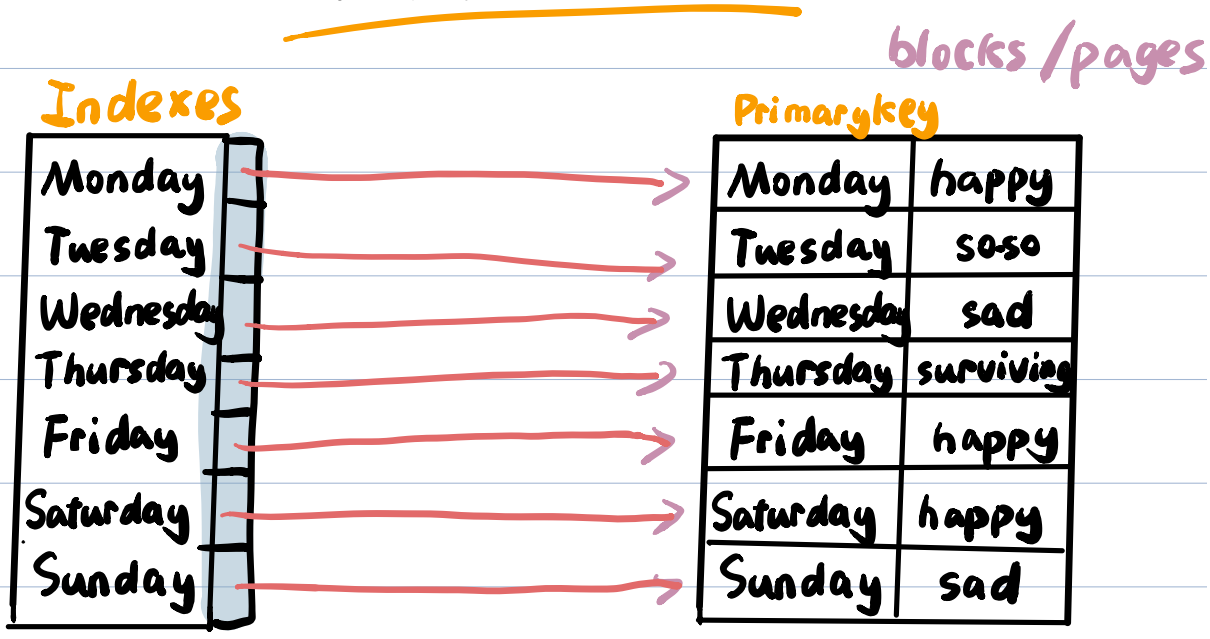
B^+ Tree in DBMS

by Paprika Chen

How do tables stored in the disk?



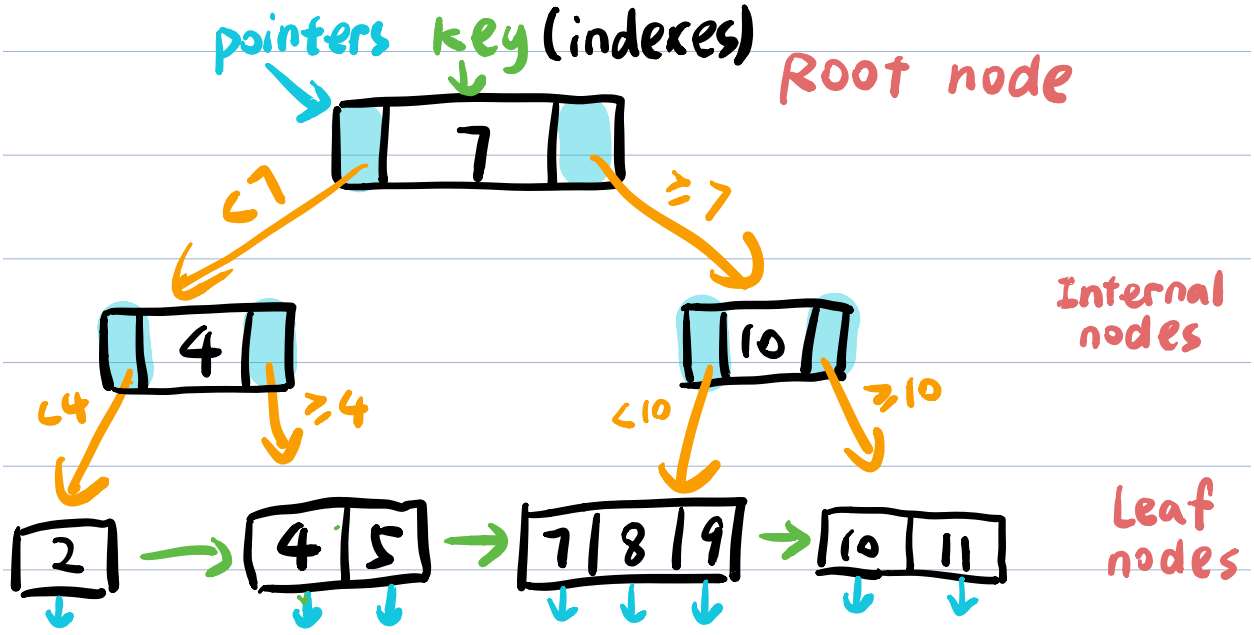
Indexes of Primary keys : Primary indexes are created automatically

⇒ find( ) is still O(n)

How can we store indexes to search efficiently?

B⁺ Tree

- balanced tree
  - all leaves are at the same depth.



