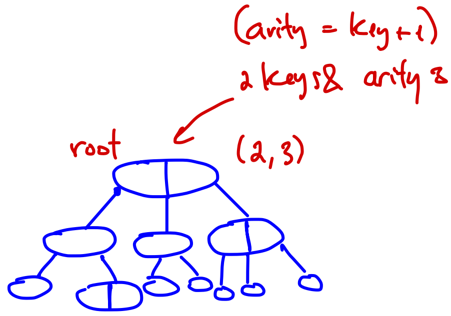
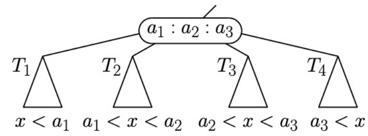
L3: (a,b)-Trees

(a,b)-Trees → Binary Search Tree
→ AA, Left-leaning RB tree, Red-Black Tree
→ B-Tree & B^+-Tree



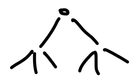
Defining (a,b) trees. (Many variants out there)

- $b \geq 2a - 1$ and $a \geq 2$
- Balanced trees
- Every non-root node has arity between a and b (inclusive)
- The root has $\leq$ b children and at least 2 children
- All leaves are at the same depth



Lemma: An (a,b)-tree with n nodes has height $O(\log_a n)$.

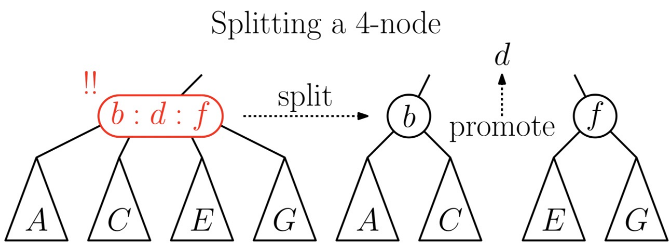
"Proof": In the "sparsest" tree, every inner node has a children.



- $1 + 2 + 2a + 2a^2 + \ldots + 2a^h \simeq n \Rightarrow 1 + 2\frac{a^{h+1} - 1}{a - 1} = n$

$\Rightarrow h = O(\log_a n)$.

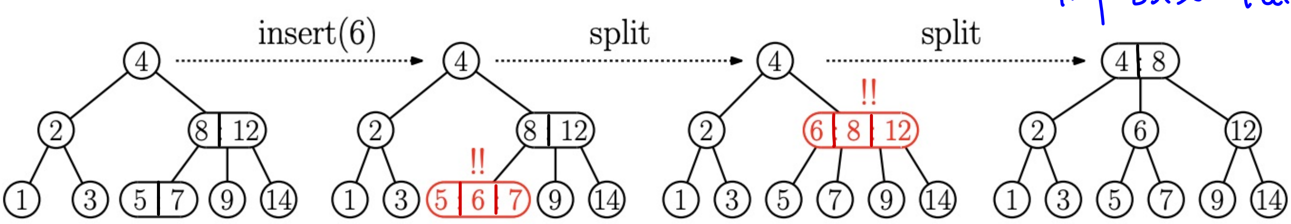
Insertion: To insert key k: (1) search for k (will end up at a leaf & fall out)
(2) insert k into this leaf (3) if overflowing, restructure!



- too full, split into two nodes
  # keys = b ⇒ promote 1 & have $\lceil \frac{b-1}{2} \rceil$ & $\lfloor \frac{b-1}{2} \rfloor$ left
- promote the middle

$\left(\frac{b-1}{2} \geq \frac{2a}{2} = a \right.$

Ex:



* possible height increase!

Deletion. (1) Locate the key we wish to delete
(2) If not a leaf, find the replacement node (successor)
(3) Recursively delete the replacement node

∴ WLOG: restructuring will begin at a leaf.

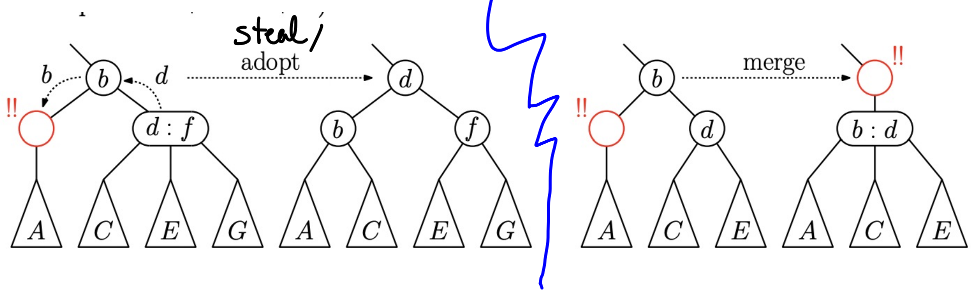
[Two cases & strategies : repair because after deletion, node is too sparse.

① Steal from sibling — if ∃ a sibling to steal from.

