Adv. Data Structures

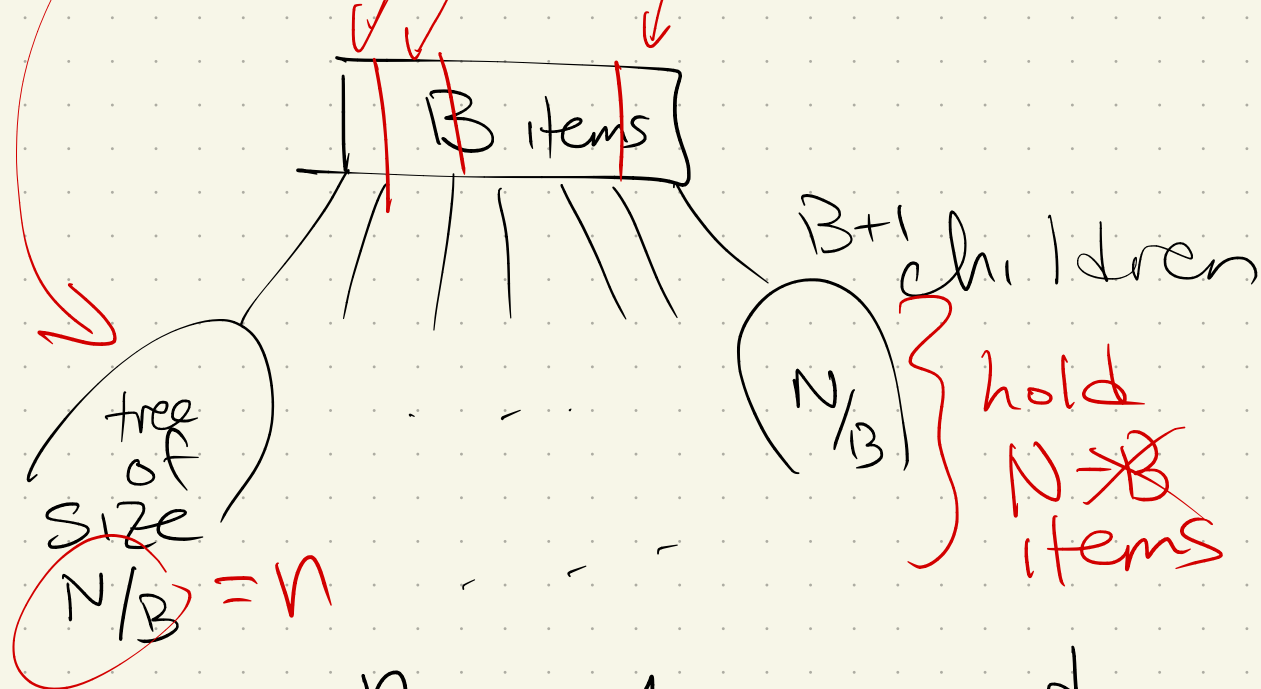
B-trees
(cont)

Recap: B-trees

(sorted) data: N, so N/B blocks $= n$

1, 2, ..., B

Tree: Take B evenly spaced items. In between, remainder is =



depth: $\dfrac{n}{B^d} = 1 \Rightarrow n = B^d$

$\log_B n = d$

Insert runtime:

- $O(\log_B n)$ to find
- Then split $O(\log_B n)$ blocks

"Time" to split:

↑ # block accesses

I/Os

$$\leq 4 \log_B n$$

$$= O(\log_B n)$$

$$= \frac{\log_2 n}{\log_2 B}$$

identity

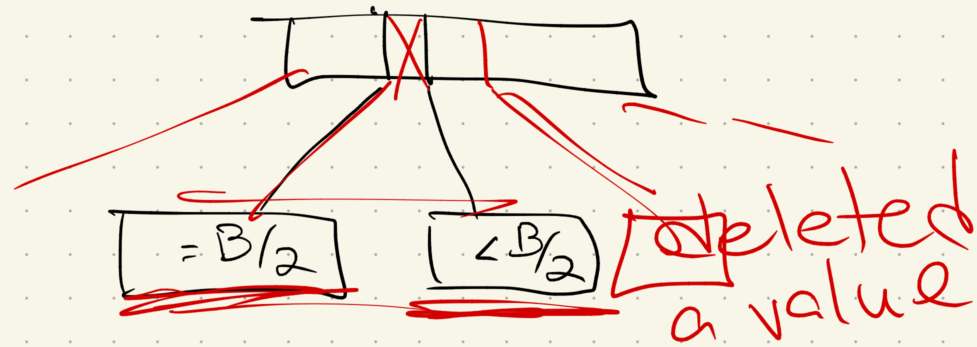
$$\boxed{\log_c d = \frac{\log d}{\log c}}$$