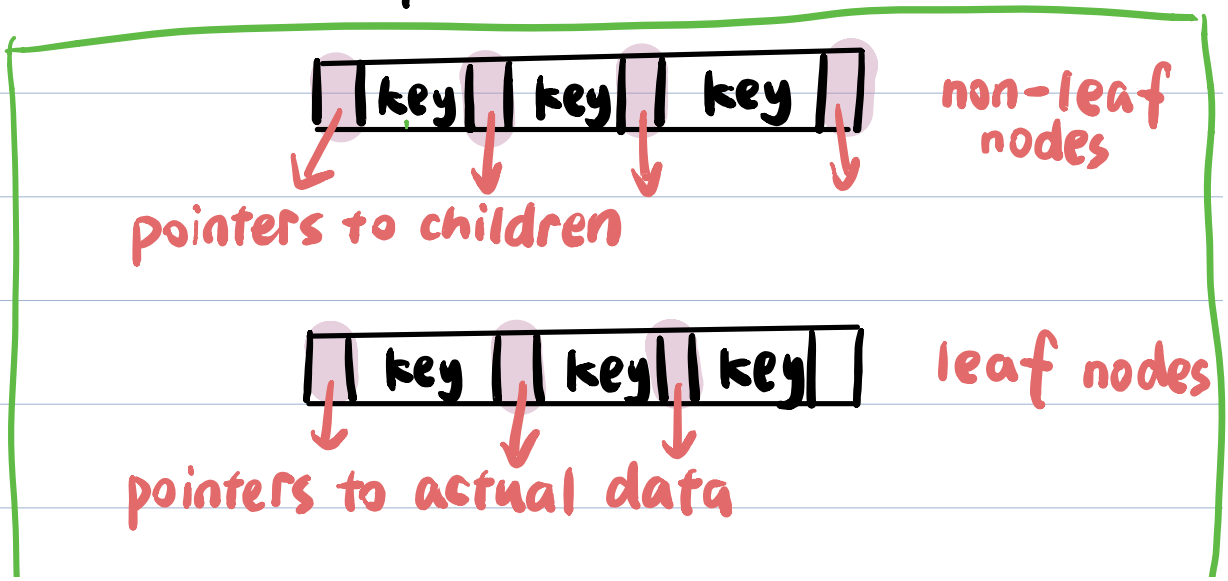
Only Leaf nodes points to the real data!

Other nodes are just route nodes.

① Children on the Left: < parent

Children on the Right: ≥ parent

② n value: A leaf node can have at most n-1 key values. And as few as $\lceil (n-1)/2 \rceil$ values. A non-leaf node is the same, but hold up to n pointers, and at least $\lceil n/2 \rceil$ pointers.



(when n = 4)

Leaf nodes are also linked.

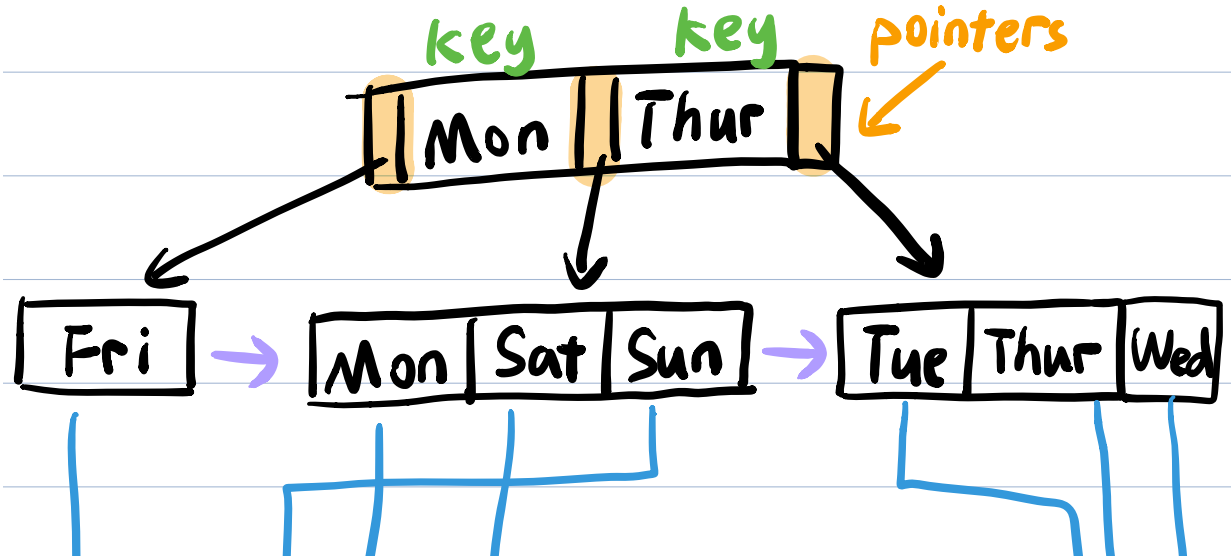
for range search. 4 - 10

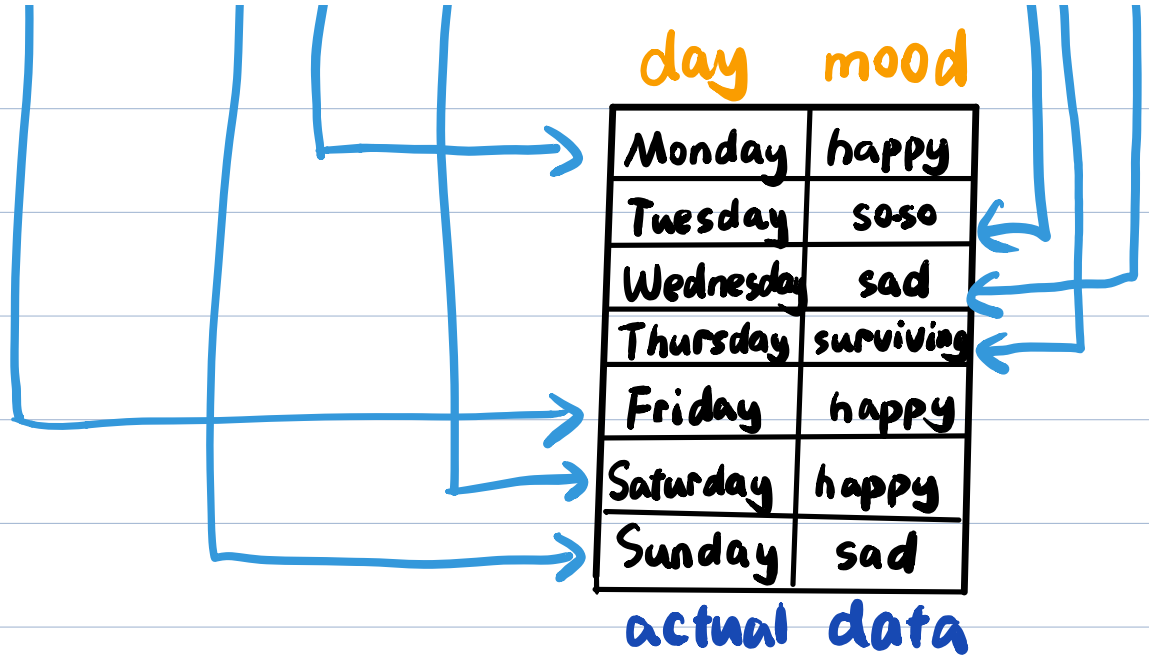
keys don't have to be numbers!

eg.

