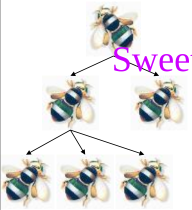
CPSC 221: Algorithms and Data Structures
Lecture #7
Sweet, Sweet Tree Hives



(B+-Trees, that is)

Steve Wolfman

2014W1

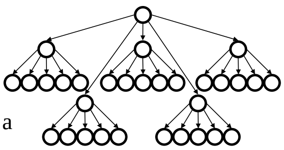
Today's Outline

- Addressing our other problem
- B+-tree properties
- Implementing B+-tree insertion and deletion
- Some final thoughts on B+-trees

M-ary Search Tree

- Maximum branching factor of ***M***
- Complete tree has height **h** ≅ $\log_M N$
- Each internal node in a complete tree has ***M* - 1** keys

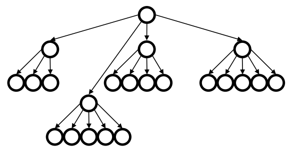
  runtime:



Actually, $h = \lceil \log_M(N(M-1)+1) \rceil - 1$

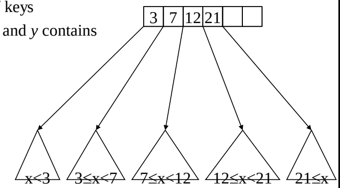
Incomplete *M*-ary Search Tree ☹

- Just like a binary tree, though, complete m-ary trees can store m^0 keys, $m^0 + m^1$ keys, $m^0 + m^1 + m^2$ keys, …
- What about numbers in between??



B+-Trees

- B+-Trees are specialized *M*-ary search trees
- Each node has many keys
  - at least some minimum # of keys
  - subtree between two keys *x* and *y* contains values *v* such that $x \leq v < y$
  - binary search within a node to find correct subtree
- Each node takes one full {*page, block, line*} of memory
- ALL the leaves are at the same depth!



Today's Outline

- Addressing our other problem
- B+-tree properties
- Implementing B+-tree insertion and deletion
- Some final thoughts on B+-trees

B+-Tree Properties

- Properties
  - maximum branching factor of M
  - the root has between 2 and M children *or* at most L keys/values
  - other internal nodes have between $\lceil M/2 \rceil$ and M children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys x and y equals x
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys/values
  - all leaves are at the same depth
- Result
  - tree is $\Theta(\log_M n)$ deep (between $\log_{M/2} n$ and $\log_M n$)
  - all operations run in $\Theta(\log_M n)$ time
  - operations get about $M/2$ to M or $L/2$ to L items at a time

7

B+-Tree Properties‡

- Properties
  - maximum branching factor of M
  - the root has between 2 and M children *or* at most L keys/values
  - other internal nodes have between $\lceil M/2 \rceil$ and M children
  - internal nodes contain only *search* keys (no data)
  - smallest datum between search keys x and y equals x
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys/values
  - all leaves are at the same depth
- Result
  - tree is $\Theta(\log_M n)$ deep (between $\log_{M/2} n$ and $\log_M n$)
  - all operations run in $\Theta(\log_M n)$ time
  - operations get about $M/2$ to M or $L/2$ to L items at a time

8

‡These are B+-Trees. B-Trees store data at internal nodes.

B+-Tree Properties

- Properties
  - maximum branching factor of M
  - the root has between 2 and M children *or* at most L keys/values
  - other internal nodes have between $\lceil M/2 \rceil$ and M children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys x and y equals x
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys/values
  - all leaves are at the same depth
- Result
  - tree is $\Theta(\log_M n)$ deep (between $\log_{M/2} n$ and $\log_M n$)
  - all operations run in $\Theta(\log_M n)$ time
  - operations get about $M/2$ to M or $L/2$ to L items at a time

