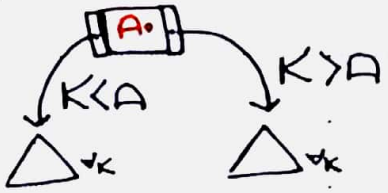
ADB Sheet 2 Part 2

Setup

- In a B-tree
  - → Each node has at most P Pointers (to node)
  - → As well as P-1 Search Keys (each with its data Pointer P_r)
    - For any of them:



- In a B+-tree
  - → Each internal node has at most P Pointers
    - At any time $\lceil \frac{P}{2} \rceil$ of them must be full
    - ↘ implies that each internal node has at most P-1 Search Keys (with no Pointers to data)

⇒ To know P given block size (B), Pointer size |P| and Key size |V|, solve

$$P|P| + (P-1)|V| \leq B$$

- The fan out of an internal node is the no. of full Pointers in it ($P_o = P_{Full}$)

  - → Each leaf node has at most 9 Keys
    - Each comes with data Pointer (↳ will call it P_{leaf})
    - there's also a Pointer to next leaf
    - Any leaf must have $\lceil 9/2 \rceil$ Keys or more

⇒ To know P_{leaf} given $B, |V|, |P_L|, |P_{next}|$ (← data ptr size (leaf); ← next leaf pointer size)

$$P_{leaf}(|P_L| + |V|) + |P_{next}| \leq B$$

- include P_{prev} if mentioned

- Note that internal nodes have no data pointers
  → hence, leaf nodes will have as much data pointers as 1st level index. (#values or #blocks)

- The no. of blocks accessed to search for a record is the no of levels (by recalling how search works)

9) $B = 512$ bytes
$P = 6$ bytes // to block
$P_r = 7$ bytes // to data record
$r = 30K$ records

Name	SSN	Dept-Code	...
30 byte	9byte	9byte	

→ Suppose file isn't ordered by key field SSN // 2ndary index
→ Want to construct a B+ tree on it

i) The orders P and P_{leaf} of B+ tree
→ recall, any node is a block

- For intermediate nodes, must satisfy

(node ptr is block ptr; ← 9 bytes (SSN))

$$P|P| + (P-1)|V| \leq B$$

$P(6) + (P-1)9 \leq 512$

$15P \leq 521 \rightarrow P \leq \frac{521}{15}$ • $P = 34$ at most (uses 501 bytes)

• For leaf nodes

next leaf is a block

$P_{leaf}(|P_L| + |V|) + |P_{next}| \leq B$

inconsis

$P_{leaf}(7+9) + 6 \leq 512 \rightarrow P_{leaf} \leq \frac{506}{16}$

↳ record Pointer

• $P_{leaf} = 31$

ii) The no. of leaf level blocks if blocks are approx. 69% full

→ hence, on average a leaf block has $\lceil 0.69\, P_{leaf} \rceil = 22$ Keys and Pointer for each

• Ceiling is safer as flooring means < 69%.

• This is a secondary index (Key) • unsorted file

→ hence each data record has a Key value (SSN)
i.e. the tree's leaves include $r = 30K$ Keys

thus,

$$\#\text{leaves} = \frac{\lceil \#\text{Keys} \rceil}{\#\text{Keys/leaf}} = \frac{\lceil 30000 \rceil}{22}$$

$= 1364$ blocks

iii) # levels needed (internal nodes de 69% Full)

→ Fanout for intermediate node

$$f_o = f_{Full} = \lceil 0.69p \rceil = 24 \quad (p = 34)$$

b_{i1}

- 1st level (leaves) → 1364 blocks (i.e. needs 1364 Pointers from 2nd level)

$$b_{i2} = \left\lceil \frac{b_{i1}}{f_o} \right\rceil = \left\lceil \frac{1364}{24} \right\rceil = 57$$

blocks in 2nd level (each needs a Pointer from 3rd level)

$$b_{i3} = \left\lceil \frac{b_{i2}}{f_o} \right\rceil = \left\lceil \frac{57}{24} \right\rceil = 3$$

blocks in the 3rd level (each needs a Pointer from 4th level)

$$b_{i4} = \left\lceil \frac{b_{i3}}{f_o} \right\rceil = \left\lceil \frac{3}{24} \right\rceil = 1$$

- 4th level has one block (root)

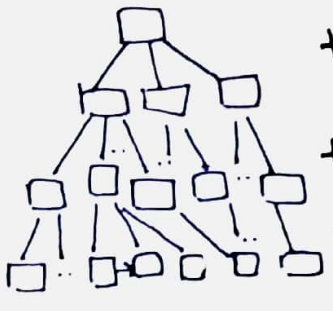
→ In total 4 levels for the tree

⇒ Clearly once we have b_{i1}, f_o then

$$b_{i(j+1)} = \left\lceil \frac{b_{ij}}{f_o} \right\rceil \quad \text{until } b_{ij+1} = 1 \text{ (TOP level)}$$

- Can thus get # levels X by $X = \lceil \log_{f_o} b_{i1} \rceil + 1$ (add level)

iv) Total no. of blocks required by tree



$+ 1$
$+ 3$
$+ 57$
$+ 1364$
$= 1425$ blocks

v) No. of blocks accesses for search

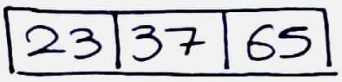
is also the no. of levels $+ 1 = 5$
(no. of levels: x)

to access actual file from leaf

14.15) Insert 23, 65, 37, 60, 46, 92, 48, 71, ... into B+ Tree

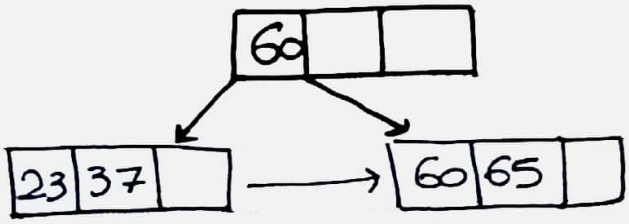
- P=4, P_{leaf}=3

1. Insert 23, 65, 37



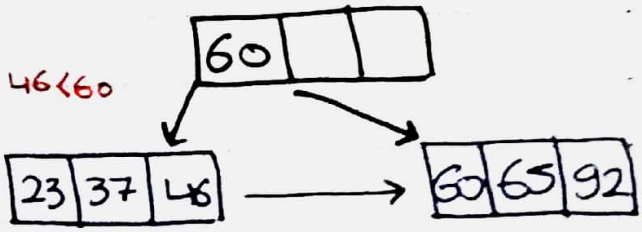
2. Insert 60

- Split
- Copy up 1st child of 2nd leaf (60)