Alphabetical order:

1. Friday
2. Monday
3. Saturday
4. Sunday
5. Thursday
6. Tuesday
7. Wednesday





Create indexes for day

=> create a B^+ tree where day is the search key

Indexes: the whole B^+ tree structure that contains

pointers and keys

Indexing Strings:

B⁺-tree indices on string-valued attributes: ① variable length

② too long ⇒ less keys in a node

key: names

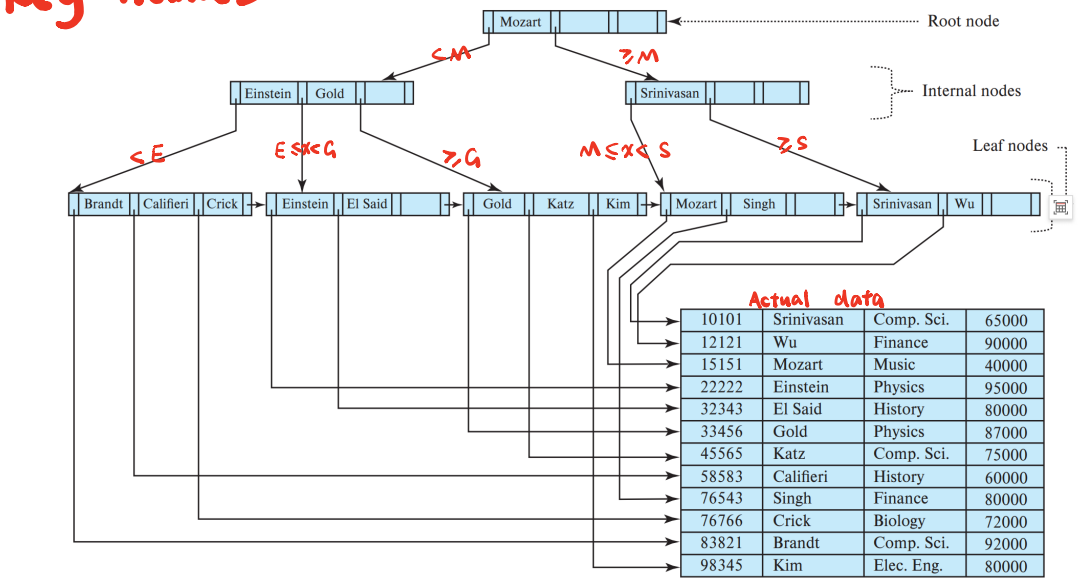


Figure 14.9 B⁺-tree for *instructor* file ($n = 4$).

If we want to find Wu with B⁺ tree: 3 blocks read

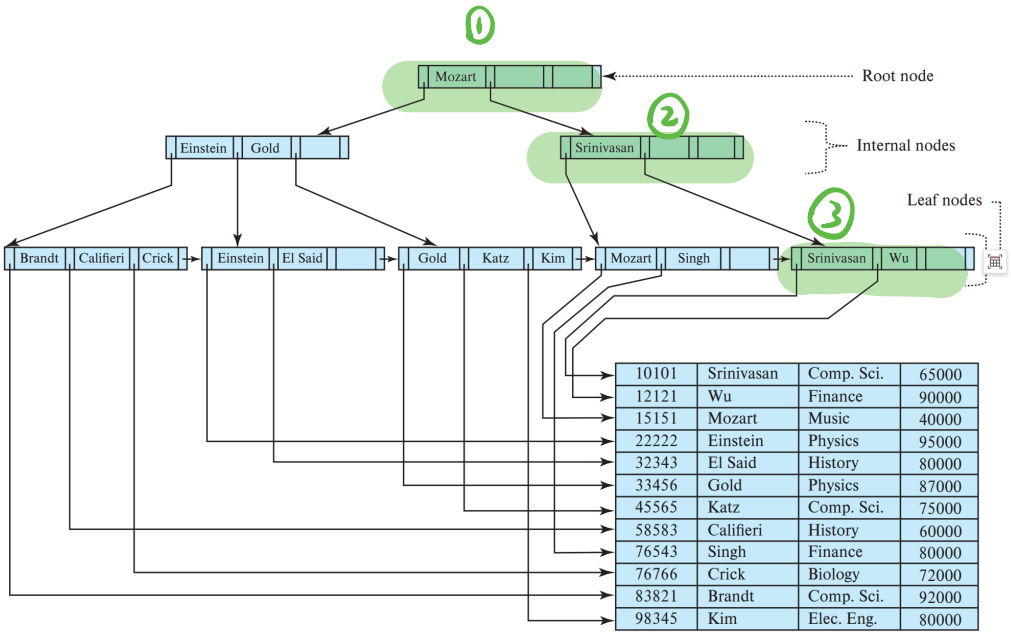
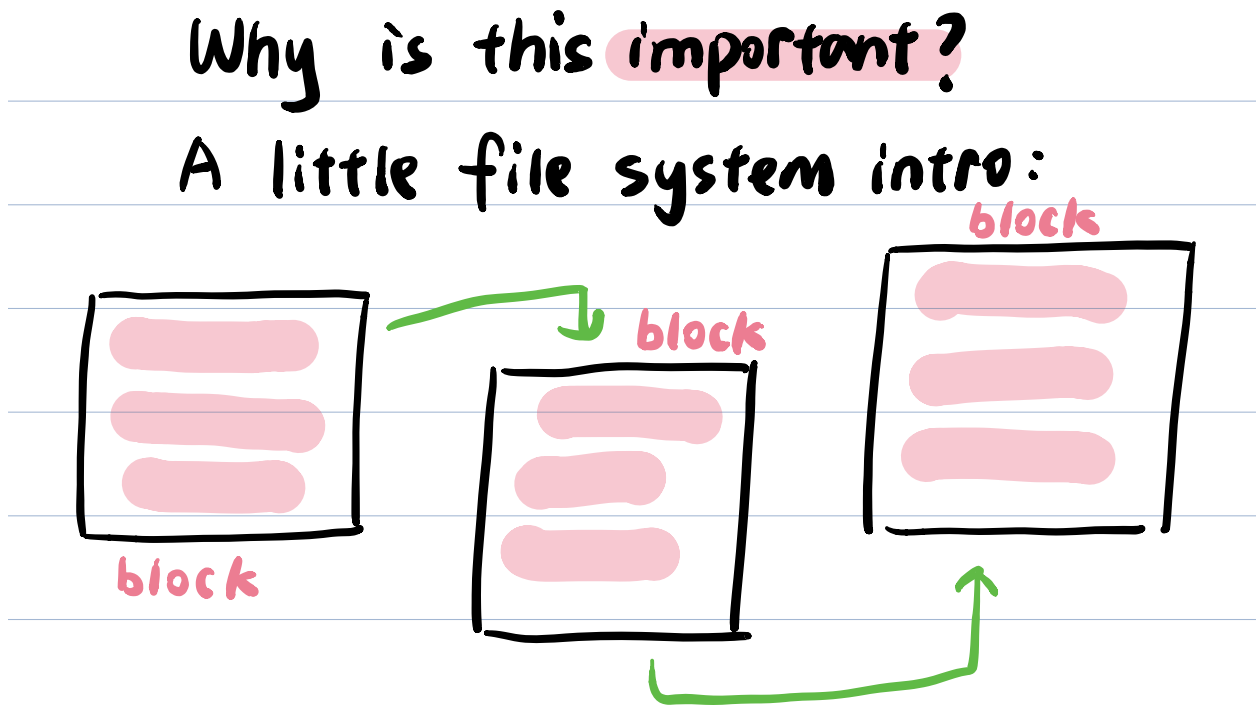


