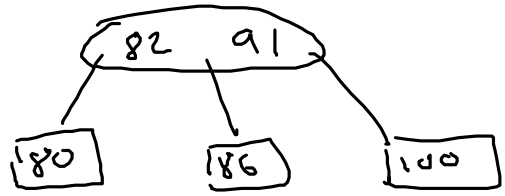
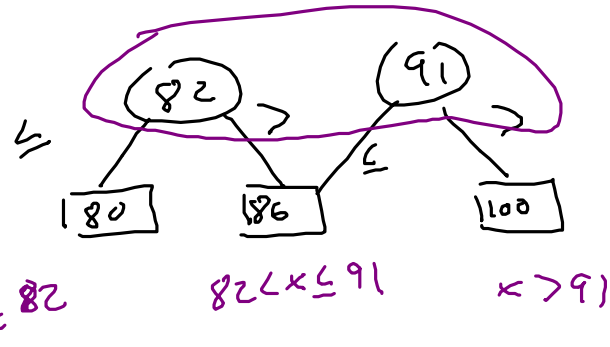
B-Trees





order : b (odd number)

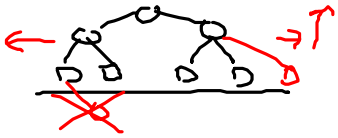
bounds on # of children of a node

underflow $\frac{b+1}{2} \longleftrightarrow b$ overflow

b	$\frac{b+1}{2}$	b
3	2	3
5	3	5
11	6	11
101	51	101

"node is at least half full"

Rules for B-Tree of order b



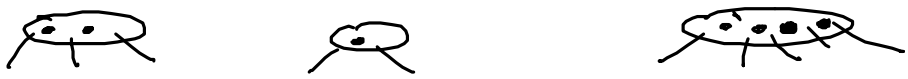
1) All leaves at same depth

2) All non-root internal nodes (internal node = non-leaf) have between $\frac{b+1}{2}$ and b children

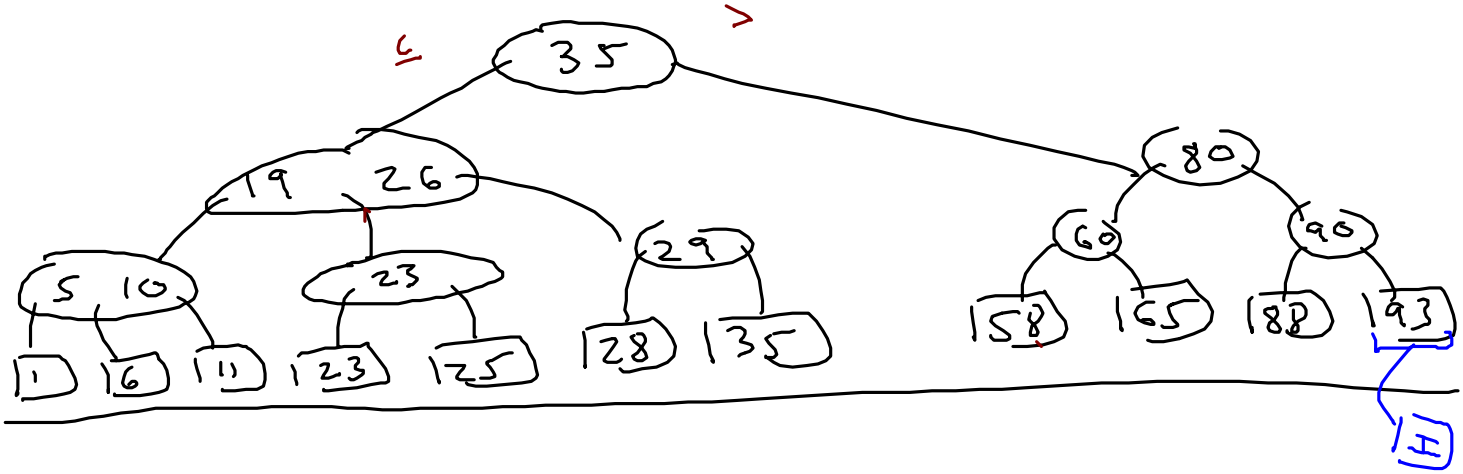
3) root can have between 2 and b children

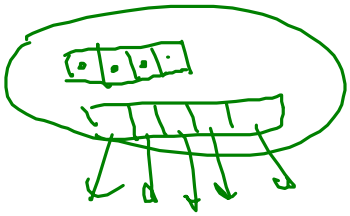
4) all real data is in leaves (one per leaf); internal nodes merely guide search

