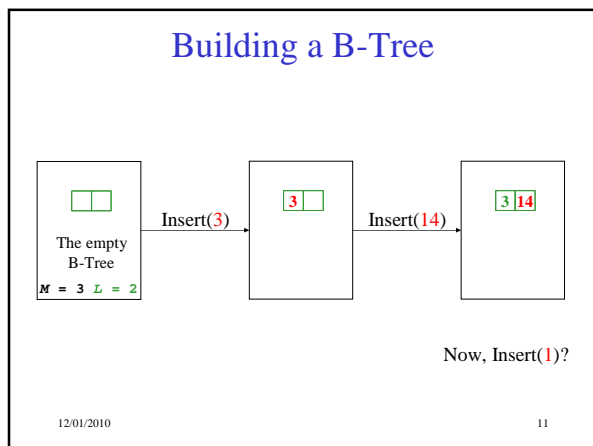
12/01/2010   9

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
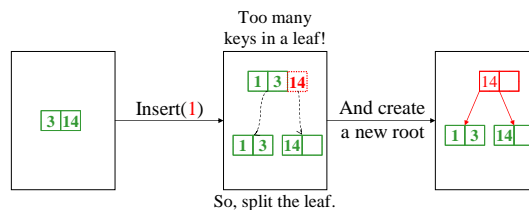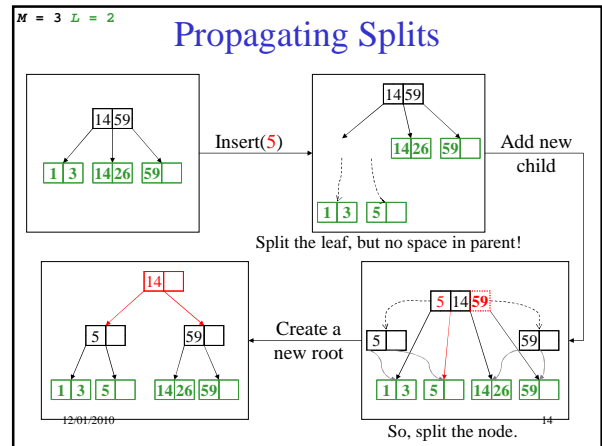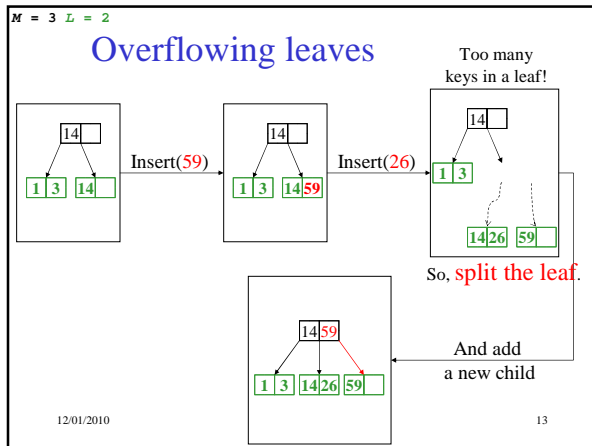
12/01/2010   10

Building a B-Tree

Now, Insert(1)?

12/01/2010   11

$M = 3\ L = 2$
Splitting the Root

12/01/2010   12

Overflowing leaves

Too many keys in a leaf!

Insert(59) Insert(26)

So, split the leaf.

And add a new child

12/01/2010 — 13

Propagating Splits

Insert(5) Add new child

Split the leaf, but no space in parent!