9

B+-Tree Properties

- Properties
  - maximum branching factor of M
  - the root has between 2 and M children *or* at most L keys/values
  - other internal nodes have between $\lceil M/2 \rceil$ and M children
  - internal nodes contain only search keys (no data)
  - smallest datum between search keys x and y equals x
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and L keys/values
  - all leaves are at the same depth
- Result
  - height is $\Theta(\log_M n)$ between $\log_{M/2} (2n/L)$ and $\log_M (n/L)$
  - all operations run in $\Theta(\log_M n)$ time
  - operations get about $M/2$ to M or $L/2$ to L items at a time

10

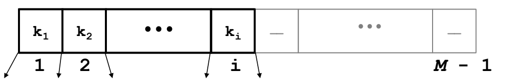
Today's Outline

- Addressing our other problem
- B+-tree properties
- Implementing B+-tree insertion and deletion
- Some final thoughts on B+-trees

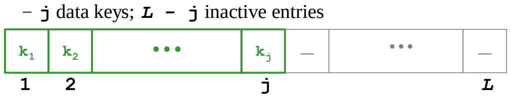
11

B+-Tree Nodes

- Internal node
  - i search keys; $i+1$ subtrees; $M - i - 1$ inactive entries



- Leaf
  - j data keys; $L - j$ inactive entries

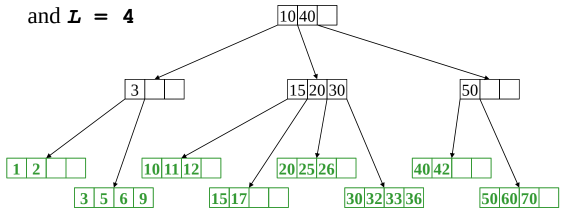


12

Example

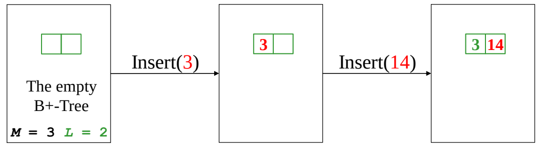
B+-Tree with $M = 4$

and $L = 4$



As with other dictionary data structures,
we show a version with no data, only keys, but ***only*** for simplicity!

Making a B+-Tree

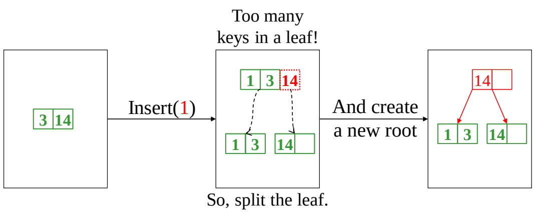


B-Tree with $M = 3$

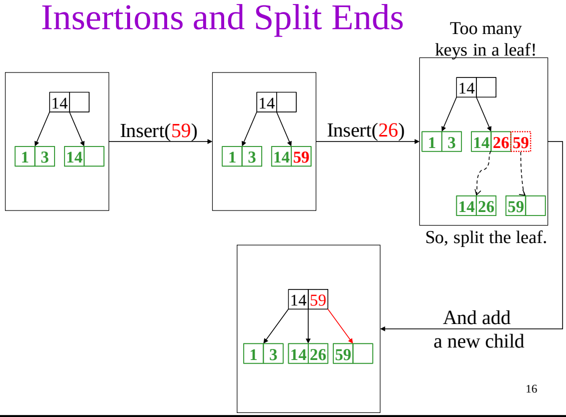
and $L = 2$

Now, Insert(1)?