5) in every internal node, # of index values is one less than # of children



B-Tree of order 3





Insert (x)

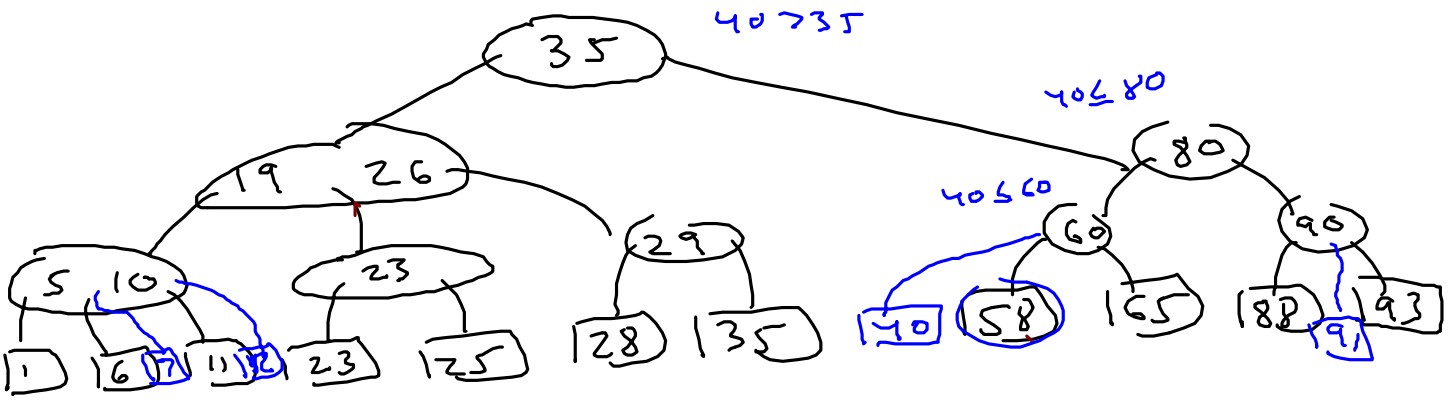
- traverse down to where x "should be"
  i.e. search for x
- if found, return (won't insert duplicate)
- else (not found) insert leaf with this new value in "same place" as leaf that "should have" contained x
- add new index value to parent (whatever is in leaf to <u>left</u> of new index value's position)
- label parent of leaf "p"

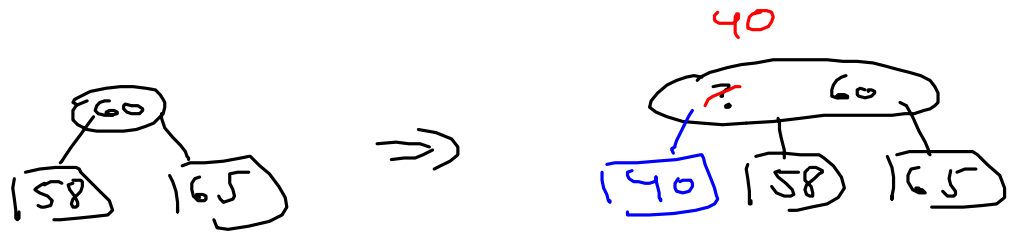
<u>cases:</u>

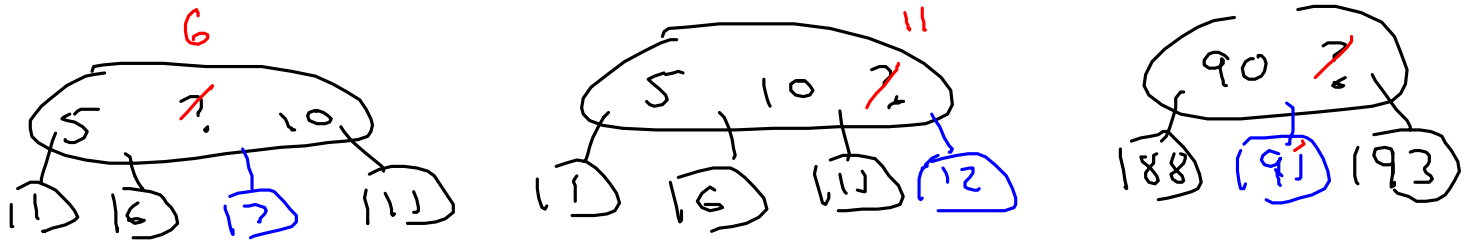
case 1: if p doesn't overflow, STOP

case 2: p overflows ⇒
SPLIT, index moves up into parent of p
relabel p's parent as p
CONTINUE