3. Insert 46, 92

46<60

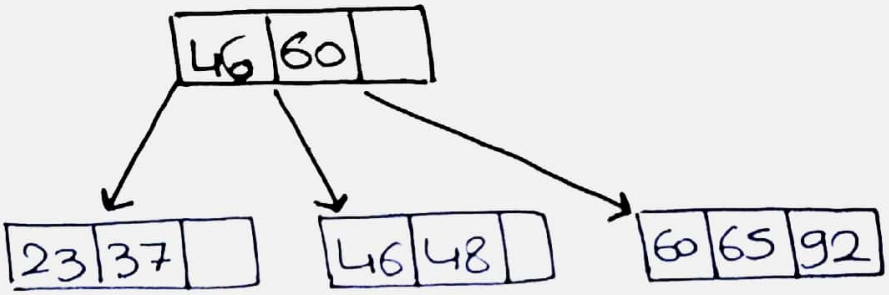


• Insert 48

48 < 60

→ Should go to 1st leaf but no space so split

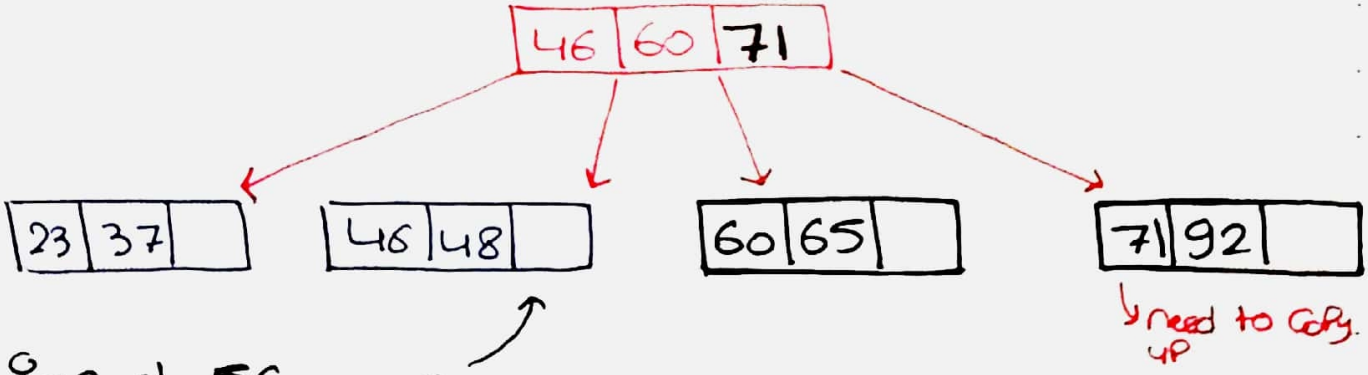
↖ will need to copy up
• Parent has space



} last level: always → connections (even if ignored)

• Insert 71

→ 71 > 60, should go to 3rd leaf but no space
So split (60, 65, 71, 92)



• Insert 56

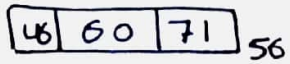
→ 56 > 46 & 56 < 60 So it goes to 2nd leaf (there is space)

• Insert 59

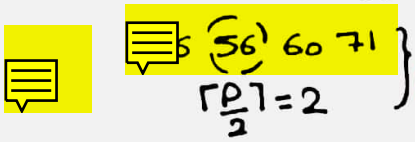
→ 59 > 46 & 59 < 60, goes to 2nd leaf (full) So we need to split (46, 48, 56, 59) → 46 48 | 56 59

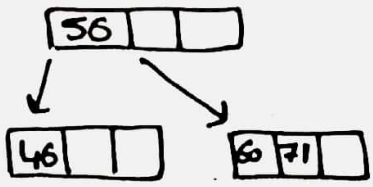
- Will need to copy up 56 but Parent has no space

