Height-balanced trees

B trees

Tyler Moore

CS 2123, The University of Tulsa

Some slides created by or adapted from Dr. Kevin Wayne. For more information see

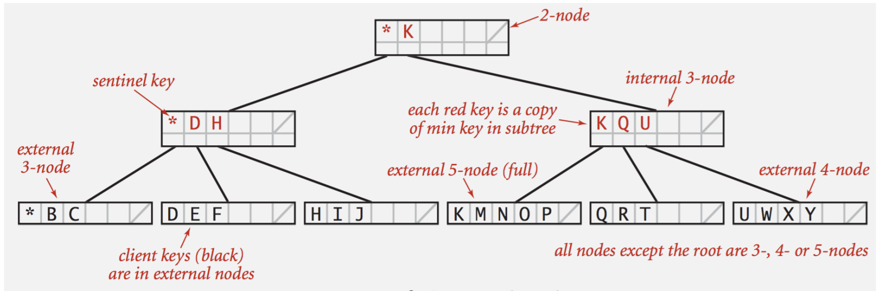
`https://www.cs.princeton.edu/courses/archive/fall12/cos226/lectures.php`

B-trees (Bayer-McCreight, 1972)

B tree of order m is a tree with the following properties

1. The root has at least two children unless it is a leaf
2. No node in the tree has more than m children
3. Every node except root and leaves has at least $\lceil \frac{m}{2} \rceil$ children
4. Internal node with k children contains exactly $k-1$ keys

Note: 2-3 trees are b-trees with $m = 3$



Anatomy of a B-tree set (M = 6)

Comparison with height-balanced trees

- In HB[k] trees, heights were allowed to vary by no more than k
- In HB[k] trees, only one key permitted per node
- In B-trees, we can have multiple keys per node
- In B-trees, we require multiple depth and vary the number of keys per node to enable cheap inserts and deletes
- In practice, we select m to be the biggest number that still fits in a page, e.g., $m = 1024$

File system model

Page. Contiguous block of data (e.g., a file or 4,096-byte chunk).
Probe. First access to a page (e.g., from disk to memory).

slow

fast

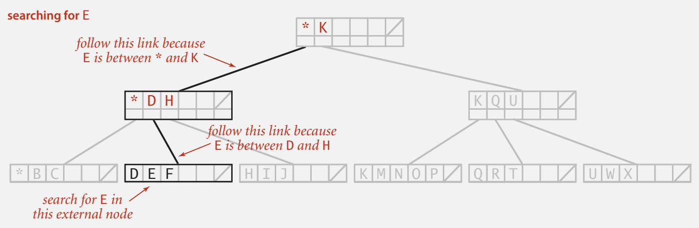
Property. Time required for a probe is much larger than time to access data within a page.

Cost model. Number of probes.

Goal. Access data using minimum number of probes.

Searching in a B-tree

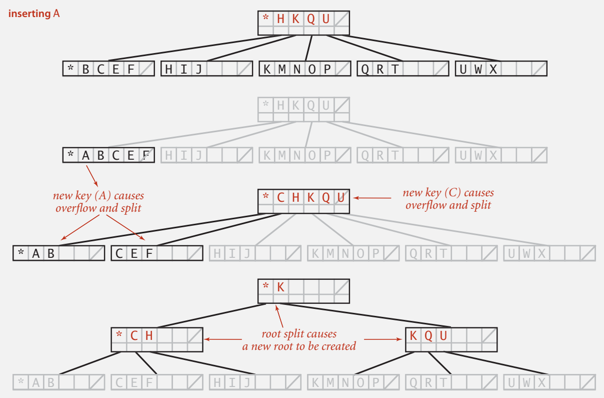
- Start at root.
- Find interval for search key and take corresponding link.
- Search terminates in external node.


searching for E
follow this link because E is between * and K
follow this link because E is between D and H
search for E in this external node
* K
* D H
K Q U
* B C
D E F
H I J
K M N O P
Q R T
U W X

Searching in a B-tree set (M = 6)

Insertion in a B-tree

- Search for new key.
- Insert at bottom.
- Split nodes with M key-link pairs on the way up the tree.


