

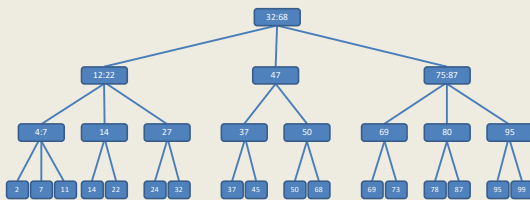
CSE 332: Data Structures and Parallelism

Fall 2022
Richard Anderson
Lecture 10: B-Trees

Announcements

- Exercises 7 (Oct 24) and 8 (Oct 28)
 - AVL Trees (useful for P2, especially Ex 08)
- Exercises 5 and 6 dropped

2-3 Trees



Internal nodes of degree 2 or 3
All leaves at the same depth
Values at leaves

Height is $O(\log n)$
Deletes and inserts may violate 2-3 invariant
Fix by child stealing, parent splitting, parent merging

Today

- Generalize search trees to external storage
- Access to external storage is slow
 - But you can get a large amount of data
 - Data only available in large chunks
- Motivates designing trees with very large branching factor
 - Shallow trees
- B Trees
 - Extend 2-3 trees to high degree, e.g., 128-256 trees

Thinking about computation

- Algorithmic view
 - Computation is a sequence of primitive operations
 - Abstract machine
 - Various approaches
 - Runtime as a function of input size
 - Asymptotic view
 - This approach has been very successful
 - Basic understanding for implementation of algorithms
 - Foundation for mathematical theory of computation

Where does this model break?

- Model: sequence of operations of roughly equal cost
- Model breaks if it does not suggest appropriate implementation techniques
- When is “roughly equal cost” wrong?

Computer Architecture

- CPU – collection of highly engineered computational gadgets
- Dominant consideration – keeping the CPU fed with data to keep all operations running
- Memory access costs
 - The closer data is to the CPU the faster it is to access
 - Different technologies in hierarchy change costs

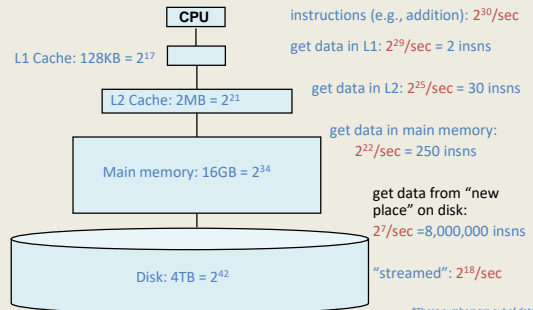
10/19/2022

CSE 332

7

A typical hierarchy

*Every desktop/laptop/server is different but here is a plausible configuration these days**



10/19/2022

CSE 332

*These numbers are out of date 8

It is much faster to do:	Than:
5 million arithmetic ops	1 disk access
2500 L2 cache accesses	1 disk access
400 main memory accesses	1 disk access

Why are computers built this way?

- Physical realities (speed of light, closeness to CPU)
- Cost (price per byte of different technologies)
- Disks get much bigger not much faster
- Speedup at higher levels makes lower levels *relatively slower*

10/19/2022

CSE 332

9

Usually, it doesn't matter . . .

The hardware automatically moves data into the caches from main memory for you

- Replacing items already there
- So algorithms much faster if "data fits in cache" (often does)

Disk accesses are done by software (e.g., ask operating system to open a file or database to access some data)

So most code "just runs" but sometimes it's worth designing algorithms / data structures with knowledge of memory hierarchy

- And when you do, you often need to know one more thing...

10/19/2022

CSE 332

10

Model of data access

- Two separate issues
 - What is the latency
 - How much data is delivered at a time
- Buying in bulk
- Natural size of data delivery (page)
- External storage boundary most important to consider

10/19/2022

CSE 332

11

BSTs?

- Looking things up in balanced binary search trees is $O(\log n)$, so even for $n = 2^{39}$ (512GB) we need not worry about minutes or hours
- Still, number of disk accesses matters
 - AVL tree could have height of 55
 - So each **find** could take about 0.5 seconds or about 100 finds a minute
 - Most of the nodes will be on disk: the tree is shallow, but it is still many gigabytes big so the *tree* cannot fit in memory
 - Even if memory holds the first 25 nodes on our path, we still need 30 disk accesses

10/19/2022

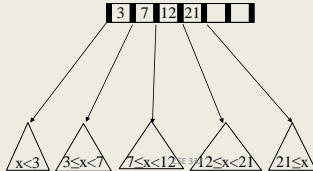
CSE 332

12

B Trees

(Also called B+ trees)

- Each internal node 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 " \leq "
- Leaf nodes have up to L sorted keys.