→ Split Parent

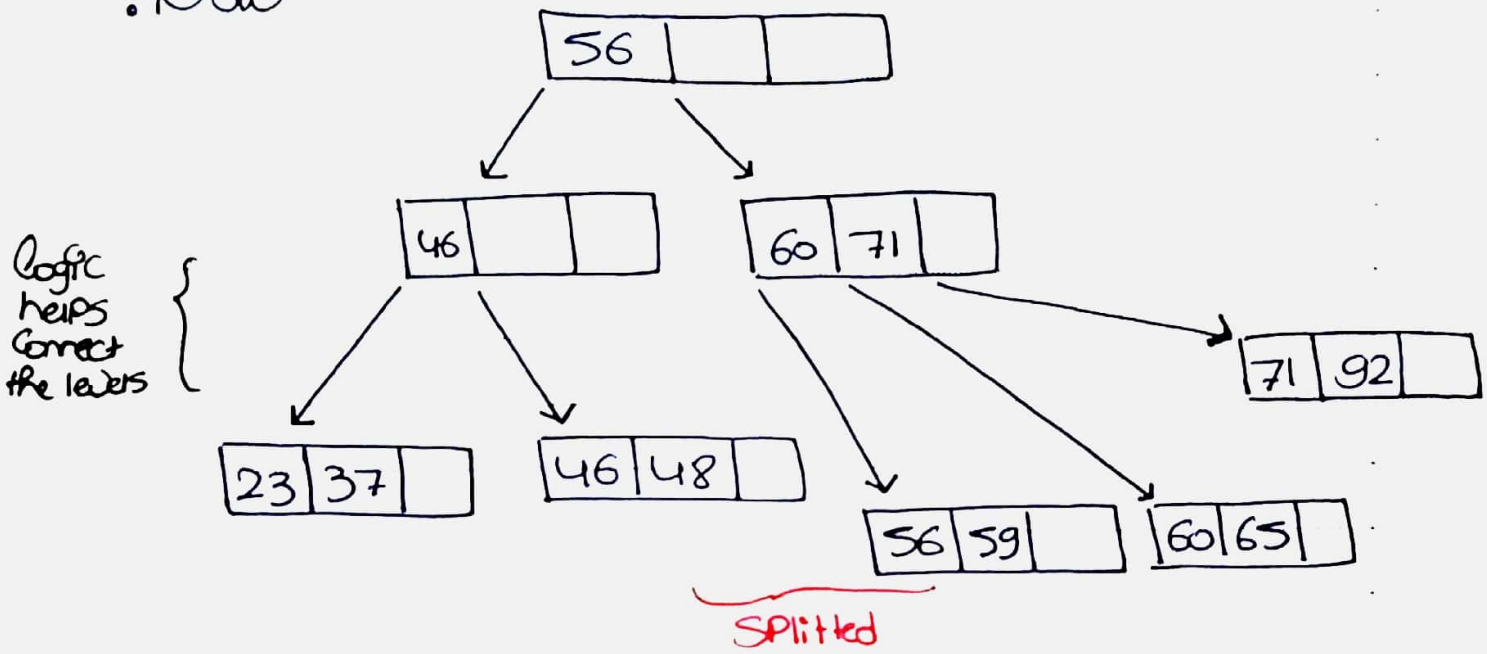


→



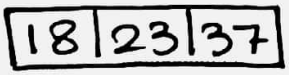


- Now



- Insert 18

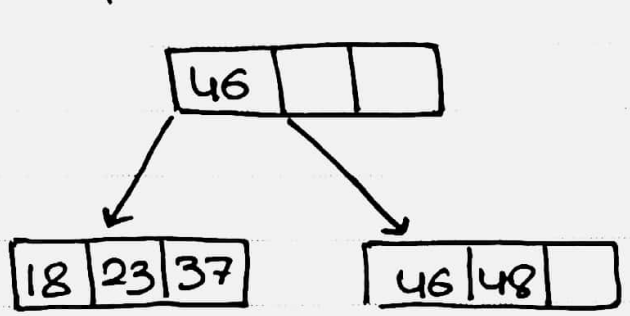
→ 1st leaf becomes



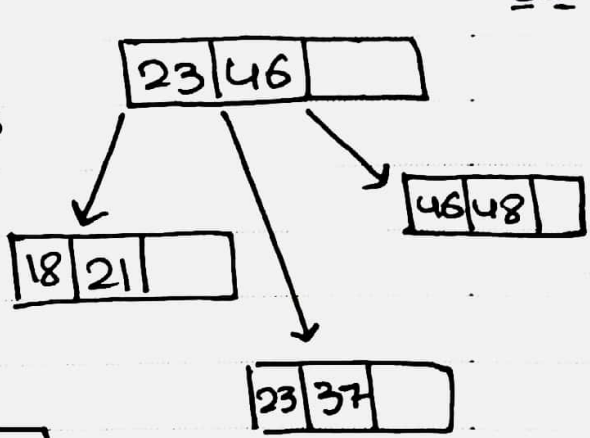
- Insert 21

→ Need to Split 1st leaf (Parent has space)

18 21 23 37



becomes →

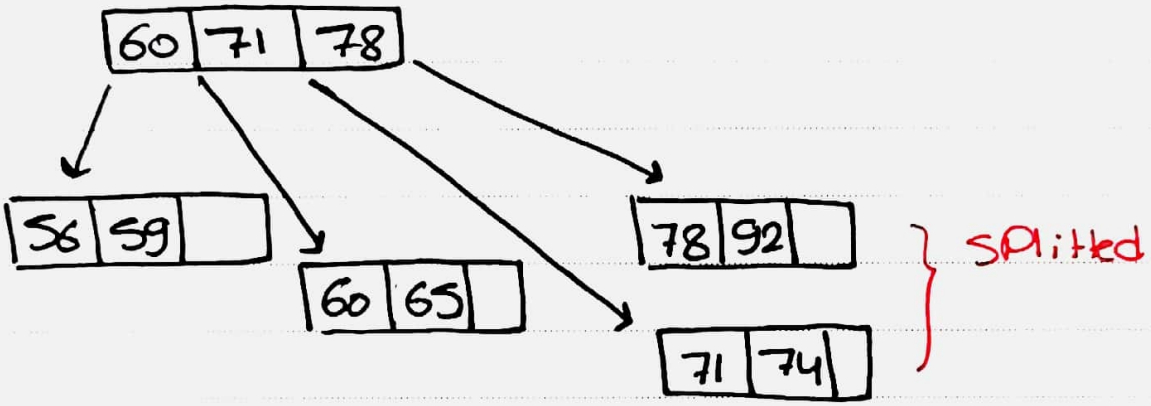


- Insert 10

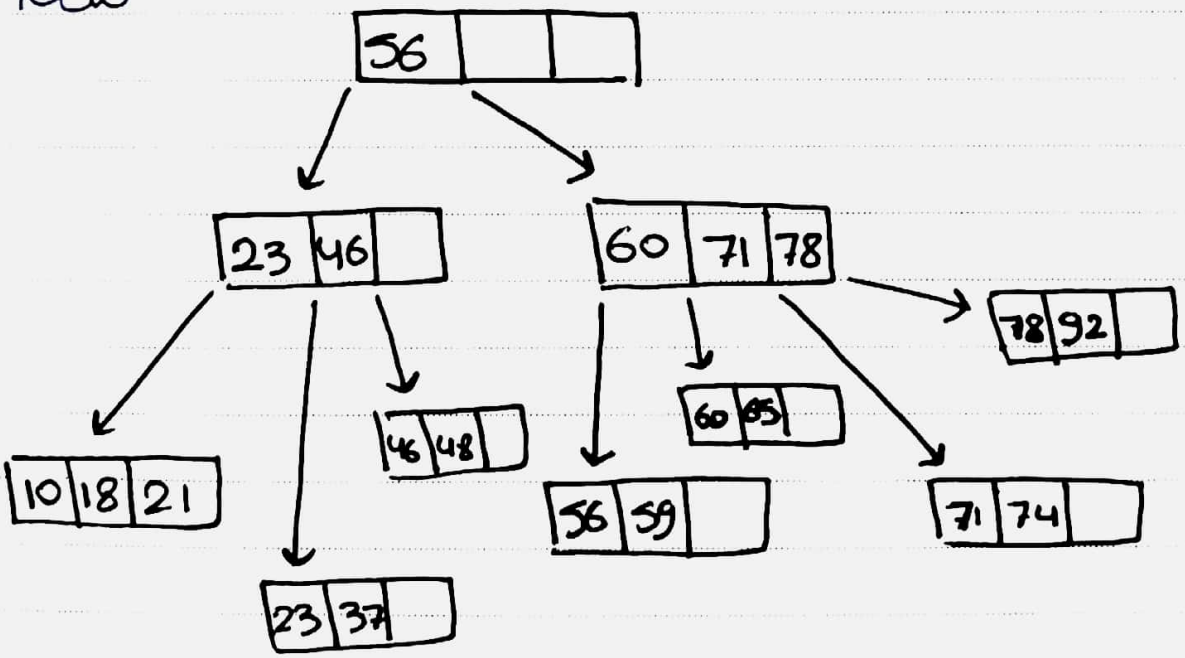
→ 1st leaf becomes

• insert 74
→ has space in last leaf (becomes 71 74 92)