Delete: Opposite of insert:

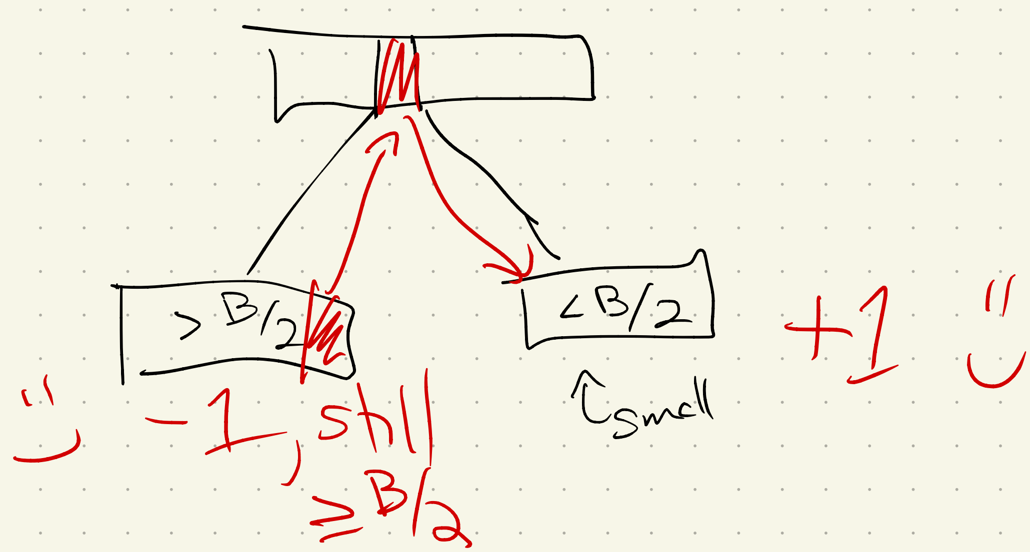
Find x & delete it.

If size is < B/2:

- there is either an immediate sibling of size = B/2



- or an immediate sibling of size > B/2



Agan, delete can propogate up, since we may need to remove a key from the internal node (if 2 merged)

Path to root has size:

$$\Rightarrow O(\log_B n)$$

Even cooler:

Suppose we're back in RAM-model, + have to pay for searches inside a block.

Find:

Know: $O(\log_B n)$ blocks to load I/Os

Inside each block: size B array.

We need to find if x is in array. How?

→ $O(\log_2 B)$ time

Bin Search in array of size B

Total seach:

$\cancel{B} \cdot \log_B n \times \log_2 B$

$= \frac{\log_2 n}{\log_2 B} \times \log_2 B$

$= O(\log_2 n)$

Same as balanced BST

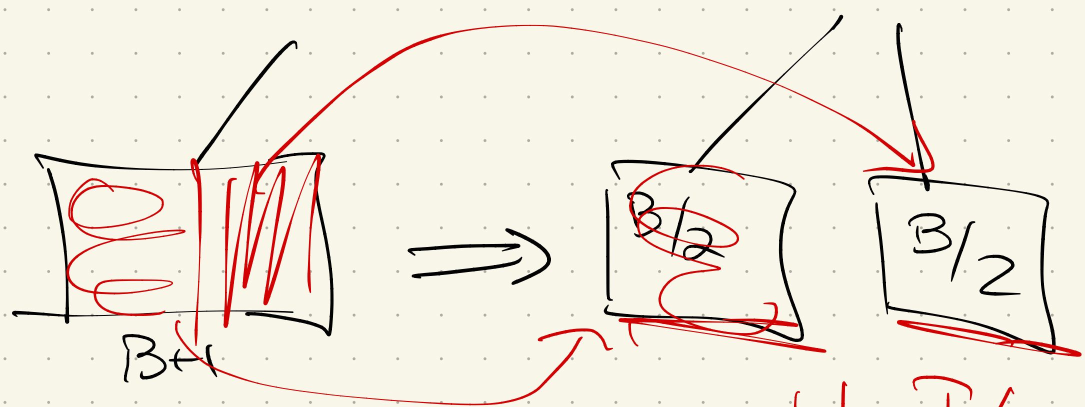
(worstcase)