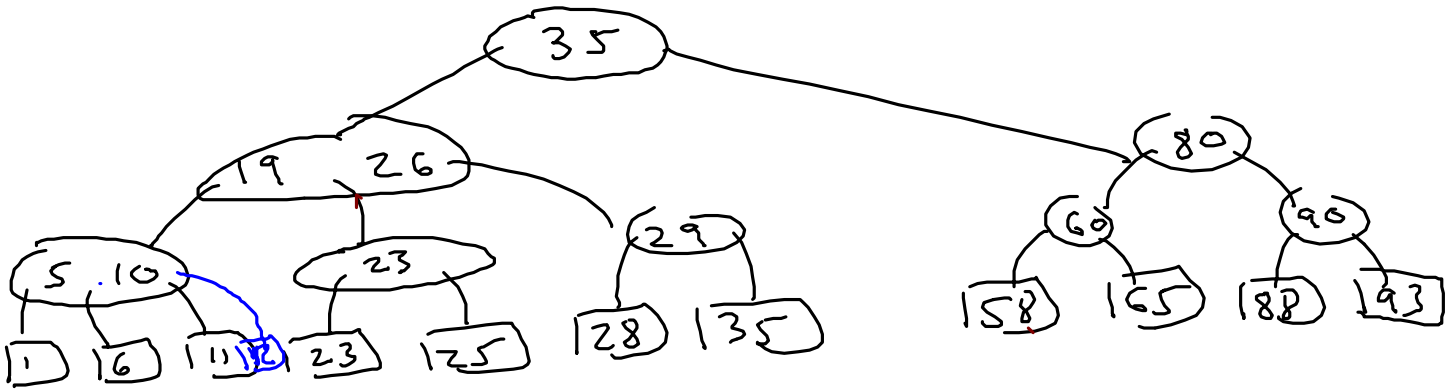
B-Tree of order 3

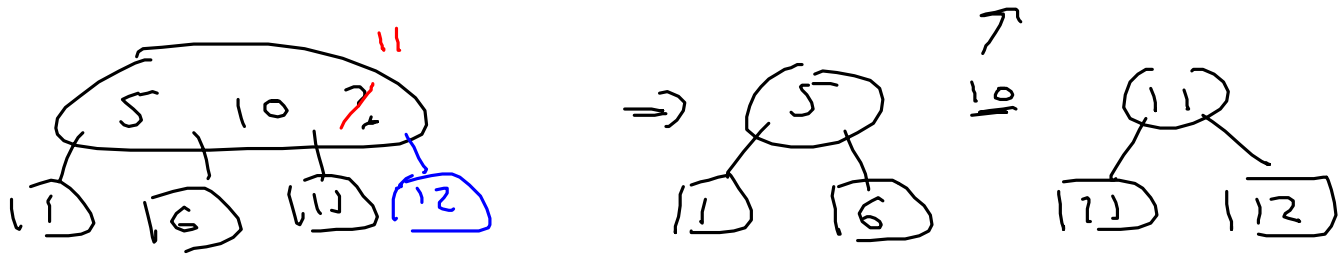




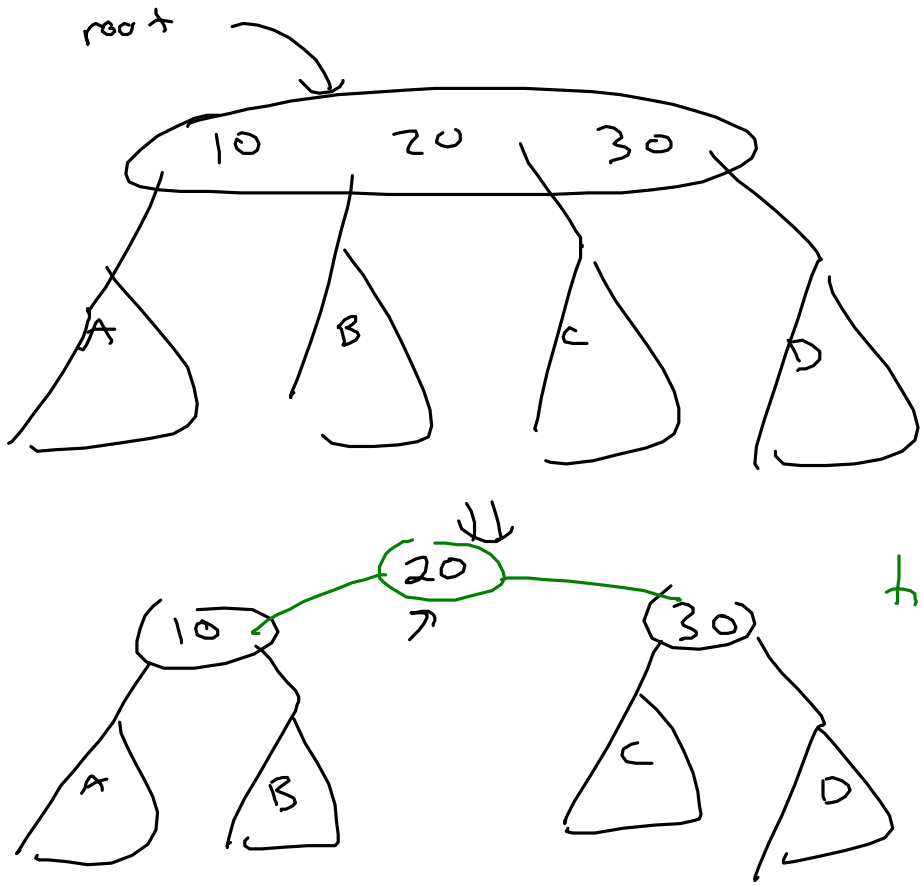


B-Tree of order 3





If root splits → new root



height ++;

Remove(x)

→ search for x

→ if not found, return

