CSE 326: Data Structures
B+ Trees

James Fogarty

Spring 2009
Administrivia

• HW 4 Released Today, Due Next Friday
  – B+ Trees today, Hashing Monday

• Project 2B Due Wednesday
  – Project 2A Feedback Monday

• MidTerm Post-Mortem Wednesday
Traversing very large datasets

Suppose we had very many pieces of data (as we would in a database), such as \( n = 2^{30} \approx 10^9 \).

How many (worst case) hops through the tree to find a node?

- BST
  - List tree: \( 10^9 \)

- AVL
  - \( \log_{\phi} 10^9 = 1.44 \log_2 2^{30} = 43 \)

- Splay
  - List tree: \( 10^9 \)
Memory considerations

What is in a tree node? In an object?

Node:
   Object obj;
   Node left;
   Node right;

Object:
   Key key;
   …data…

Suppose the data is 1KB.

How much space does the tree take?\[10^9\text{KB} = 1\text{TB}\]

How much of the data can live in 1GB of RAM?\[10^6\text{ items, } 1/1000^{th}\text{ of data}\]
Cycles to access:

- Registers: 1
- L1 Cache: 2
- L2 Cache: 30
- Main memory: 250
- Disk:
  - Random: 30,000,000
  - Streamed: 5000
Minimizing random disk access

Almost all of our data structure is on disk.

Thus, hopping around in a tree amounts to random accesses to disk.

They are really, really painful.

How can we address this problem?

Big branching factor
Implemented using arrays of children at each node
Store keys in nodes, data at leaves
**$M$-ary Search Tree**

Suppose we devised a search tree with branching factor $M$:

Complete tree has height:

# hops for find:

Runtime of find:

- Balanced: $O(\log_M n)$
- Worst: $O(n)$

- Best: $O(\log_2 M \log_M n)$
- Worst: $O(n)$
B+ Trees
(book calls these B-trees)

• Each internal still has (up to) $M-1$ keys:

• Order property:
  – subtree between two keys $x$ and $y$
    contain leaves with values $v$
    such that $x \leq v < y$
  – Note the “≤”

• Leaf nodes contain up to $L$ sorted keys.
B+ Tree Structure Properties

Root (special case)
   – has between 2 and $M$ children (or root could be a leaf)

Internal nodes
   – store up to $M-1$ keys
   – have between $\left\lceil \frac{M}{2} \right\rceil$ and $M$ children

Leaf nodes
   – where data is stored
   – all at the same depth
   – contain between $\left\lceil \frac{L}{2} \right\rceil$ and $L$ data items

Nodes are at least $\frac{1}{2}$ full

Leaves are at least $\frac{1}{2}$ full
B+ Tree: Example

B+ Tree with $M = 4$ (# pointers in internal node) and $L = 5$ (# data items in leaf)

Data objects… which I’ll ignore in slides

Definition for later: “neighbor” is the next sibling to the left or right.
Disk Friendliness

What makes B+ trees disk-friendly?

1. Many keys stored in a node
   - All brought to memory/cache in one disk access.

2. Internal nodes contain only keys;
   Only leaf nodes contain keys and actual data
   - Much of tree structure can be loaded into memory irrespective of data object size
   - Data actually resides in disk
B+ trees vs. AVL trees

Suppose again we have \( n = 2^{30} \approx 10^9 \) items:

• Depth of AVL Tree

\[
\log_{128} 10^9 = 4.3
\]

• Depth of B+ Tree with \( M = 256, \quad L = 256 \)

\[
\log_{128} 10^9 = 4.3
\]

So let’s see how we do this…
Building a B+ Tree with Insertions

The empty B-Tree

$M = 3 \quad L = 3$
$M = 3 \quad L = 3$
\[ M = 3 \quad L = 3 \]
Insert(16)
Insert(12,40,45,38)

\[ M = 3 \quad L = 3 \]
1. Insert the key in its leaf in sorted order
2. If the leaf ends up with \( L+1 \) items, **overflow**!
   - Split the leaf into two nodes:
     • original with \( \lceil \frac{L+1}{2} \rceil \) smaller keys
     • new one with \( \lfloor \frac{L+1}{2} \rfloor \) larger keys
   - Add the new child to the parent
   - If the parent ends up with \( M+1 \) children, **overflow**!
3. If an internal node ends up with \( M+1 \) children, **overflow**!
   - Split the node into two nodes:
     • original with \( \lceil \frac{M+1}{2} \rceil \) children with smaller keys
     • new one with \( \lfloor \frac{M+1}{2} \rfloor \) children with larger keys
   - 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
5. Propagate keys up tree.

This makes the tree deeper!

Note that new leaves/ internal nodes guaranteed to have enough nodes after split. Why?
And Now for Deletion…

\[ M = 3 \quad L = 3 \]

Let them eat cake!
Are we okay?

Are you using that 14? Can I borrow it?

Dang, not half full

\( M = 3 \quad L = 3 \)
\[ M = 3 \quad L = 3 \]
Are you using that 12? Are you using that 18?

\[ M = 3 \quad L = 3 \]
Are you using that 18/30?

\[ M = 3 \quad L = 3 \]
$M = 3 \quad L = 3$
$M = 3 \quad L = 3$

Yay I like cake!
$M = 3 \quad L = 3$
$M = 3 \quad L = 3$
\[ M = 3 \quad L = 3 \]
Deletion Algorithm

1. Remove the key from its leaf

2. If the leaf ends up with fewer than \( \lfloor \frac{L}{2} \rfloor \) items, underflow!
   - Adopt data from a neighbor; update the parent
   - If adopting won’t work, delete node and merge with neighbor
   - If the parent ends up with fewer than \( \lfloor \frac{M}{2} \rfloor \) children, underflow!
3. If an internal node ends up with fewer than \(
\left\lfloor \frac{M}{2} \right\rfloor
\) children, **underflow**!
   - Adopt from a neighbor; update the parent
   - If adoption won’t work, merge with neighbor
   - If the parent ends up with fewer than \( \left\lfloor \frac{M}{2} \right\rfloor \) children, **underflow**!

4. If the root ends up with only one child, this reduces the height of the tree!
Thinking about B+ Trees

• B+ Tree insertion can cause (expensive) splitting and propagation up the tree
• B+ Tree deletion can cause (cheap) adoption or (expensive) merging and propagation up the tree
• Split/merge/propagation is rare if \( M \) and \( L \) are large (Why?)
• Pick branching factor \( M \) and data items/leaf \( L \) such that each node takes one full page/block of memory/disk.

Only 1/L inserts cause split, only 1/M of these go up!
Complexity

• Find: $O(M)$ costs are negligible
• Insert: $O(\log_2 M \log_M n)$
  – find: $O(\log_2 M \log_M n)$
  – Insert in leaf: $O(M)$
  – split/propagate up: $O(M \log_M n)$

• Claim: $O(M)$ costs are negligible, it’s disk that kills us