Insert: A bit more complex:

$O(\log_B n)$ loads

Then traveling back up:

if leaf is full: split



How long?

copying an array:

Initial new size B array

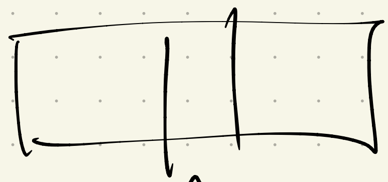
copy B/2 elements

Runtime: $O(B) \cdot O(\log_B n)$

Delete:

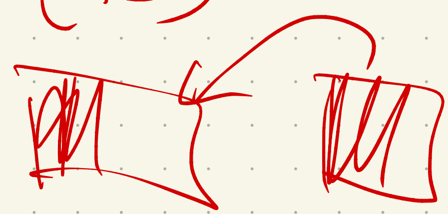
$O(\log_B n)$ loads

Inside each:



remove

: $O(B)$ time



Again, $O(\log_B n)$ of these

$\Rightarrow O(B \cdot \log_B n)$

So Bad news: (in RAM-model)

Find : $O(\log n)$

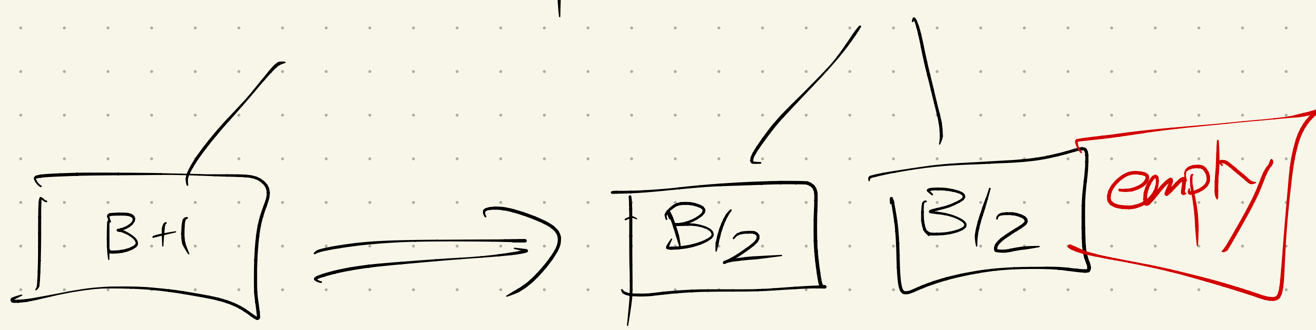
Insert: $O(B \log n)$

Delete : $O(B \log n)$

Well, really?

Think of insert :

after we split



things are empty!

(Remember that push-back in a vector is worst case $O(n)$, but amortized $O(1)$ time?)

Thm: Any sequence of m Insert/Remove operations results in $O(m)$ splits, merges, or borrows.

Result: $O(\log n)$ amortized time per operation

Proof: Accounting version again.

Each insert "pays" \$3 (instead on \$1)

By the time a node buffer is full, has built up $\$(3-1) \times (B/2) = \$B$ to pay for its split/merge.

Practical notes

These are (arguably) the most used BST!

- File systems: Apple's HFS+, MS's NTFS, + Linux Ext4
- <u>Every</u> major database system
- Cloud computing

See linked reference (in "Open DS") for code: Java, Python, or C++

One reason: these work better than expected.

- B is usually big: 100's or 1000's, at least
- So 99% of data is in the leaves

Result:

- Load entire tree in RAM/local memory
- Then a single leaf access to get data

Variants:

- B^+ trees
- B^* trees
- (a,b)-trees (next time)