→ else (found) remove leaf, and index from parent

→ label parent P

cases

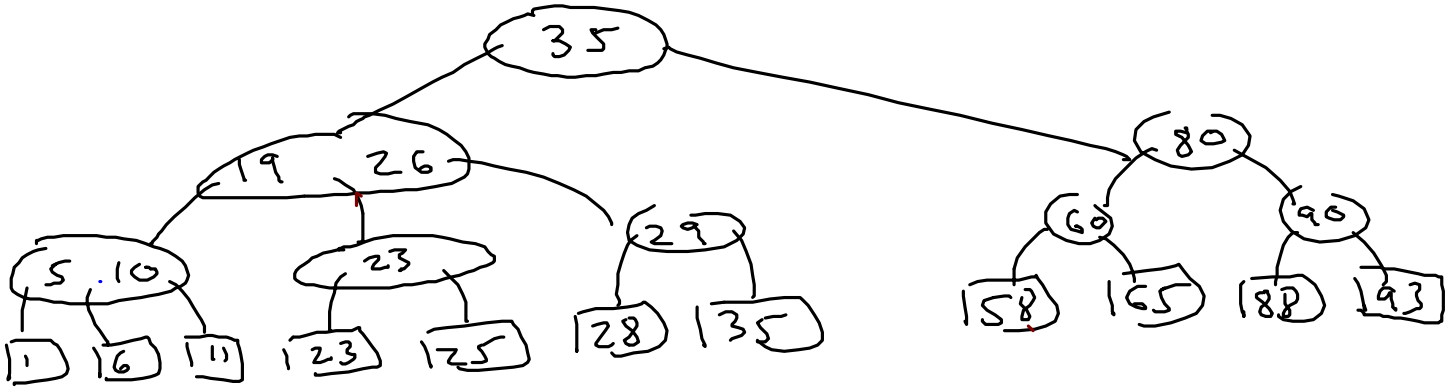
1) if P does NOT underflow, STOP

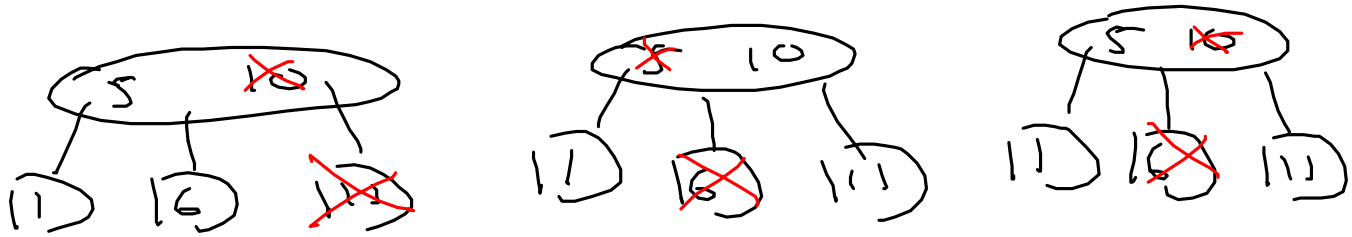
2) if P does underflow, but we can steal from immediate sibling, steal, STOP