Figure 14.9 B$^+$-tree for *instructor* file ($n = 4$).



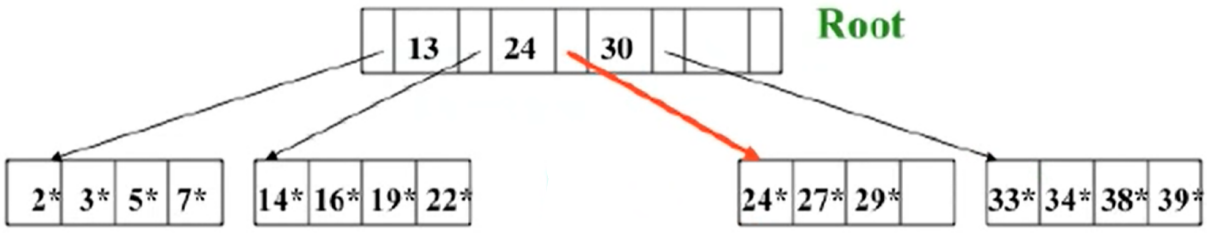
Disk Read is SLOW!

— B^+ Tree update operations

- Insertion

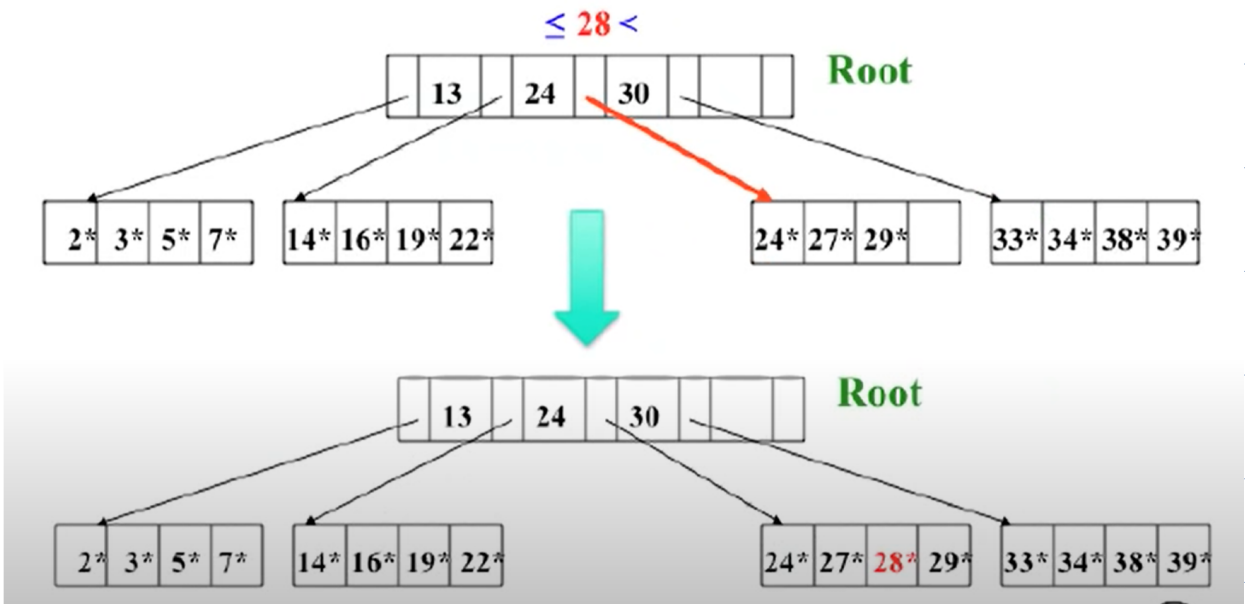
INSERT 28*

* indicates information associated with 28



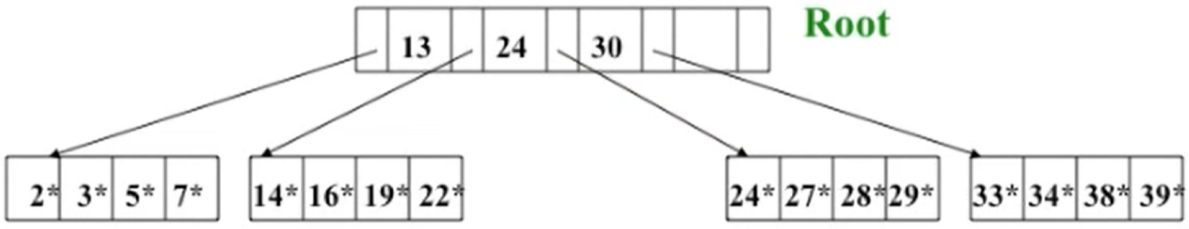
INSERT 28*

*** indicates information associated with 28**

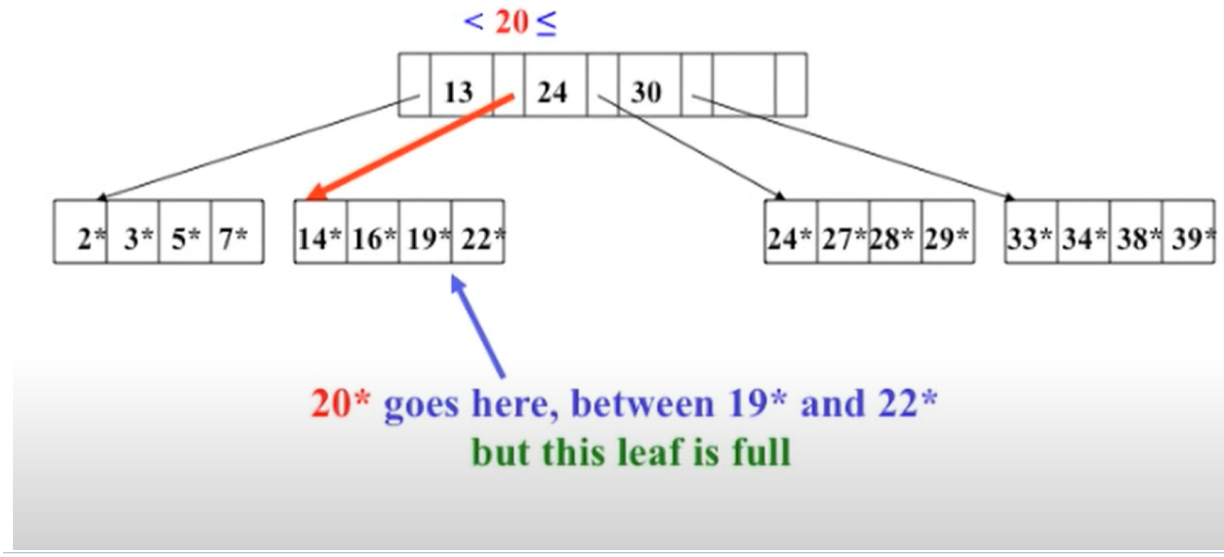


Easy!

