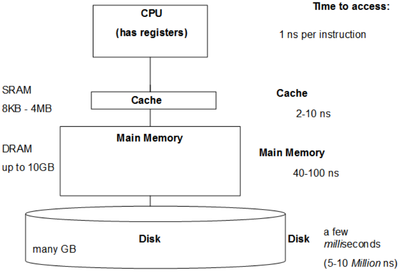
CSE 326 Data Structures

Dave Bacon

B-Trees

- Midterms graded, get back in section Thurs
- Project 2 due Wednesday at 11:59pm
- Homework 4 due Friday in class
  (Problem 2 Weiss 4.27 not 4.71)
- Reading: Finish Chapter 4
  → Chapt. 5.


CPU
(has registers)
Time to access:
1 ns per instruction
SRAM
8KB - 4MB
Cache
Cache
2-10 ns
DRAM
up to 10GB
Main Memory
Main Memory
40-100 ns
Disk
many GB
Disk
a few *milli*seconds
(5-10 *Million* ns)

Dictionary ADT

Trees so far

N = 10 Million

- BST

  Worst case $O(N)$

  10 Mil disk accesses

  $10^6 \cdot 10^{-3} s = 10^3 sec$

- AVL

  Worst case $O(\log_2 N)$

  $\log_2 10^7 \simeq 23$ → 3 disk accesses

  $23 \times 10^{-3} = 0.23$

- Splay

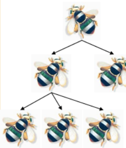
  amortized

  $O(\log_2 N)$

  2

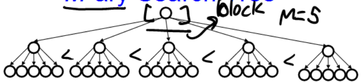
  4

B-Trees



Chapter 4 in Weiss

M-ary Search Tree



- Maximum branching factor of M
- Complete tree has height = $O(\log_M N)$

disk accesses for *find*: Best: $O(\log_M n)$

Worst: $O(n)$

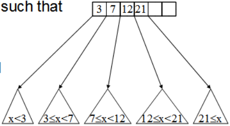
Runtime of *find*: $O(\log_M n \log_2 M) = O(\log_2 n)$

Solution: B-Trees

- specialized *M*-ary search trees

  ↖ M edges for each node

- 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

B-Trees

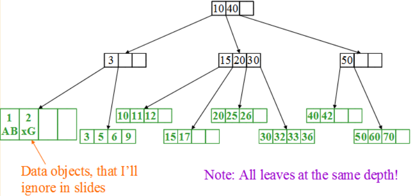
What makes them disk-friendly?

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

B-Tree: Example

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



Note: All leaves at the same depth!

B-Tree Properties ‡

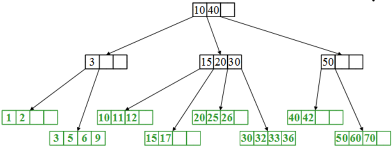
- Data is stored at the leaves
- All leaves are at the same depth and contains between $\lceil L/2 \rceil$ and L data items
- Internal nodes store up to M-1 keys
- Internal nodes have between $\lceil M/2 \rceil$ and M children
- Root (special case) has between 2 and M children (or root could be a leaf)

‡These are technically B^+-Trees

Example, Again

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

24 entries → 2 deep
BST → 4 deep



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

B-trees vs. AVL trees

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

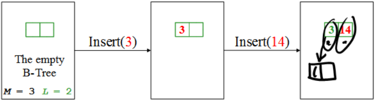
- Depth of AVL Tree

  $\log_2 10^8 = 26.6$

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

  $\log_{128} 10^8 = 3.8$

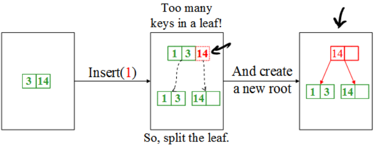
Building a B-Tree



Now, Insert(1)?