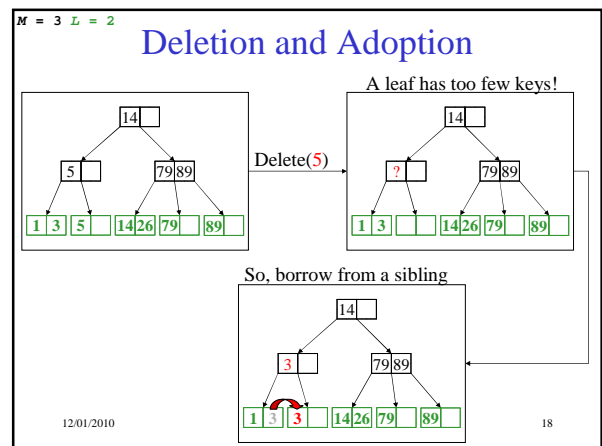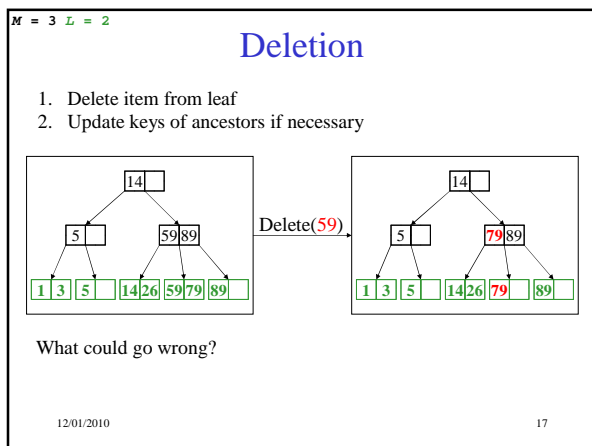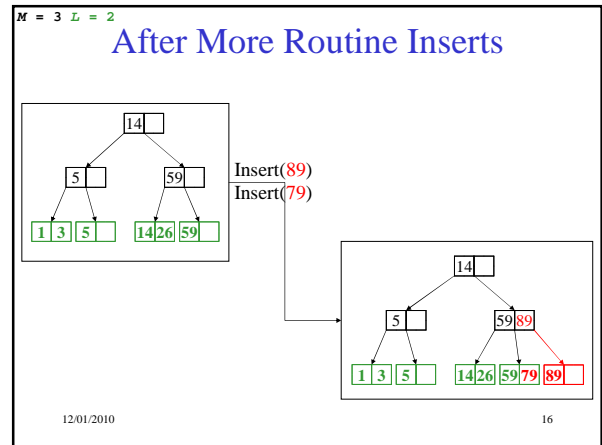Create a new root

So, split the node.

12/01/2010 — 14

Insertion Algorithm

1. Insert the key in its leaf
2. If the leaf ends up with L+1 items, **overflow!**
   - Split the leaf into two nodes:
     - original with $\lceil (L+1)/2 \rceil$ items
     - new one with $\lfloor (L+1)/2 \rfloor$ items
   - Add the new child to the parent
   - If the parent ends up with **M+1** items, **overflow!**

3. If an internal node ends up with M+1 items, **overflow!**
   - Split the node into two nodes:
     - original with $\lceil (M+1)/2 \rceil$ items
     - new one with $\lfloor (M+1)/2 \rfloor$ items
   - Add the new child to the parent
   - If the parent ends up with **M+1** items, **overflow!**

4. Split an overflowed root in two and hang the new nodes under a new root

This makes the tree deeper!

12/01/2010 — 15

After More Routine Inserts

Insert(89)
Insert(79)

12/01/2010 — 16

Deletion

1. Delete item from leaf
2. Update keys of ancestors if necessary

Delete(59)

What could go wrong?

12/01/2010 — 17

Deletion and Adoption

A leaf has too few keys!

Delete(5)

So, borrow from a sibling

12/01/2010 — 18

3

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 $\lceil L/2 \rceil$?

12/01/2010   19

$M = 3 \quad L = 2$

Deletion and Merging

A leaf has too few keys!

Delete(3)

And no sibling with surplus!

So, delete the leaf

But now an internal node has too few subtrees!

12/01/2010   20

$M = 3 \quad L = 2$

Deletion with Propagation (More Adoption)

Adopt a neighbor

12/01/2010   21

$M = 3 \quad L = 2$

A Bit More Adoption

Delete(1) (adopt a sibling)

12/01/2010   22

$M = 3 \quad L = 2$

Pulling out the Root

A leaf has too few keys! And no sibling with surplus!

Delete(26)

So, delete the leaf; merge

A node has too few subtrees and no neighbor with surplus!

Delete the node

But now the *root* has just one subtree!

12/01/2010   23

$M = 3 \quad L = 2$

Pulling out the Root (continued)

The *root* has just one subtree!

Simply make the one child the new root!

12/01/2010   24

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**!

12/01/2010   25

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!

12/01/2010   26

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

12/01/2010   27

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

12/01/2010   28

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 =

12/01/2010   29

5