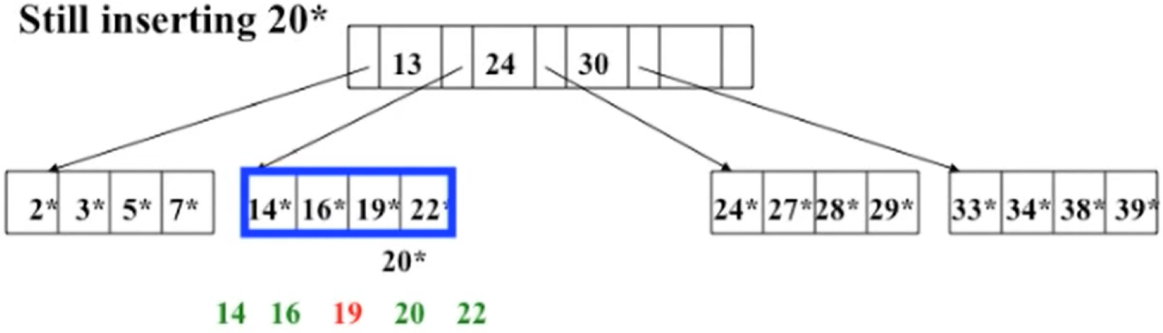
Insert 20* into the following tree



Insert 20* into the following tree

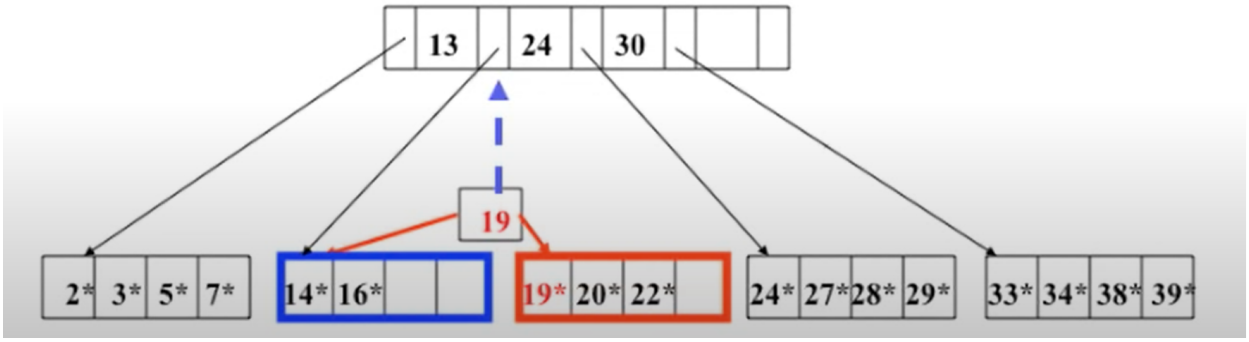


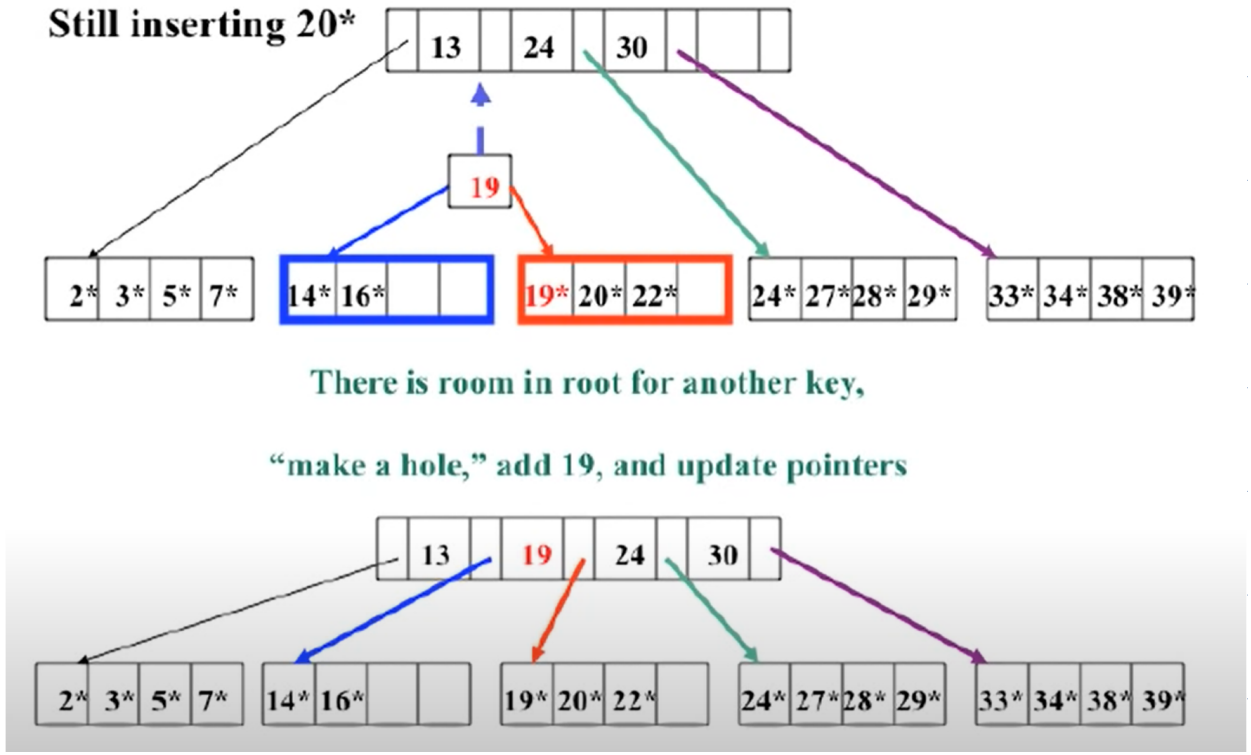
Still inserting 20*



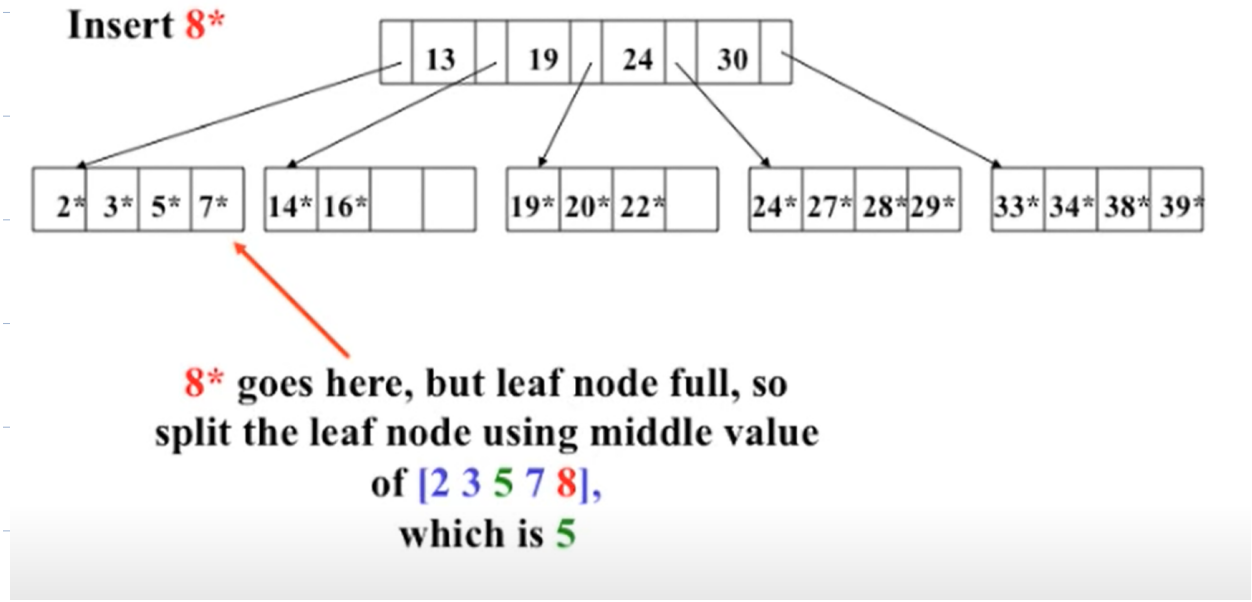
14 16 19 20 22

19 is the middle value