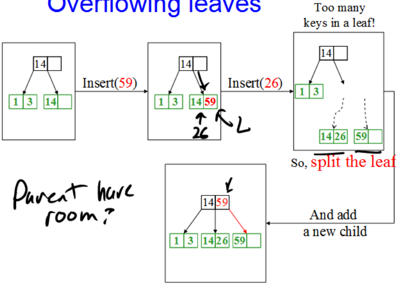
$M = 3$ $L = 2$

Splitting the Root



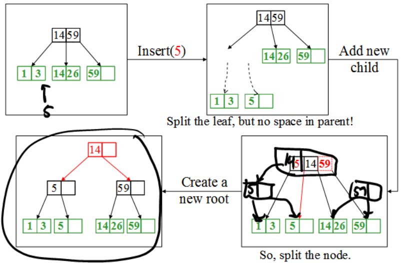
M = 3 L = 2

Overflowing leaves



M = 3 L = 2

Propagating Splits



Insertion Algorithm

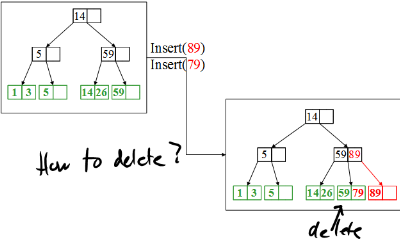
1. Insert the key in its leaf
2. If the leaf ends up with L+1 items, **overflow**!
   - Split the leaf into two nodes:
     - original with $\lceil (L+1)/2 \rceil$ items
     - new one with $\lfloor (L+1)/2 \rfloor$ items
   - Add the new child to the parent
   - If the parent ends up with $M+1$ items,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

This makes the tree deeper!

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

$M = 3$ $L = 2$

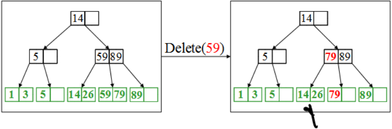
After More Routine Inserts



$M = 3$ $L = 2$

Deletion

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

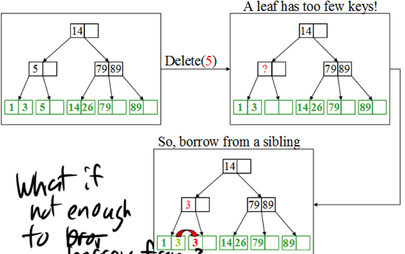


What could go wrong?

Might cause leaf < L/2 children

$M = 3$ $L = 2$

Deletion and Adoption



What if not enough to ~~bor~~ borrow from?

Does Adoption Always Work?

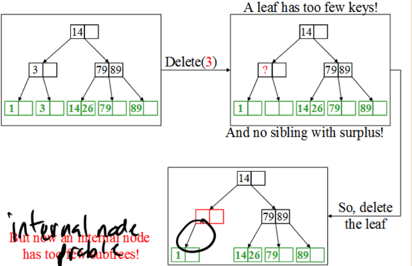
- What if the sibling doesn't have enough for you to borrow from?

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

Merge Together

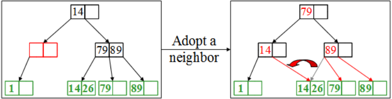
M = 3 L = 2

Deletion and Merging



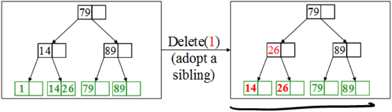
M = 3 L = 2

Deletion with Propagation (More Adoption)



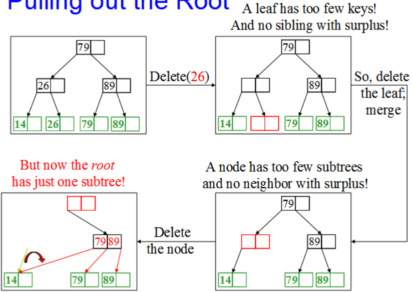
$M = 3$ $L = 2$

A Bit More Adoption



M = 3 L = 2

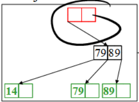
Pulling out the Root



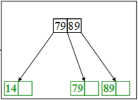
M = 3 *L* = 2

Pulling out the Root (continued)

The *root* has just one subtree!



Simply make the one child the new root!



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**!

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!

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

Tree Names You Might Encounter

FYI:

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