$d_{me} = a-1 (+1) = a$ ✓ (+1: stolen)

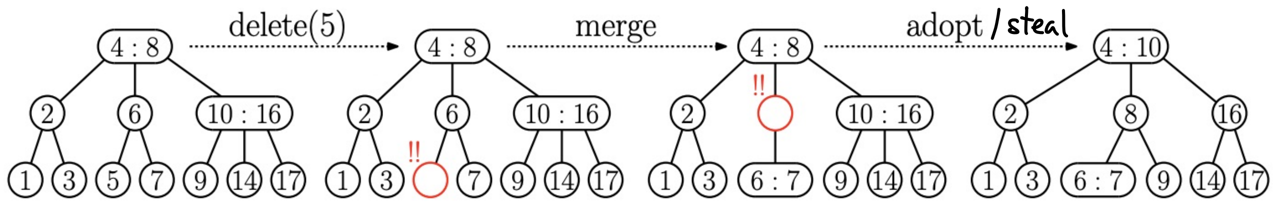
② Merge with sibling. — if sibling is too sparse to steal from

$d' = a$ (sib) $+ d = a-1$ (me) $\Rightarrow d' + d = 2a-1 \leq b.$ ✓



* possible height decrease.

Ex:



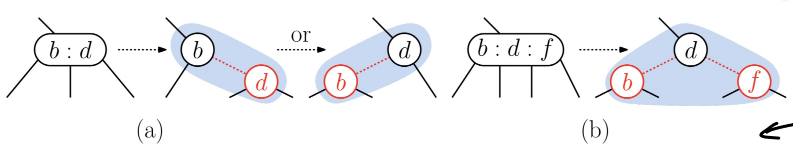
Remarks: • (2,3) or (2,4) trees aren't binary

• What's the deal? K[ ] keys && Node[ ] children, ⇒ . wasted space? . weird logic

(K[ ]: 2 or 3; Node[ ]: 3 or 4.)

• What if they can be "seen" & stored as BSTs?

Red-Black tree (1978) • Guibas & Sedgewick



(a) (b)

(2,4) tree

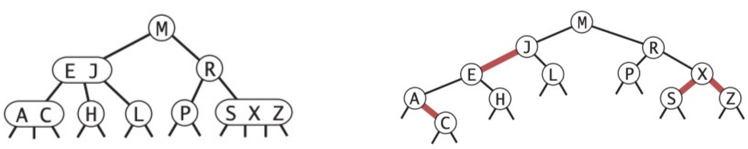
• AA tree (after its inventor Arne Anderson, 1993)

⇒ if a node is red, it's the right child of a black node (rule out (b) above)

⇒ (2,3) - tree

• LLRB (Sedgewick, 2008)

Left leaning Red-Black tree



Discussion: (1) (a,b) is super configurable

(2) Low values "reconstruct" BBST

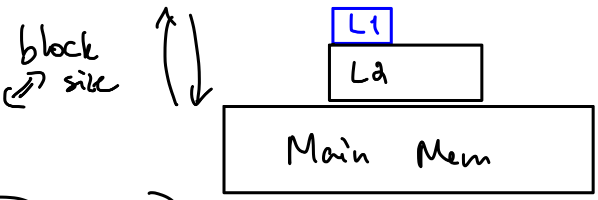
(3) Any good use for high values of a & b?

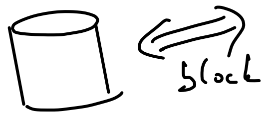
Detour Architecture 101:

- cache line size (32, 64, 128 bytes)
- storage block size

e.g. SSD: 512KB?

Older SATA/IDE: 4KB?





time ~ # of blocks / cache lines "touched"

B-Tree: • (B)BST → too deep (i.e. touching too many blocks)

- Each node doesn't fully utilize the width of a line/block.
- Invented by Bayer and McCreight (Boeing Research) in 1970.
- (a,b) tree where $a = \lceil b/2 \rceil$ (i.e., $b = 2a$ or $b = 2a-1$)
- Typical $b \approx 100$

Theorem 1 *Starting with an empty tree, the amortized number of rebalancing operations in an (a, b)-tree is:*

1. $\Theta(\log_a n)$ *when* $b = 2a - 1$.
2. $\Theta(1)$ *when* $b = 2a$.
3. $\Theta(1/(\epsilon a))$ *when* $b = (2 + \epsilon)a$, *for* $\epsilon > 0$.

Applications

- File systems: NTFS, ext4, APFS
- DB indexes
- NVRAM layout

B^+-Tree: idea • Keep data only in the leaves

Why? • Chaining of the leaves.

* Because B+ trees don't have data associated with interior nodes, more keys can fit on a page of memory. Therefore, it will require fewer cache misses in order to access data that is on a leaf node.

* The leaf nodes of B+ trees are linked, so doing a full scan of all objects in a tree requires just one linear pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree. This full-tree traversal will likely involve more cache misses than the linear traversal of B+ leaves.