10/19/2022

13

B Tree Structure Properties

Internal nodes

- store up to $M-1$ keys
- have between $\lceil M/2 \rceil$ and M children

Leaf nodes

- where data is stored
- all at the same depth
- contain between $\lceil L/2 \rceil$ and L data items

Root (special case)

- has between 2 and M children (or root could be a leaf)

10/19/2022

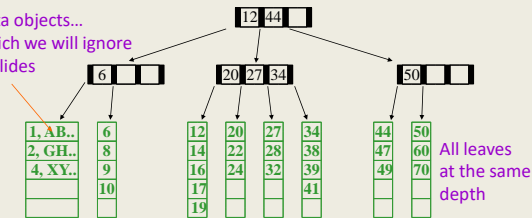
CSE 332

14

B Tree: Example

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

Data objects...
which we will ignore
in slides



All leaves
at the same
depth

Definition for later: "neighbor" is the next sibling to the left or right.

10/19/2022

CSE 332

15

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

10/19/2022

CSE 332

16

B trees vs. AVL trees

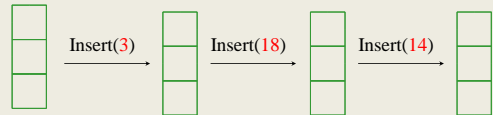
- Suppose again we have $n = 2^{30} \approx 10^9$ items:
- Depth of AVL Tree
- Depth of B+ Tree with $M = 256$, $L = 256$
- Great, but how do we actually make a B tree and keep it balanced...?

10/19/2022

CSE 332

17

Building a B Tree with Insertions



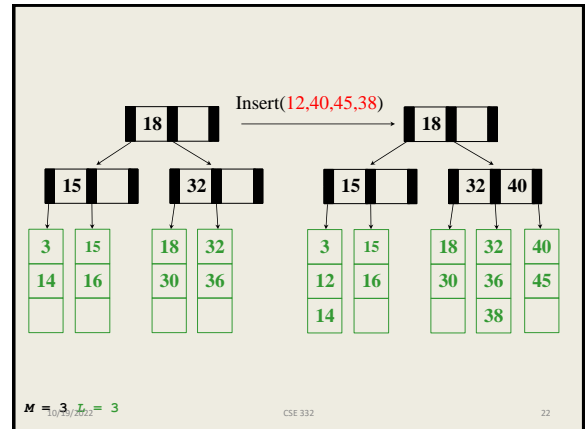
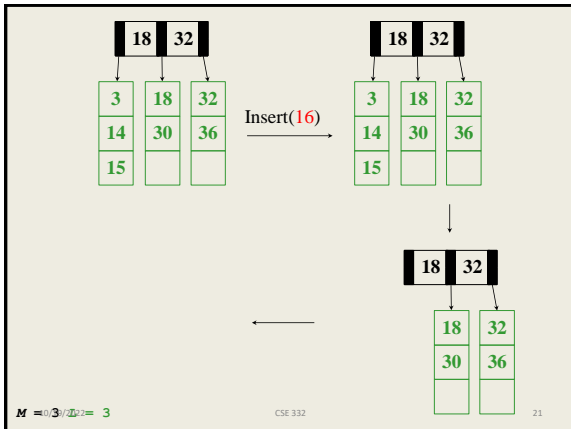
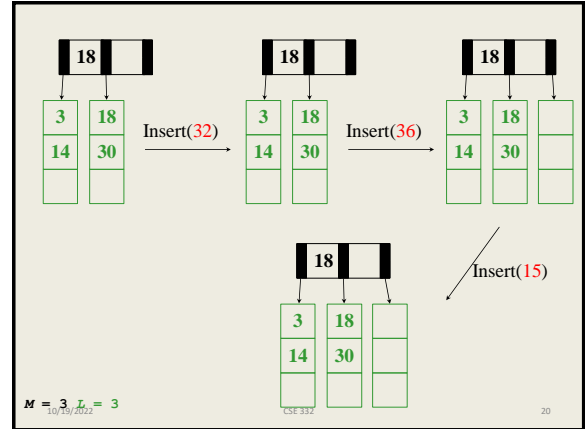
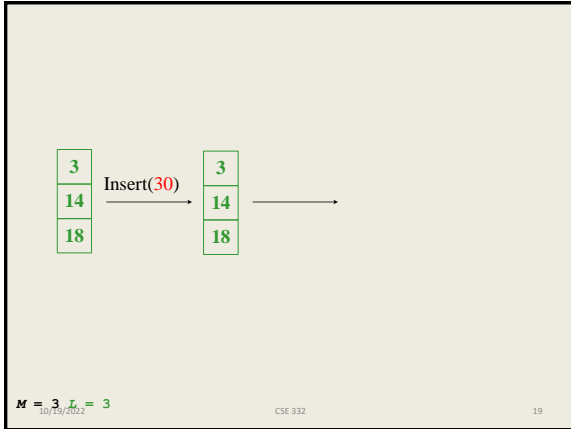
The empty
B-Tree