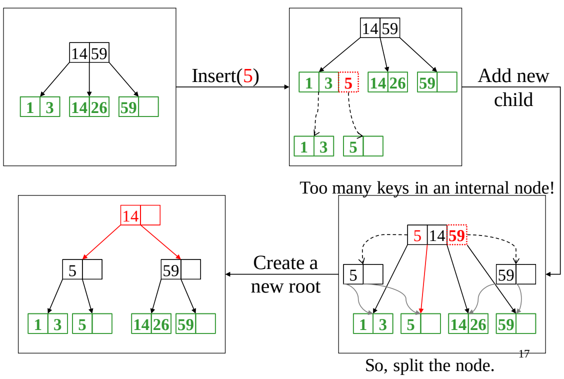
Splitting the Root



Insertions and Split Ends



Propagating Splits

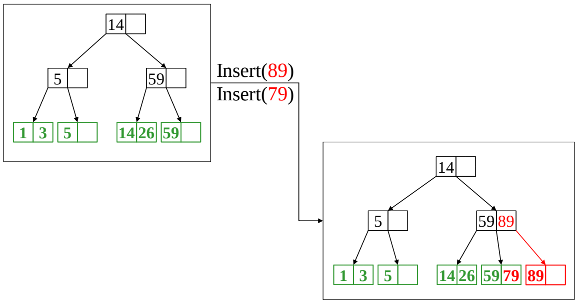


Insertion in Boring Text

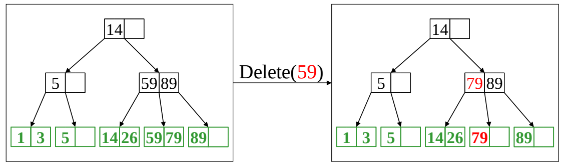
- Insert the key in its leaf
- If the leaf ends up with L+1 items, **overflow**!
  - Split the leaf into two nodes:
    - original with $\lceil (L+1)/2 \rceil$ items
    - new one with $\lfloor (L+1)/2 \rfloor$ items
  - Add the new child to the parent
  - If the parent ends up with $M+1$ items, **overflow**!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

This makes the tree deeper!

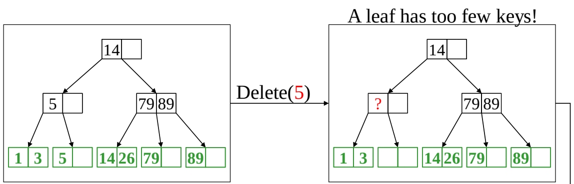
After More Routine Inserts



Deletion

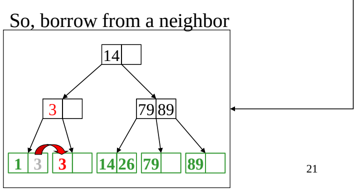


Deletion and Adoption

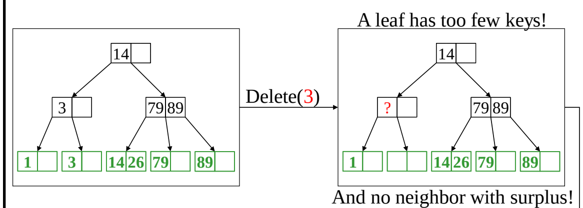


P.S. Parent + neighbour pointers. Expensive?

a. Definitely yes
b. Maybe yes
c. Not sure
d. Maybe no
e. Definitely no

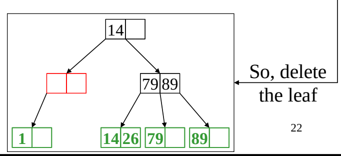


Deletion with Propagation

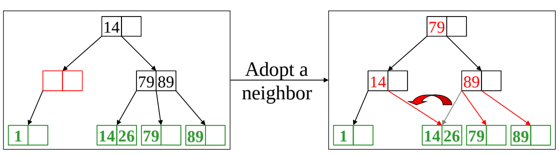


But now a node has too few subtrees!

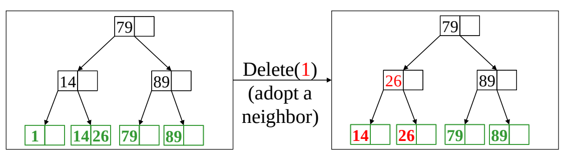
WARNING: with larger L, can drop below L/2 without being empty! (Ditto for M.)