• insert 78 ← 71 74 78 92
→ need to split last leaf (Parent has space)

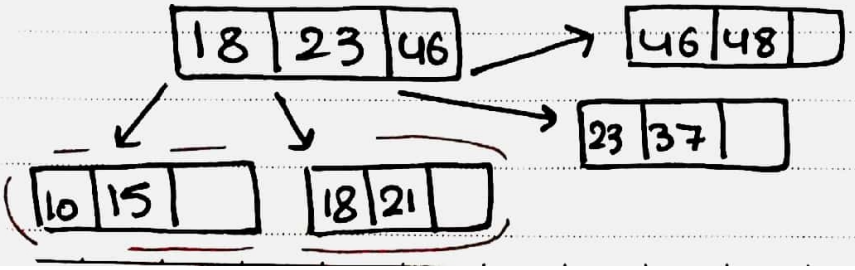


* Now

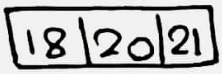


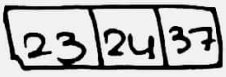
• insert 15
→ 1st leaf is full but Parent has space (split)

10 15 18 21



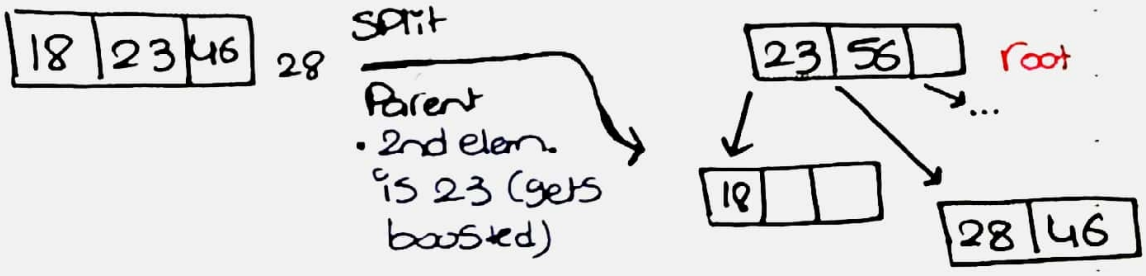
- Insert 16
  → 1st leaf becomes 

- Insert 20
  → 2nd leaf becomes 

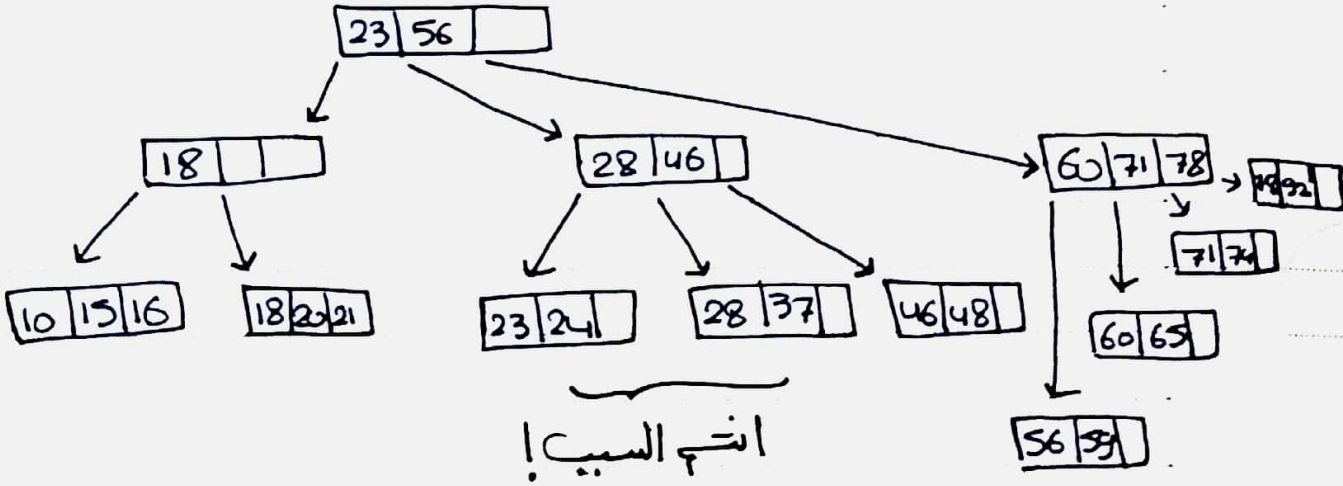
- Insert 24
  → 3rd leaf becomes 

- Insert 28
  → No space in 3rd leaf

→ Need to copy up 28, no space at Parent



- Now tree becomes



and so on (have covered enough insertion cases, Patrol

This is an excerpt of TA slides with personal notes.

Question 14.17

14.17 Suppose the following search field values are deleted in the given order from the B + -tree of Exercise 14.15, show how the tree will shrink and show the final tree. The deleted values are: 65, 75, 43, 18, 20, 92, 59, 37, 10.

Summary of Deletion Cases:
1 – Leaf if rich: just delete key.
2 – Leaf is poor but has rich sibling: delete key, steal key from it.
3 – Leaf is poor with poor siblings: delete key, merge with right sibling and remove a key from parent.

In the third case, parent should remain rich after removing key.