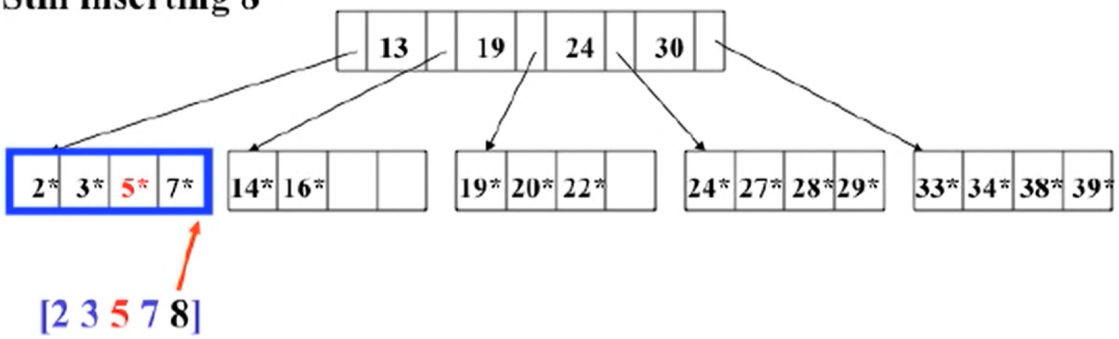




Now let's insert 8!

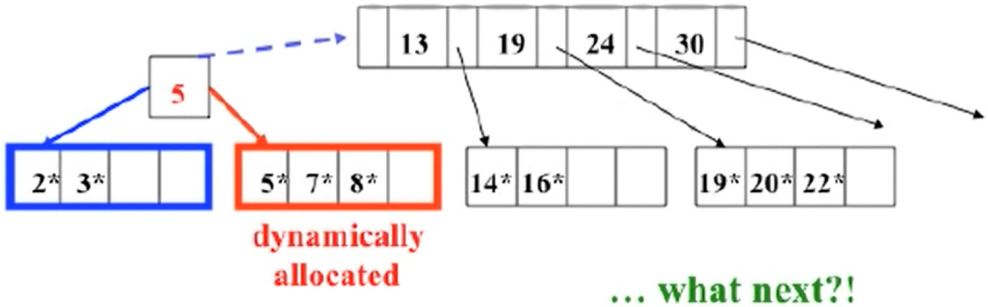


Still inserting 8*



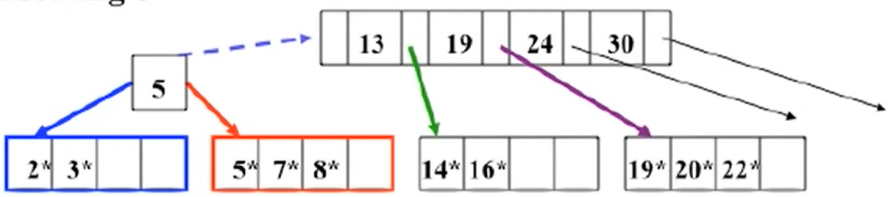
[2 3 5 7 8]

Copy 5 up after splitting leaf, but root is full so cannot include 5



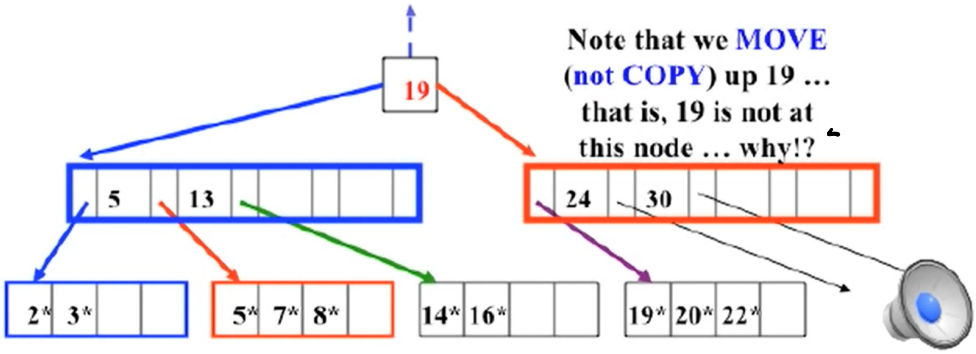
… what next?!

