Get out some paper:
Characteristics of some Dictionary ADT Implementations

- (unbalanced) Binary Search Trees
- AVL Trees
- Splay Trees
- Other?

Something We Forgot: Disk Acesses

We Want To Minimize Disk Acesses!

1024 bytes

- Entire blocks transferred into memory at a time
- Transfer time much less than search time
- Therefore we need to minimize disk accesses!

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
run time:
B-Trees

- B-Trees are specialized $M$-ary search trees
- Each node has many keys
  - subtree between two keys $x$ and $y$ contains leaves with 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

B-Tree Properties

- Properties
  - maximum branching factor of $M$
  - the root has between 2 and $M$ children or at most $L$ keys
  - other internal nodes have between $\lceil M/2 \rceil$ and $M$ children
  - internal nodes contain only search keys (no values)
  - smallest datum between search keys $x$ and $y$ equals $x$
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and $L$ keys
  - all leaves are at the same depth

B-Tree Properties

- Properties
  - maximum branching factor of $M$
  - the root has between 2 and $M$ children or at most $L$ keys
  - other internal nodes have between $\lceil M/2 \rceil$ and $M$ children
  - internal nodes contain only search keys (no values)
  - smallest datum between search keys $x$ and $y$ equals $x$
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and $L$ keys
  - all leaves are at the same depth

B-Tree Properties

- Properties
  - maximum branching factor of $M$
  - the root has between 2 and $M$ children or at most $L$ keys
  - other internal nodes have between $\lceil M/2 \rceil$ and $M$ children
  - internal nodes contain only search keys (no values)
  - smallest datum between search keys $x$ and $y$ equals $x$
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and $L$ keys
  - all leaves are at the same depth

B-Tree Properties

- Properties
  - maximum branching factor of $M$
  - the root has between 2 and $M$ children or at most $L$ keys
  - other internal nodes have between $\lceil M/2 \rceil$ and $M$ children
  - internal nodes contain only search keys (no values)
  - smallest datum between search keys $x$ and $y$ equals $x$
  - each (non-root) leaf contains between $\lceil L/2 \rceil$ and $L$ keys
  - all leaves are at the same depth

B-Tree Nodes

- Internal node
  - $i$ search keys; $i+1$ subtrees; $M - 1 - i$ inactive entries
- Leaf
  - $j$ values; $L - j$ inactive entries
  - all operations run in $\Theta(\log_m n)$ time
  - operations pull in about $M/2$ or $L/2$ items at a time

These are technically B+ Trees
Example

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

Making a B-Tree

Insert(3) → Insert(14) → Insert(1)

Now, Insert(1)?

Splitting the Root

Insert(3) → Insert(14) → Insert(1)

Insertions and Split Ends

Insert(3) → Insert(14) → Insert(1)

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

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

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

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

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

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

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

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

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

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

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

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 $(L+1)/2$ items
    - new one with $(L+1)/2$ 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 $(M+1)/2$ items
    - new one with $(M+1)/2$ items
  - Add the new child to the parent
  - If the parent ends up with $M+1$ items, overflow!
After More Routine Inserts

Deletion

Deletion and Adoption

Deletion with Propagation

Finishing the Propagation (More Adoption)

A Bit More Adoption
Pulling out the Root

A leaf has too few keys!
And no neighbor with surplus!

Delete(26)

But now the root
has just one subtree!

Pulling out the Root (continued)

The root
has just one subtree!

Just make the one child
the new root!

Deletion in Two
Boring Slides of Text

• Remove the key from its leaf
• If the leaf ends up with fewer
  than \( \lceil N/2 \rceil \) 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 \( \lceil N/2 \rceil \) items, underflow!

Deletion Slide Two

• If a node ends up with fewer
  than \( \lceil N/2 \rceil \) 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 \( \lceil N/2 \rceil \) 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!

B-trees vs AVL trees

We have a database* with 100 million items (100,000,000):

• Depth of AVL Tree
• Depth of B+ Tree with \( B = 128 \), \( L = 64 \)

Thinking about B-Trees

• B-Tree insertion can cause (expensive) splitting and propagation
• 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

* A very simple type of database, called "Berkeley Database" is basically a B-tree
A Tree with Any Other Name

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

**Why would we ever use these?**

Another Way

- B-Trees are *hairy* in practice

*hairy*: **Slang**. Fraught with difficulties; hazardous:
  a hairy escape; hairy problems.