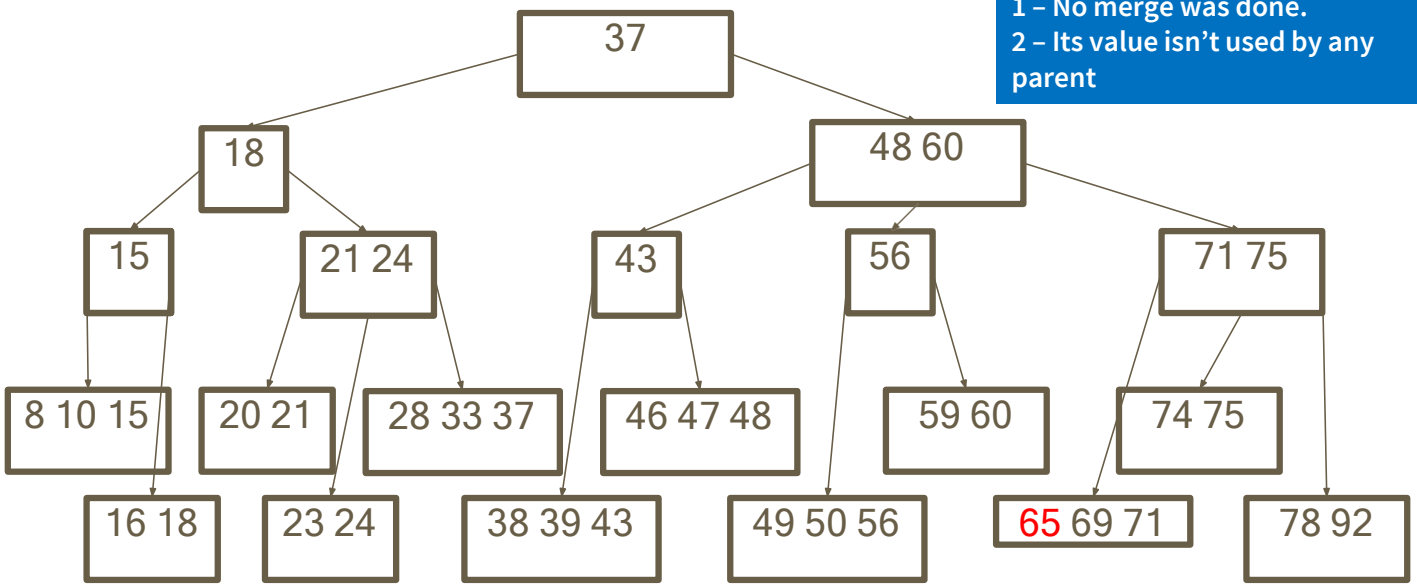
In all cases, if the deleted key was used by an ancestor then it must be updated as well.

Remember that they key in any intermediate node carries the minimum number of in its right sub-tree. Use this fact while updating parents.

Okay.

Delete 65

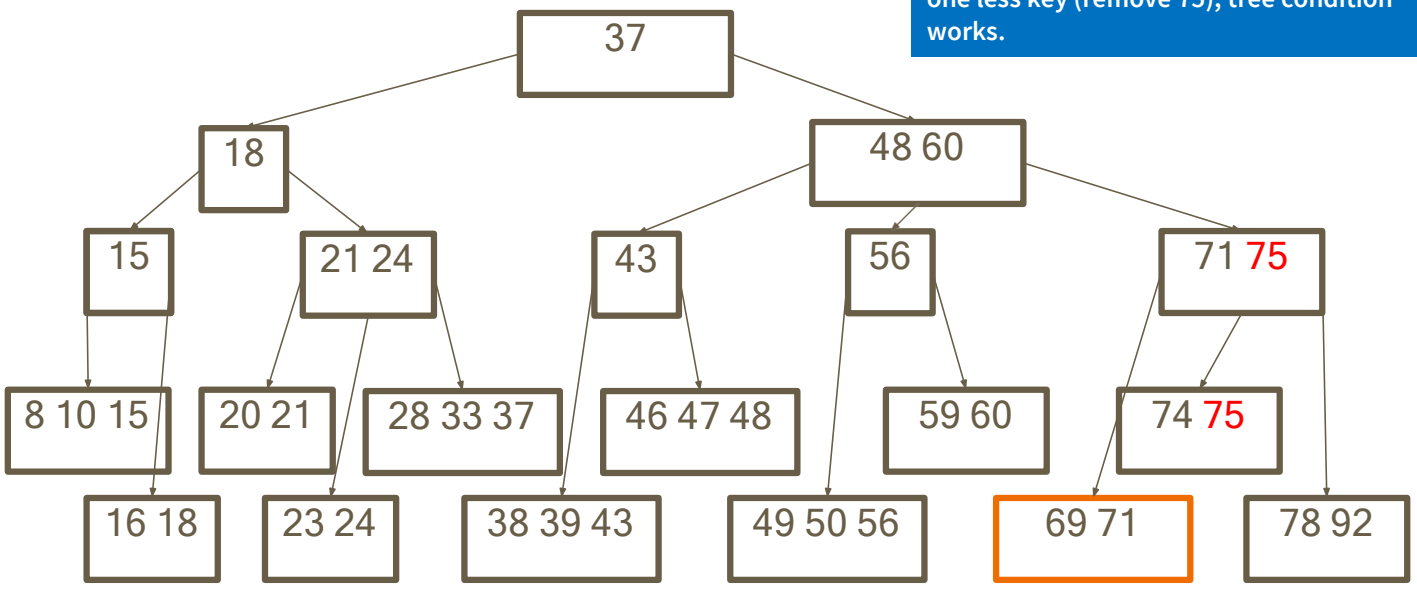
Delete 65 from 10th leaf. It still has 2 or more elements.
No need to update parent as:
1 – No merge was done.
2 – Its value isn't used by any parent



In this convention (max-left), pointer to left of any key is max of the pointed leaf (subtree). Take care while updating parents.
In the lecture we followed (min-right) convention.

