Beyond Binary Trees

➢ One of the most important applications for search trees is databases
➢ If the DB is small enough to fit into RAM, almost any scheme for balanced trees is okay

<table>
<thead>
<tr>
<th>Year</th>
<th>RAM (MB)</th>
<th>DB (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>1MB</td>
<td>100 MB</td>
</tr>
<tr>
<td>2002</td>
<td>1,000 MB</td>
<td>1,000,000 MB (terabyte)</td>
</tr>
</tbody>
</table>

*gap between disk and main memory growing!

Time Gap

➢ For many corporate and scientific databases, the search tree must mostly be on disk
➢ Accessing disk 200,000 times slower than RAM
➢ Visiting node = accessing disk
➢ Even perfectly balance binary trees a disaster!

\[ \log(10,000,000) = 24 \text{ disk accesses} \]

*Goal: Decrease Height of Tree*

M-ary Search Tree

➢ Maximum branching factor of \( M \)
➢ Complete tree has depth \( \log_M n \)
➢ Each internal node in a complete tree has \( M - 1 \) keys

\[ \text{runtime:} \]

B-Trees

➢ B-Trees are balanced \( M \)-ary search trees
➢ Each node has many keys
  • internal nodes - between \( \frac{M}{2} \) and \( M \) children (except root), no data – only keys
  • smallest datum between search keys \( x \) and \( y \) equals \( x \)
  • binary search within a node to find correct subtree
  • each leaf contains between \( \frac{L}{2} \) and \( L \) keys
  • all leaves are at the same depth
  • choose \( M \) and \( L \) so that each node takes one full (page, block, line) of memory (why?)

Result:

\[ \text{tree is } \log_{\frac{M}{2}} n(L/2) +/1 \text{ deep} \]

When Big-O is Not Enough

\[ \log_{\frac{M}{2}} n(L/2) \]
\[ = \log_{\frac{M}{2}} n - \log_{\frac{M}{2}} L/2 \]
\[ = O(\log_{\frac{M}{2}} n) \text{ steps per option} \]
\[ = O(\log n) \text{ steps per operation} \]

*Where’s the beef?!*

\[ \left\lceil \log(10,000,000) \right\rceil = 24 \text{ disk accesses} \]
\[ \left\lceil \log_{\text{Enode}}(10,000,000/(200/2)) \right\rceil = \left\lceil \log_{100}(100,000) \right\rceil = 3 \text{ accesses} \]
Making a B-Tree

The empty B-Tree

Insert(3)

Insert(14)

Insert(3)

Now, Insert(1)?

Splitting the Root

Too many keys in a leaf!

Insert(1)

And create a new root

So, split the leaf.

Insertions and Split Ends

Too many keys in a leaf!

Insert(59)

Insert(26)

So, split the leaf.

And add a new child

Propagating Splits

Too many keys in an internal node!

Insert(5)

Add new child

Create a new root

So, split the node.

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 $\lfloor (L+1)/2 \rfloor$ items
    - new one with $\lceil (L+1)/2 \rceil$ items
    - Add the new child to the parent
    - If the parent ends up with $M+1$ items, overflow!

This makes the tree deeper!

Deletion

Delete(59)
Deletion and Adoption

A leaf has too few keys!

Delete(5)

So, borrow from a neighbor

Deletion with Propagation

A leaf has too few keys!

Delete(3)

And no neighbor with surplus!

But now a node has too few subtrees!

So, delete the leaf

Finishing the Propagation

(More Adoption)

Adopt a neighbor

Deletion in Two

Boring Slides of Text

Run Time Analysis of B-Tree Operations

- If an internal node ends up with fewer than \([M/2]\) 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 \([M/2]\) 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!

For a B-Tree of order M:
- Depth is \( \log_{M/2} \sqrt{M/(L/2)} + \epsilon \)
- Find: run time in terms of both \( n \) and \( M=L \) is:
  - \( O(\log M) \) for binary search of each internal node
  - \( O(\log L) = O(\log M) \) for binary search of the leaf node
  - Total is \( \leq O((\log \sqrt{M/(M/2)})/(\log M) + \log M) \)
    \( = O(\log n/(M/2)) + O(\log M) \)
    \( = O(\log n + \log M) \)
Run Time Analysis of B-Tree Operations

- Insert and Delete: run time in terms of both $n$ and $M=L$ is:
  - $O(M)$ for search and split/combine of internal nodes
  - $O(L)$ = $O(M)$ for search and split/combine of leaf nodes
  - Total is $\leq O((\log_{\log_{n}} n/(M/2))(M+M))$
    = $O(M/(\log M) \log n)$

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

Why would we ever use these?

Summary

- BST: fast finds, inserts, and deletes $O(\log n)$ on average (if data is random!)
- AVL trees: guaranteed $O(\log n)$ operations
- B-Trees: also guaranteed $O(\log n)$, but shallower depth makes them better for disk-based databases

- What would be even better?
  - How about: $O(1)$ finds and inserts?

Coming Up

- Hash Tables
- Another assignment ?!
- Midterm
- Another project?!