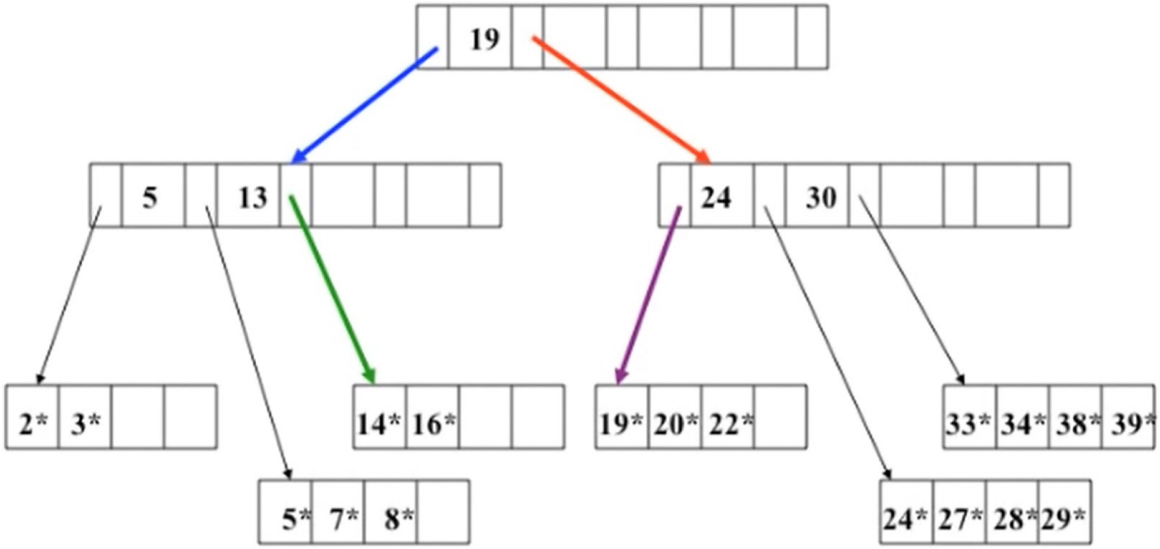
Still inserting 8*



… what next?! … Split the root!

[5 13 19 24 30]