$M = 3$ $L = 3$

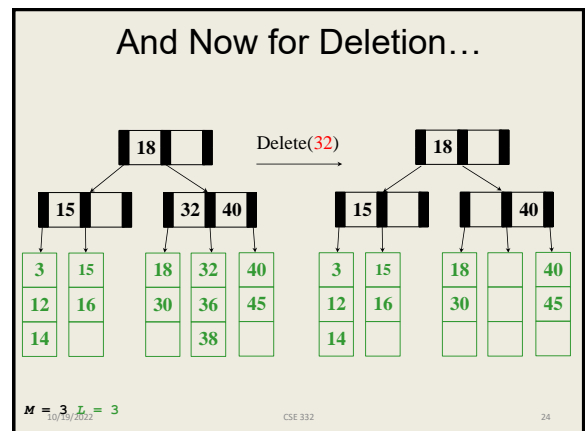
10/19/2022

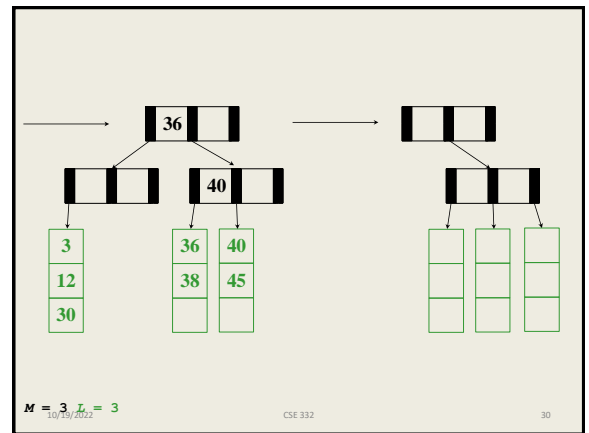
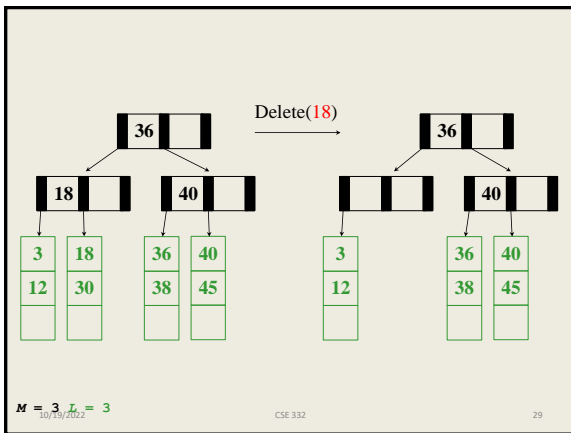
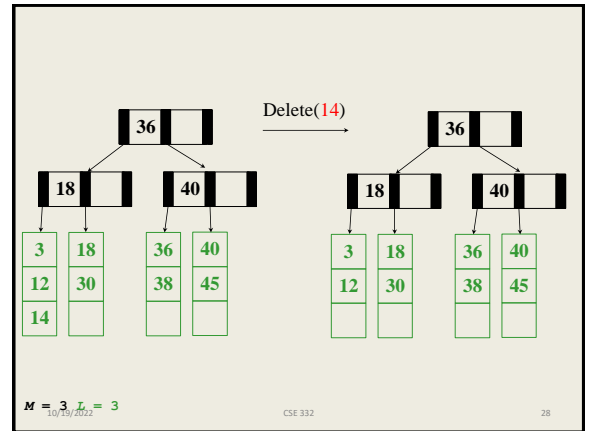
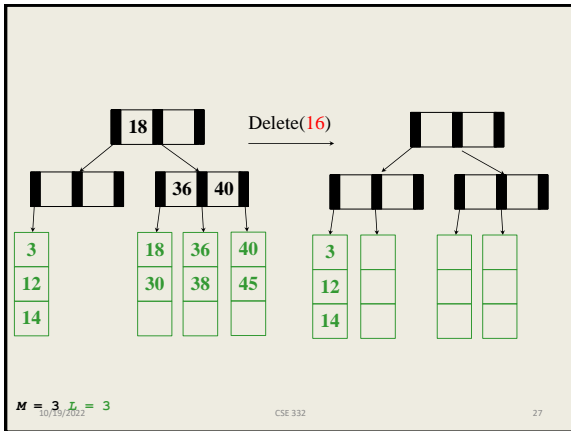
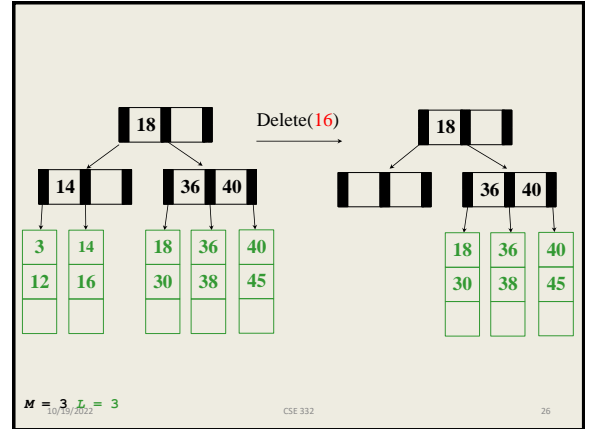
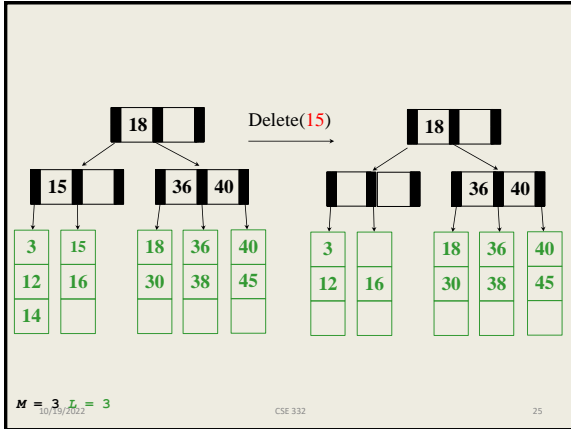
CSE 332

18



- ### Insertion Algorithm
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 $\lfloor (L+1)/2 \rfloor$ smaller keys
 - new one with $\lfloor (L+1)/2 \rfloor$ 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 $\lfloor (M+1)/2 \rfloor$ children with smaller keys
 - new one with $\lfloor (M+1)/2 \rfloor$ 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!
- 10/19/2022
- CSE 332
- 23





Deletion Algorithm

1. Remove the key from its leaf
- 2. If the leaf ends up with fewer than $\lfloor L/2 \rfloor$ 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 $\lfloor M/2 \rfloor$ children, **underflow!**

10/19/2022

CSE 332

31

Deletion Slide Two

- 3. If an internal node ends up with fewer than $\lfloor M/2 \rfloor$ 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 $\lfloor M/2 \rfloor$ children, **underflow!**
- 4. If the root ends up with only one child, make the child the new root of the tree

5. Propagate keys up through tree.

This reduces the height of the tree!

10/19/2022

CSE 332

32

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.
- Lots of engineering of B-Trees in a database system, including many performance hacks.

10/19/2022

CSE 332

33

Complexity

- Find:
- Insert:
 - find:
 - Insert in leaf:
 - split/propagate up:
- Claim: $O(M)$ costs are negligible

10/19/2022

CSE 332

34