3) else COMBINE two nodes parent loses child & index, label parent of P as P

CONTINUE

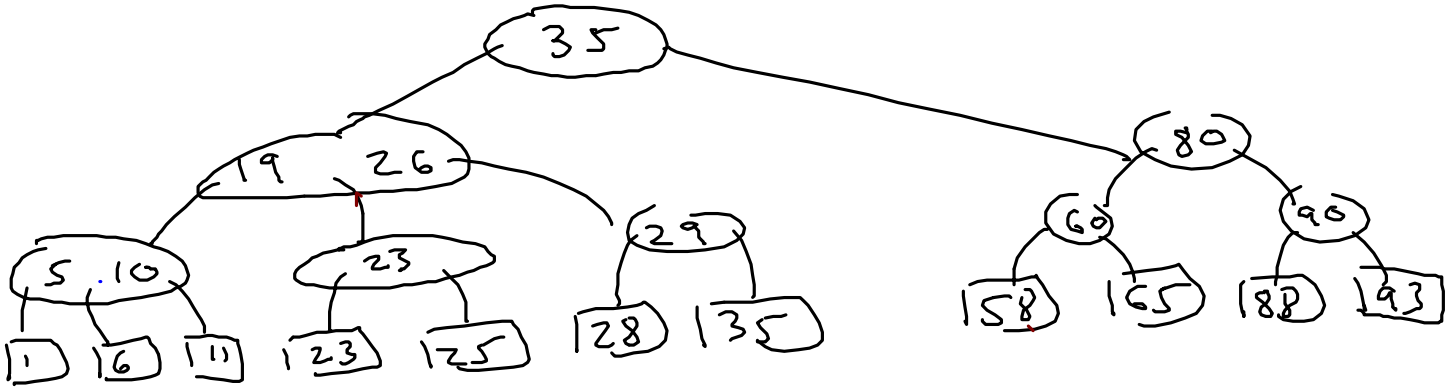
B-Tree of order 3

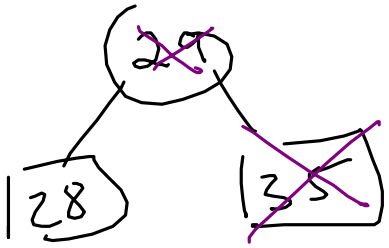


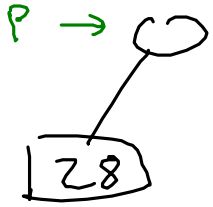


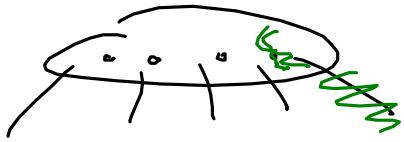
1

B-Tree of order 3



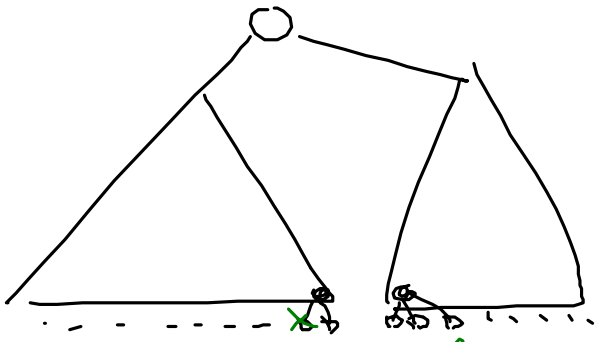






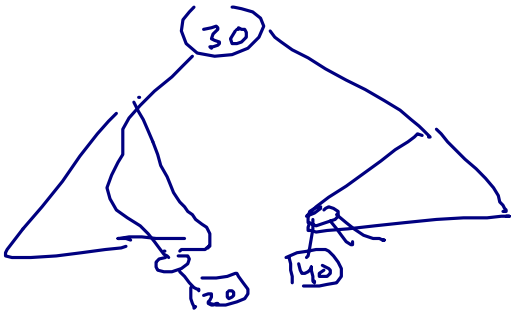
1

Immediate Sibling

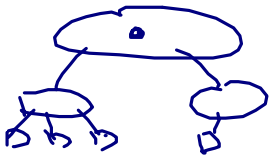


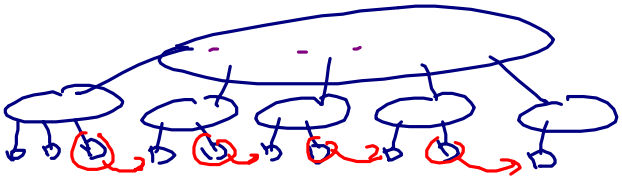
remove from here

not a sibling!

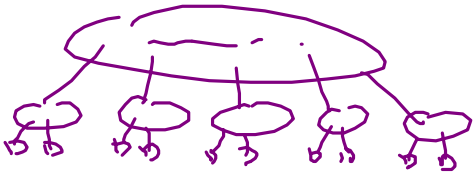


NO!