inserting A
* H K Q U
* B C E F
H I J
K M N O P
Q R T
U W X
* A B C E F
new key (A) causes overflow and split
* C H K Q U
new key (C) causes overflow and split
* A B
C E F
* K
* C H
K Q U
root split causes a new root to be created

Inserting a new key into a B-tree set

Rules for insertion in B-tree (courtesy RLW)

Insert a key into B tree:

1. Insert the key into the proper leaf node of the B Tree
2. If no overflow, insert complete. If overflow:
   - a Try to redistribute keys evenly with left sibling; if fails, then:
   - b Try to redistribute keys evenly with right sibling; if fails, then:
   - c Split overflow node into two nodes and promote 'middle' key to parent node. If parent node overflows, repeat step 2. If no parent, then create one (new root).

Rules for deletion from B-tree (courtesy RLW)

1. Locate the key to be deleted.
2. If the key to be deleted is in a leaf node; delete. Otherwise:
   - a Locate the next larger key (right child, then left to a leaf node).
   - b Exchange the next larger key with the key to be deleted. This places the key to be deleted in a leaf node. Now delete the key.
3. At this point a key has been removed from a leaf node. If there is no underflow, the delete is completed. If underflow then
   - a Try to redistribute keys evenly with left sibling; if fails, then:
   - b Try to redistribute keys evenly with right sibling; if fails, then:
   - c Combine two nodes into one node (the underflow node with its left sibling if it has one, otherwise combine the underflow node with the right sibling) and pull down the "splitter" key from the parent to be included in the combined node. If there is no underflow in the parent node, the delete is completed. If the underflow parent node is empty and is the root of the tree then remove the node, otherwise repeat starting at Step 3a.

B Tree Exercise

Sizing a b-tree

For order m b-tree:

- Maximum tree height for n keys: $k = \log_{\lceil \frac{m}{2} \rceil} n$

$\Longrightarrow$ Maximum height k is 4 for $m = 1024, n = 62$ billion

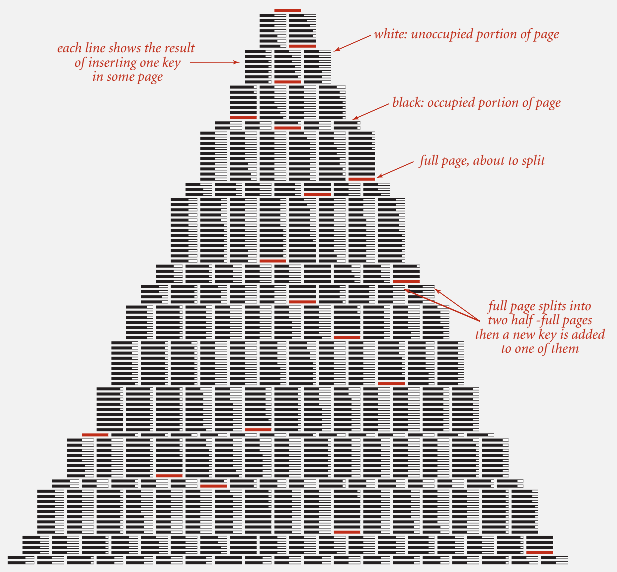
- $\#$ keys supported for height k: $n = \lceil \frac{m}{2} \rceil^k$

$\Longrightarrow$ For $m = 32, k = 3$: $n = \lceil \frac{32}{2} \rceil^3 = 4096$

Performance comparison of trees

Tree	Worst-case cost (after n inserts)			Avg.-case cost (after n inserts)		
	search	insert	delete	search	insert	delete
Unordered List	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Ordered Array	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$
BST	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$
AVL	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$
B-tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$

Building a large B tree


each line shows the result of inserting one key in some page
white: unoccupied portion of page
black: occupied portion of page
full page, about to split
full page splits into two half -full pages then a new key is added to one of them

B-trees in the real world

B-trees (and variants B* trees, B+ trees, etc.) are widely used for file systems and databases

- Windows: HPFS
- Mac: HFS, HFS+
- Linux: ReiserFS, XFS, Ext3FS, JFS
- Databases: ORACLE, DB2, INGRES, SQL, PostgreSQL