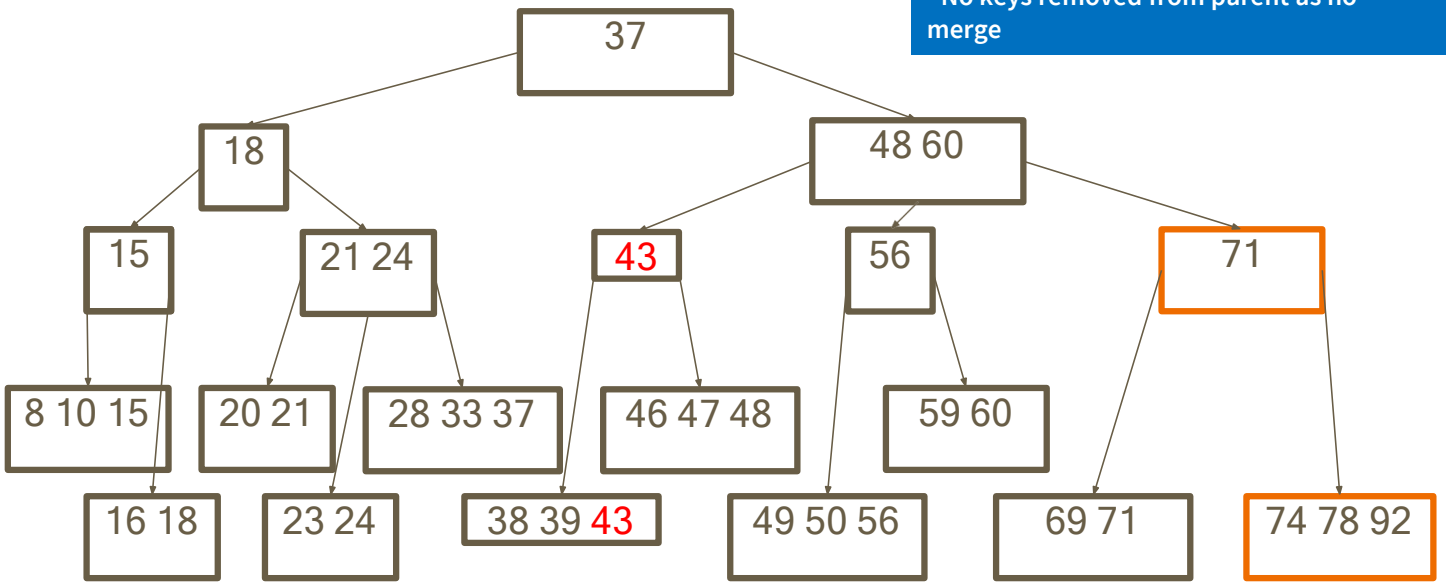
Delete 75

75 is in 11th leaf and both its siblings are poor. Thus, delete and merge with right [74 78 92]
- Two children merged, so parent needs one less key (remove 75), tree condition works.



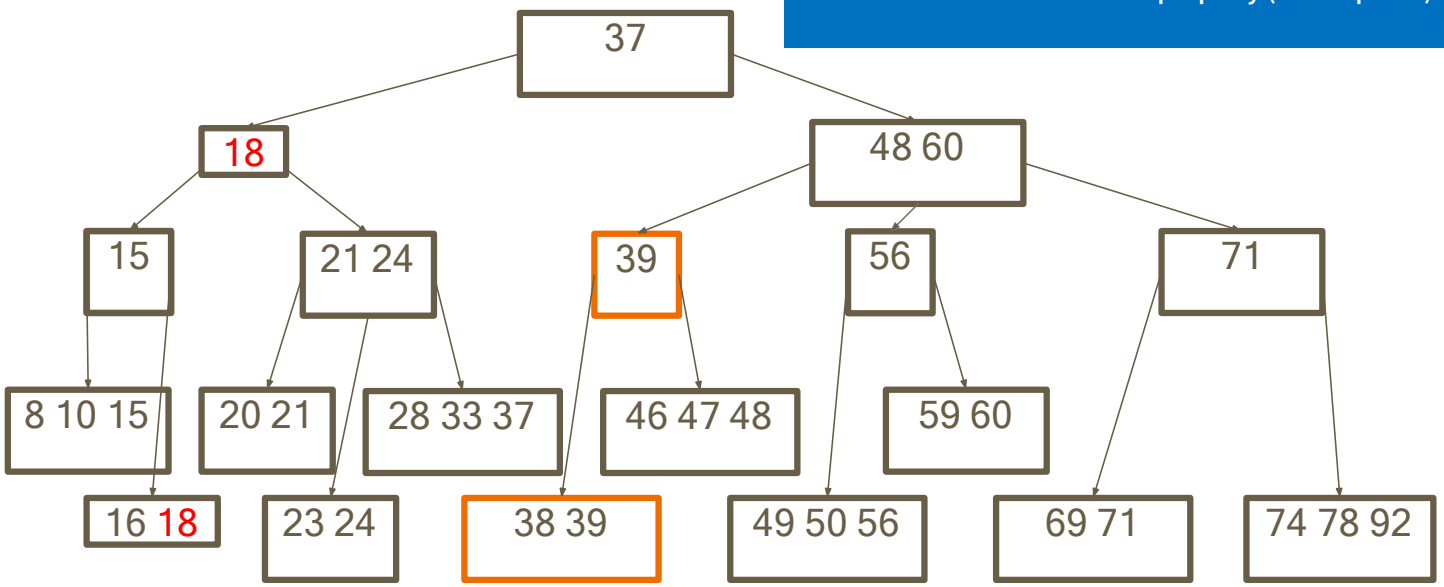
Delete 43

43 is in 6th leaf. Removing it is okay.
Note that max-left property must still hold for each key in parent, so it gets updated as well.
- No keys removed from parent as no merge



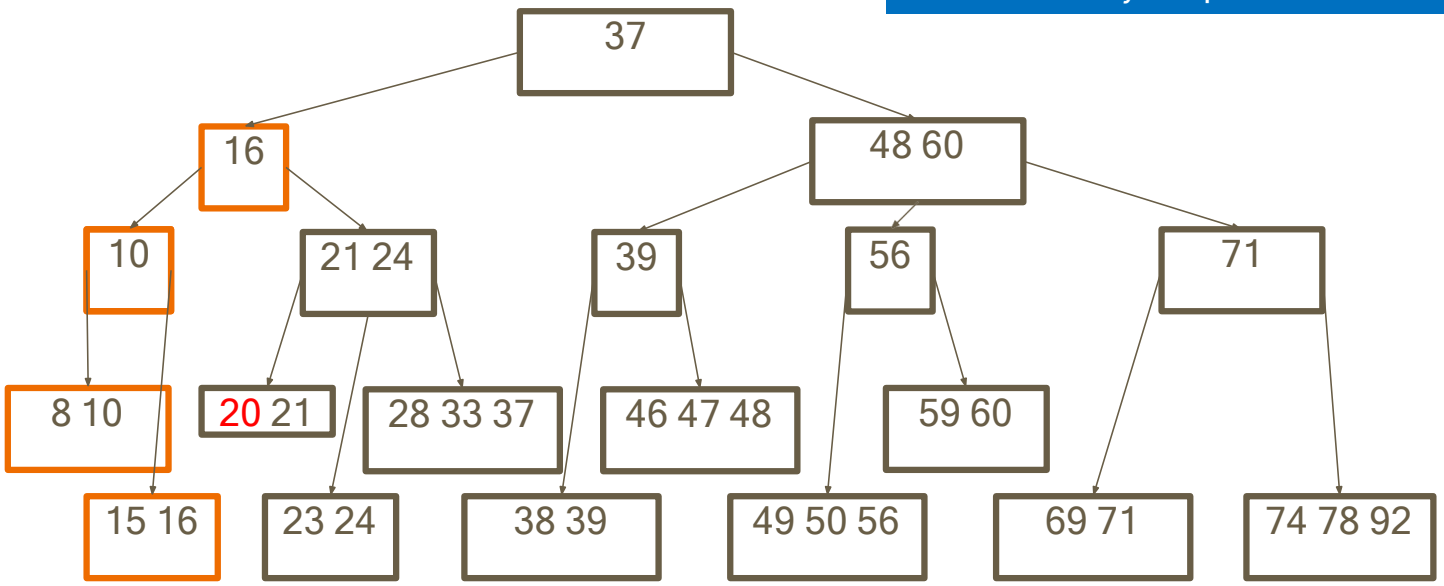
Delete 18

18 is in 2nd leaf.
Left sibling is rich so 15 can be stolen from them.
-No merges so no keys removed form parent
-Parent used 15 for max-left property (need update)
-Ancestor used 18 for max-left property (need update)