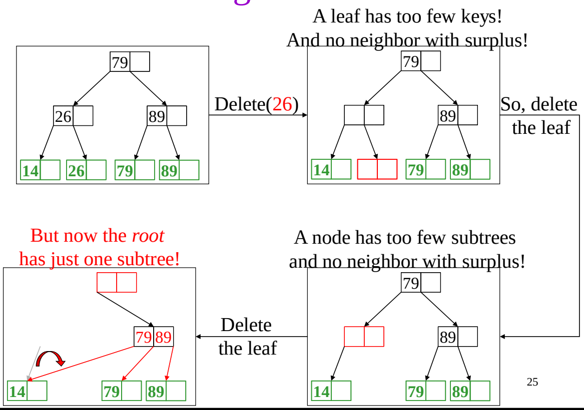
Finishing the Propagation (More Adoption)



A Bit More Adoption

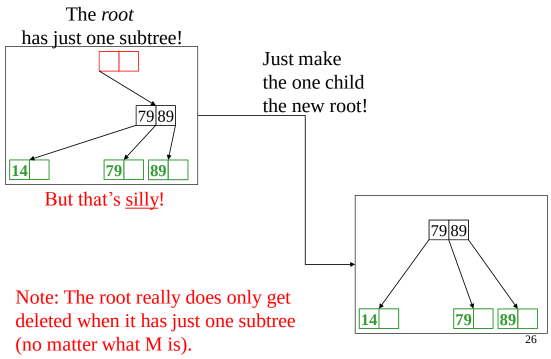


Pulling out the Root



25

Pulling out the Root (continued)



Note: The root really does only get deleted when it has just one subtree (no matter what M is).

26

Deletion in *Two* Boring Slides of Text

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

Will dumping keys always work if adoption does not?
a. Yes
b. It depends
c. No

27

Deletion Slide Two

- If a node ends up with fewer than $\lceil M/2 \rceil$ items, **underflow**!
  - Adopt subtrees from a neighbor; update the parent
  - If borrowing won't work, delete node and divide subtrees between neighbors
  - If the parent ends up with fewer than $\lceil M/2 \rceil$ items, **underflow**!
- 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!

28

Today's Outline

- Addressing our other problem
- B+-tree properties
- Implementing B+-tree insertion and deletion
- Some final thoughts on B+-trees

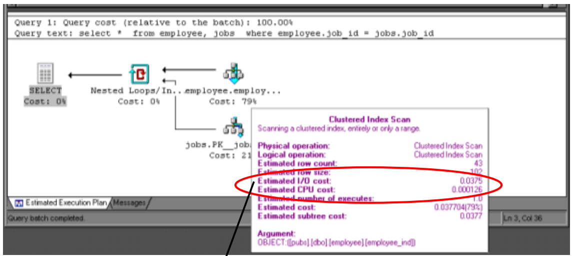
29

Thinking about B-Trees

- B+-Tree insertion can cause (expensive) splitting and propagation (could we do something like borrowing?)
- B+-Tree deletion can cause (cheap) borrowing or (expensive) deletion and propagation
- Propagation is rare if M and L are large *(Why?)*
- Repeated insertions and deletion can cause thrashing
- If $M = L = 128$, then a B-Tree of height 4 will store at least 30,000,000 items

30

Cost of a Database Query
(14 years ago… more skewed now!)



I/O to CPU ratio is 300!

A Tree with Any Other Name

FYI:

– B+-Trees with **M = 3**, **L = x** are called 2-3 trees
– B+-Trees with **M = 4**, **L = x** are called 2-3-4 trees
– Other balanced trees include Red-Black trees (rotation-based), Splay Trees (rotation-based and *amortized* O(lg n) bounds), B-trees, B*-trees, …

To Do

- Hashing readings

Coming Up

- In some order:
  – Everyone Gets a Crack at Parallelism
  – Hash Tables