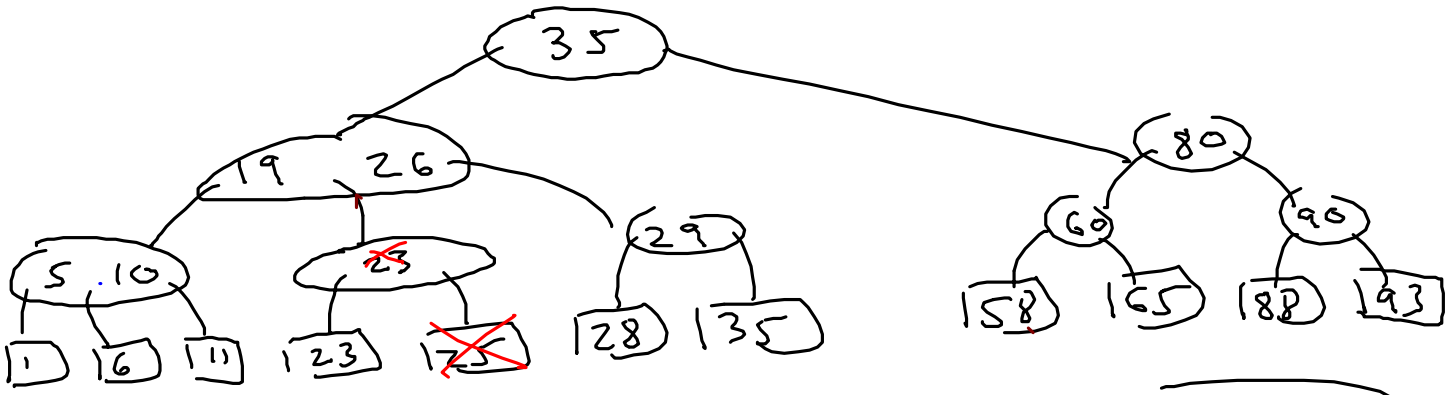


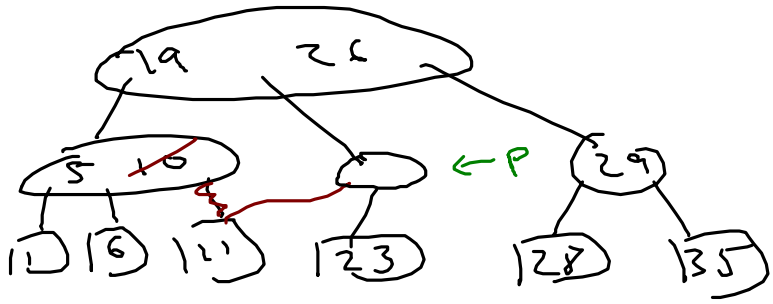


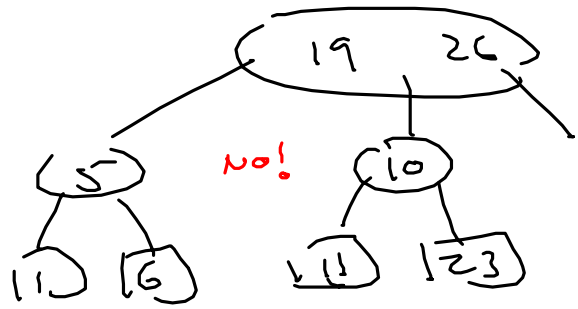
NO!