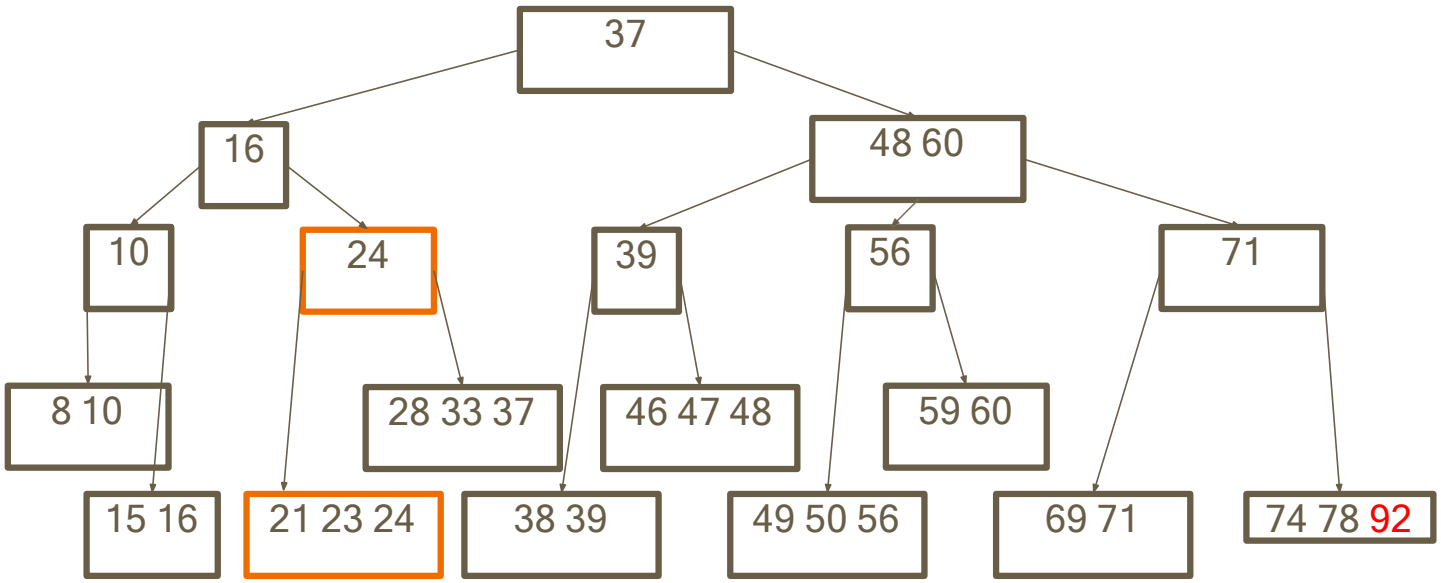
Delete 20

20 is in 3rd leaf.
Both siblings are poor.
Delete 20 and merge with right sibling into[21 23 24]
- Remove the 21 key from parent



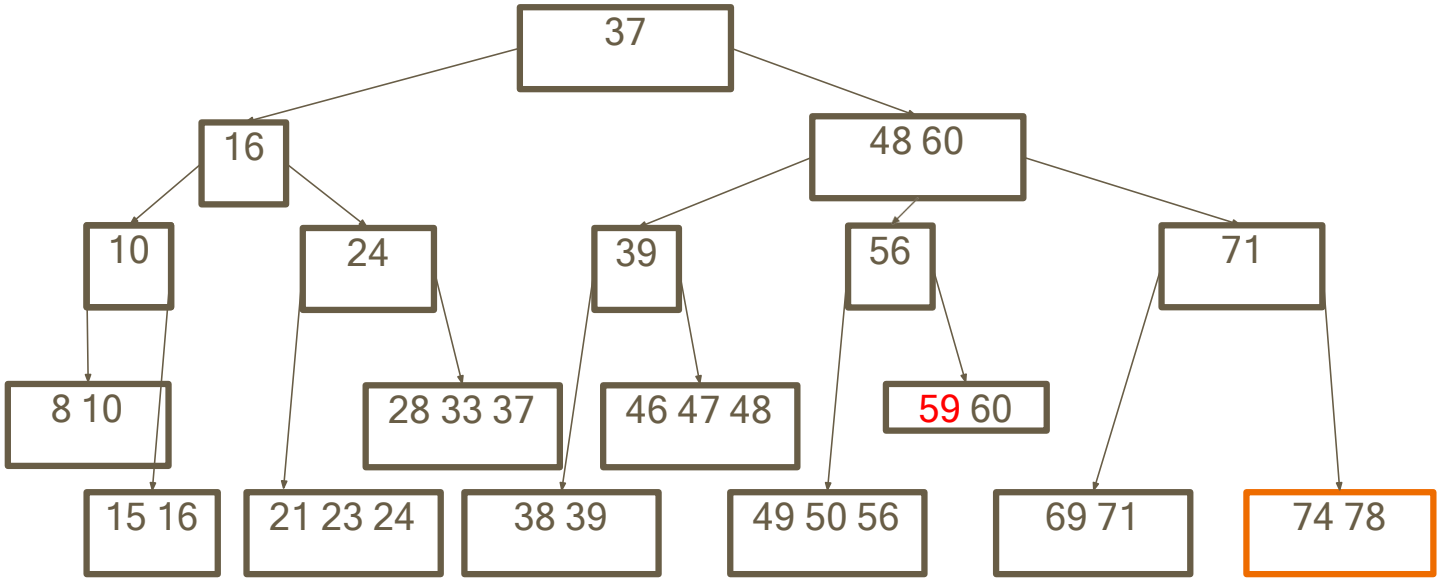
Delete 92

92 is in last leaf.
Can be safely removed (leaf is rich and no parent uses 92)



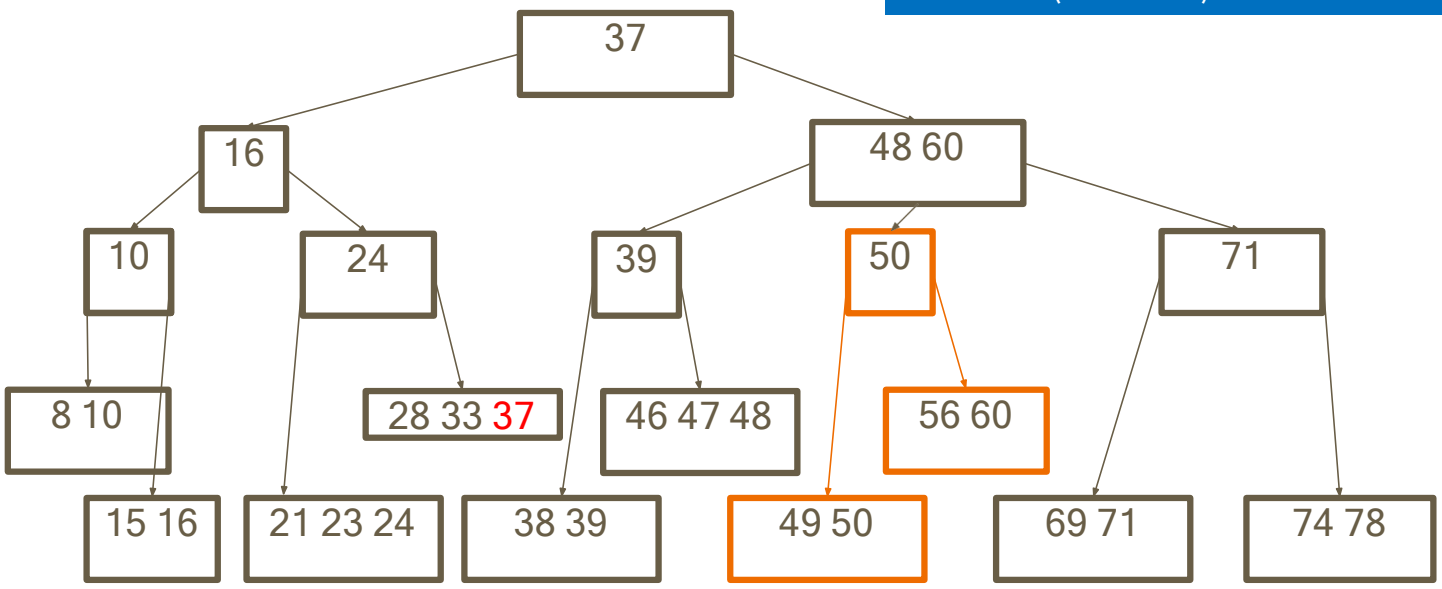
Delete 59

59 is in 8th leaf.
Left sibling is rich so can steal 56 from them.
Parent gets updated to 50 (satisfy max-left)



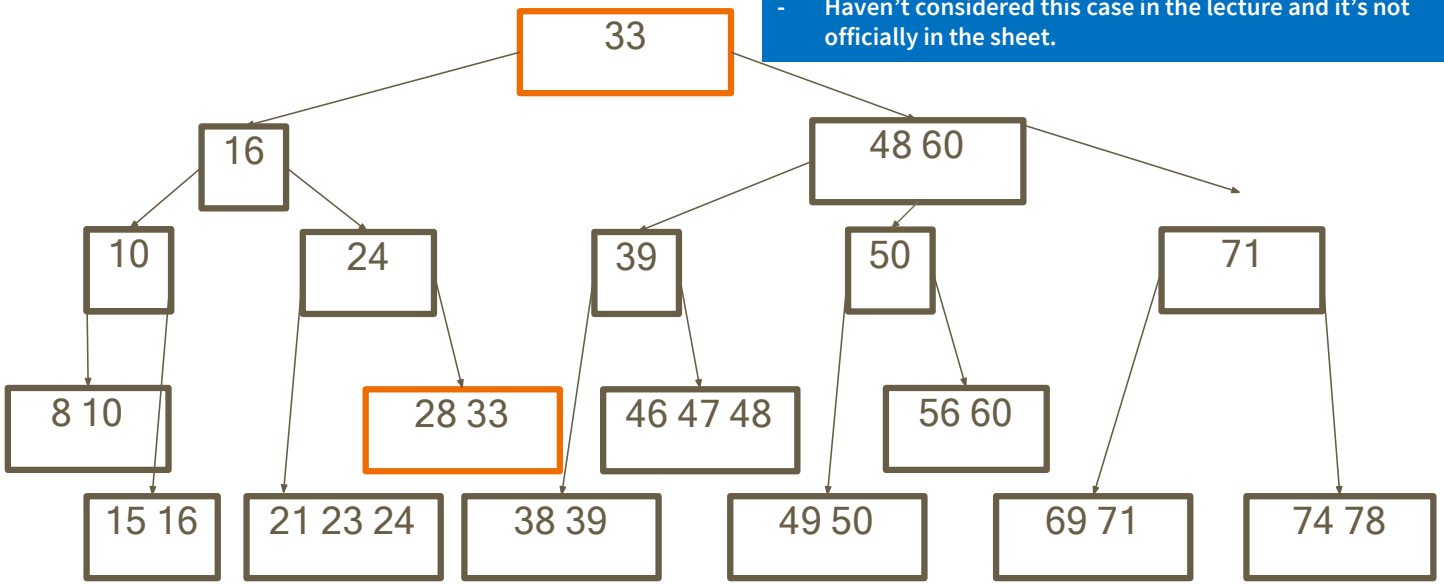
Delete 37

37 is in 4th leaf.
- Leaf is rich
- Value is used in some ancestor (root)
- Remove and update ancestor to satisfy max-left (becomes 33)



Delete 10

10 is in 1st leaf.
- Can't steal from right sibling
- Delete and merge with right sibling
- Remove key from parent
- This will make parent poor and it will need to merge.
- Haven't considered this case in the lecture and it's not officially in the sheet.



Delete 10

-We didn't remove key from parent as mentioned in the slides. (It's a special case.)
-TA slides he put this and further examples to cover all possible cases.
-Have argued that it's not in the lecture slides; should not be possible.
Check TA slides for more.