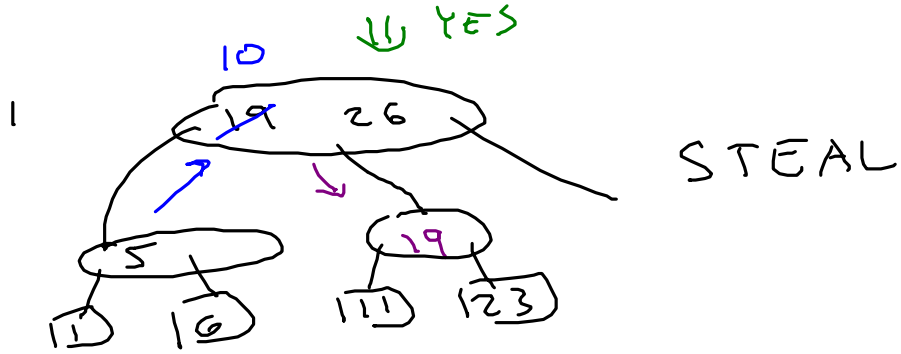


B-Tree of order 3



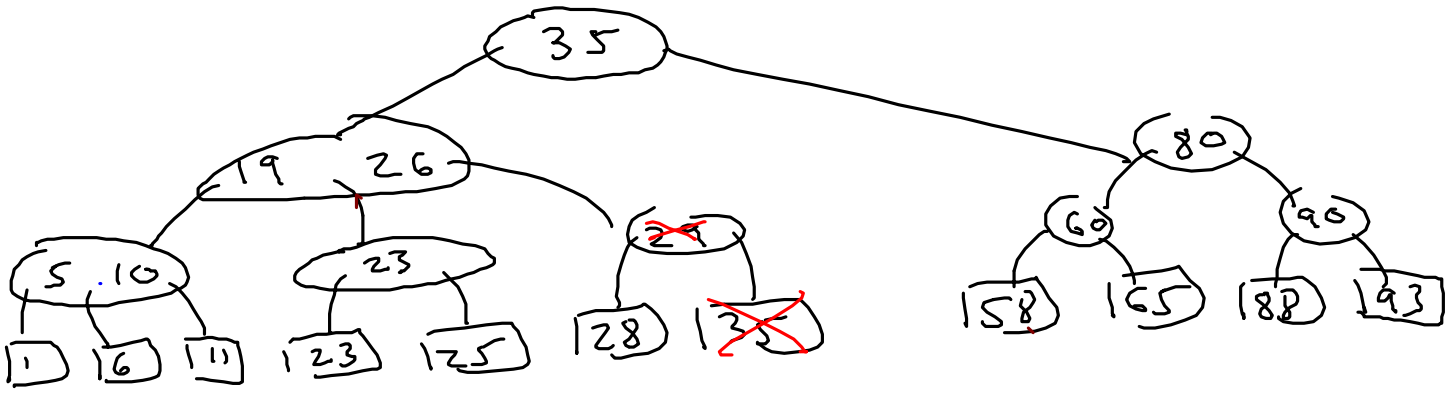


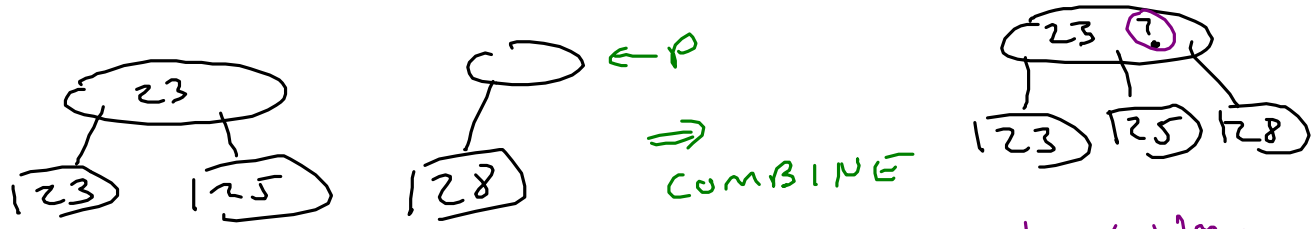




STEAL

B-Tree of order 3



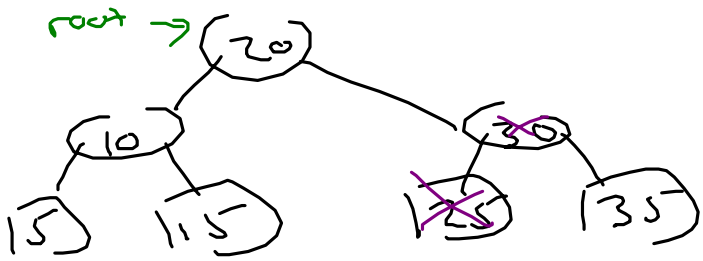


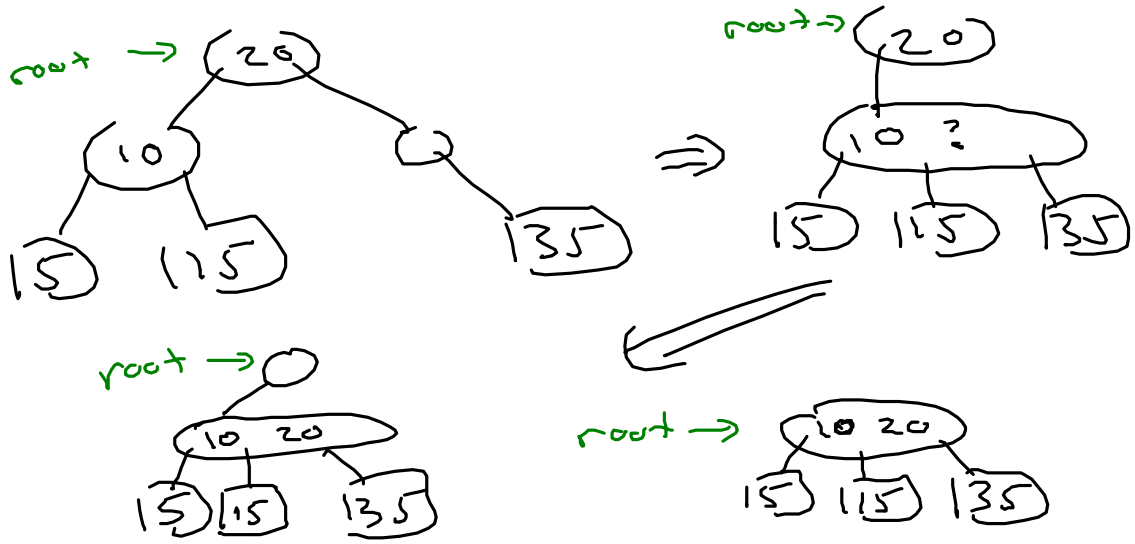
$\frac{b+1}{2}$ children $\qquad$ $\frac{b+1}{2}-1$ children $\rightarrow$ b children

$\frac{b+1}{2}-1$ indices $\leftarrow P$ $\qquad$ $\frac{b+1}{2}-2$ indices $\rightarrow$ b−2 indices

26

Root underflow

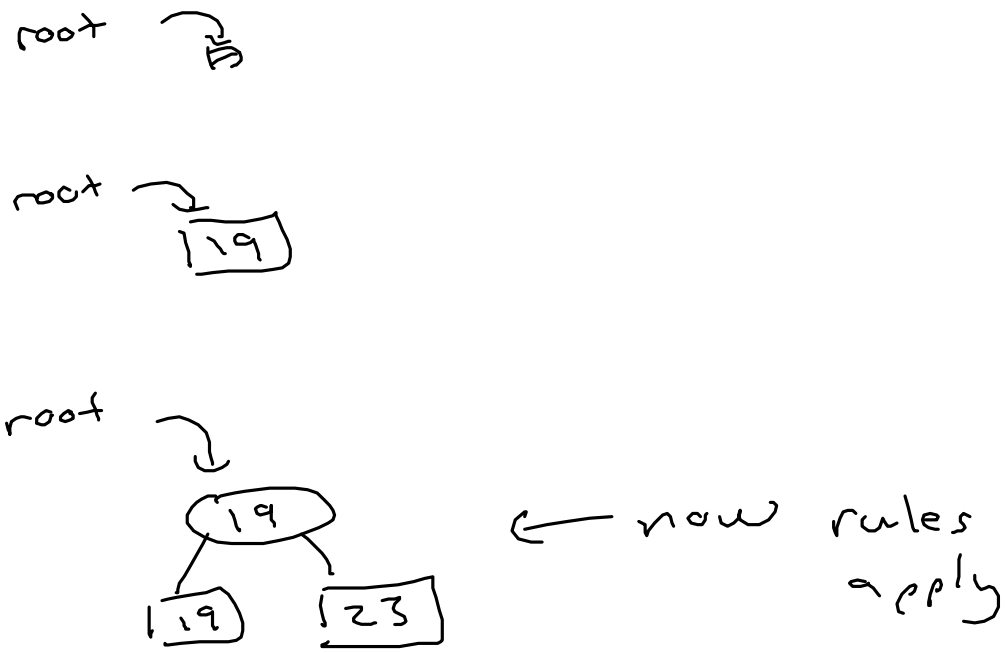




underflowing root is deleted, its 1 child is new root

height --;

Degenerate cases



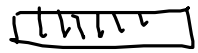
1 million values

List : 1 million nodes in unsuccessful search

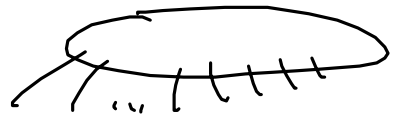
AVL tree : height ≈ 20

B-Tree order 199 : height ≈ 4

Binary search



↳ but node :



199 children

198 index values

loooooong time to decide subtree

AVL

main memory is an order of magnitude faster to access than disk

let's reduce # of disk reads when lots of info on disk

AVL : height 20 → 20 levels, 20 reads (1 million values)

B-Tree of order 199 → <u>if</u> each node only as large as a disk block, then 4 levels, 4 disk reads (1 million values)

↳ searching node is <u>much</u> faster than getting it into memory in first place