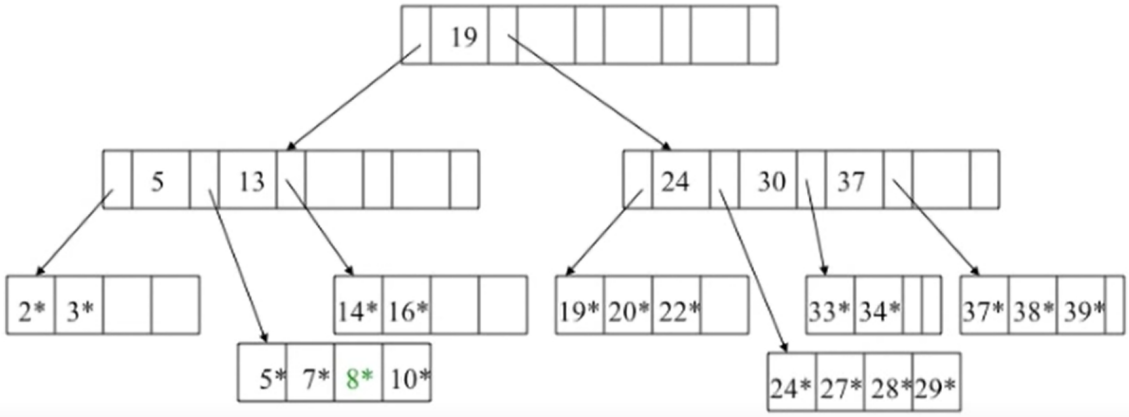


Done inserting 8*

Exercise: ① insert 10

② insert 37

Tree after inserting 10* followed by 37* to previous tree



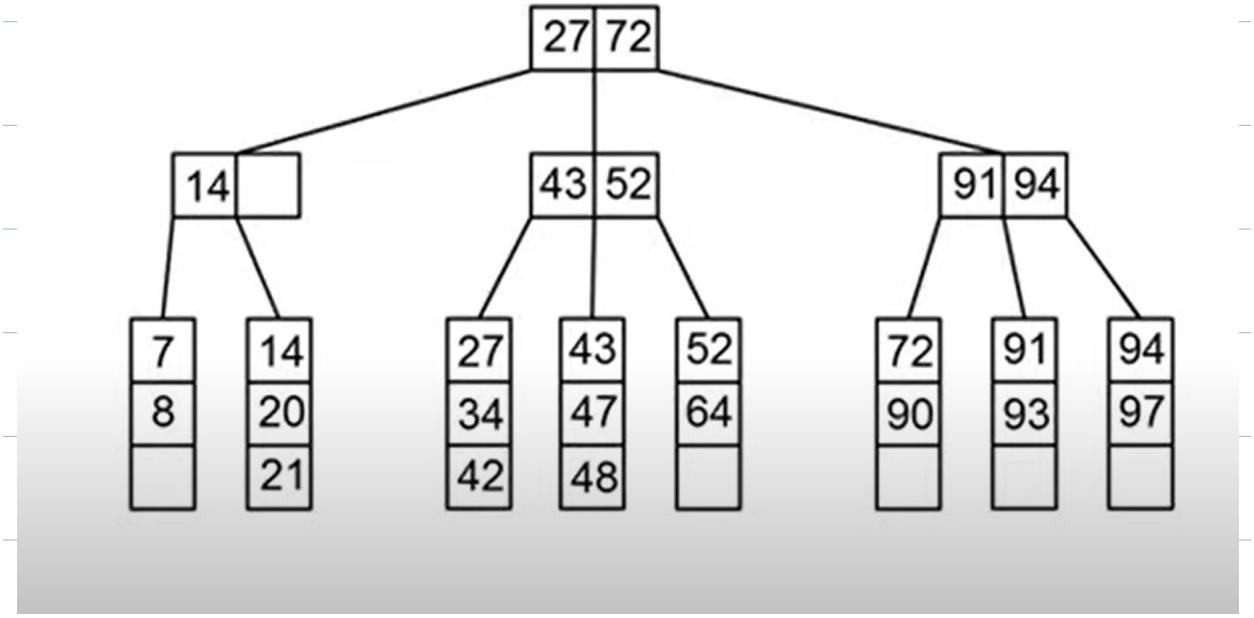
Deletion

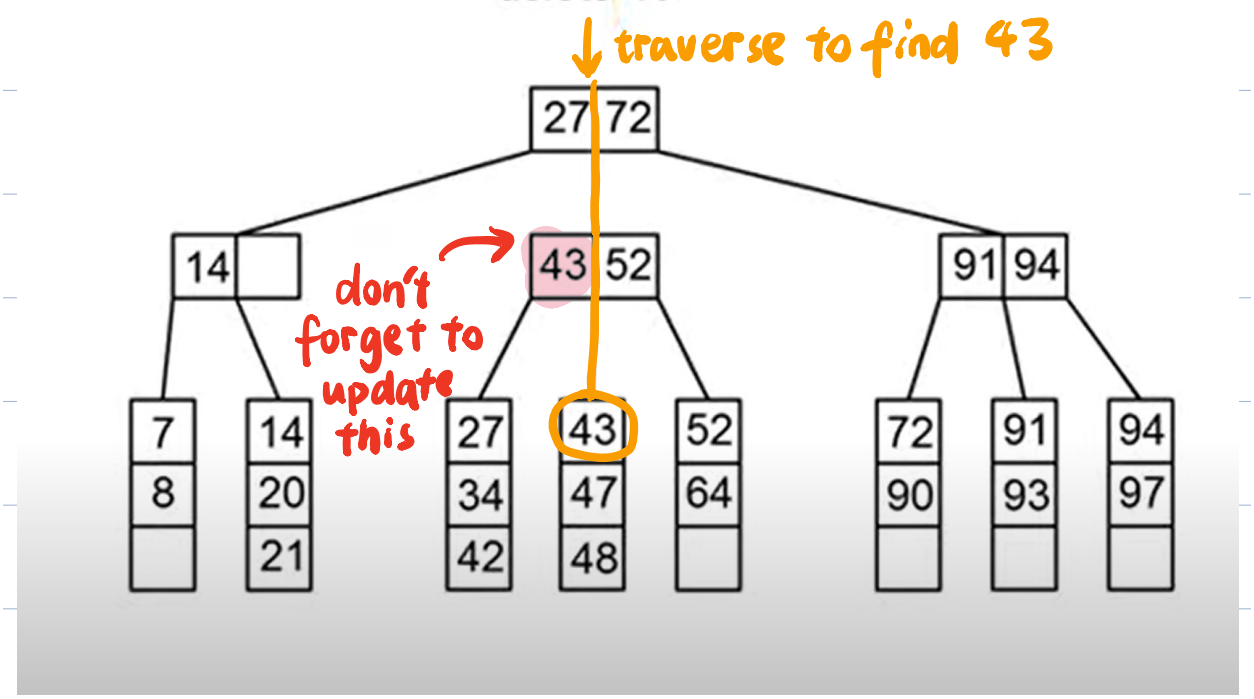
- If after deletion, the number of keys stored in a node $\geq \frac{n}{2}$, easy!
- If not, try the following in order

① Borrow from left
② Merge with left
③ Borrow from right
④ Merge with right

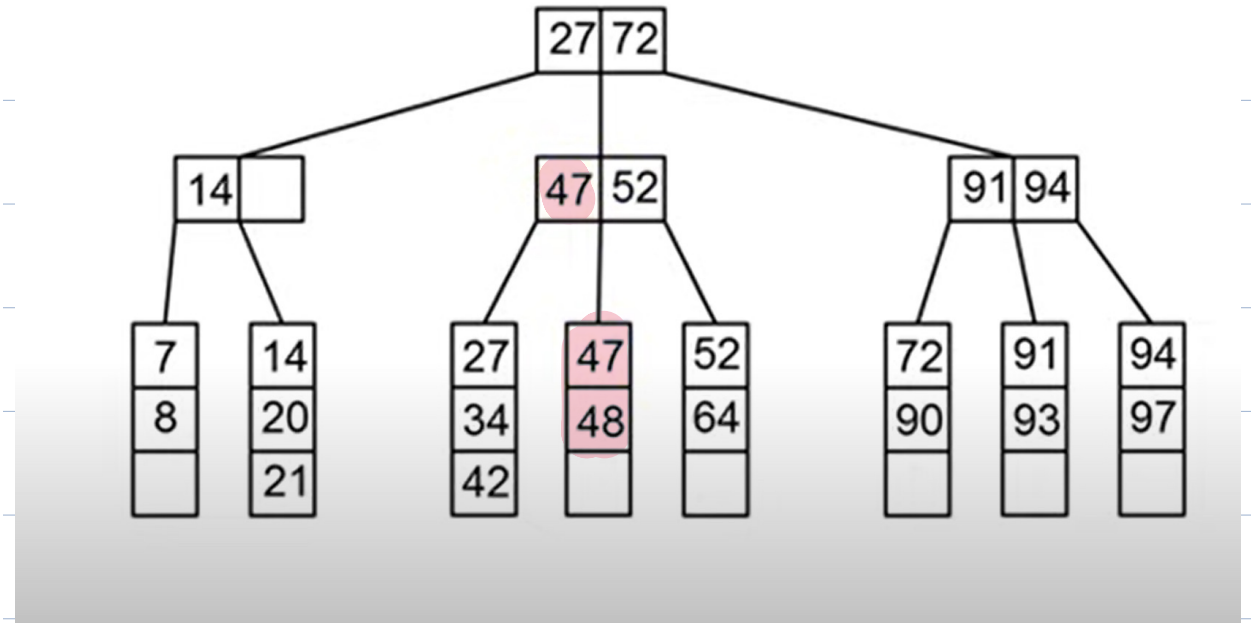
Example:

Delete 43.
$(n = 4)$

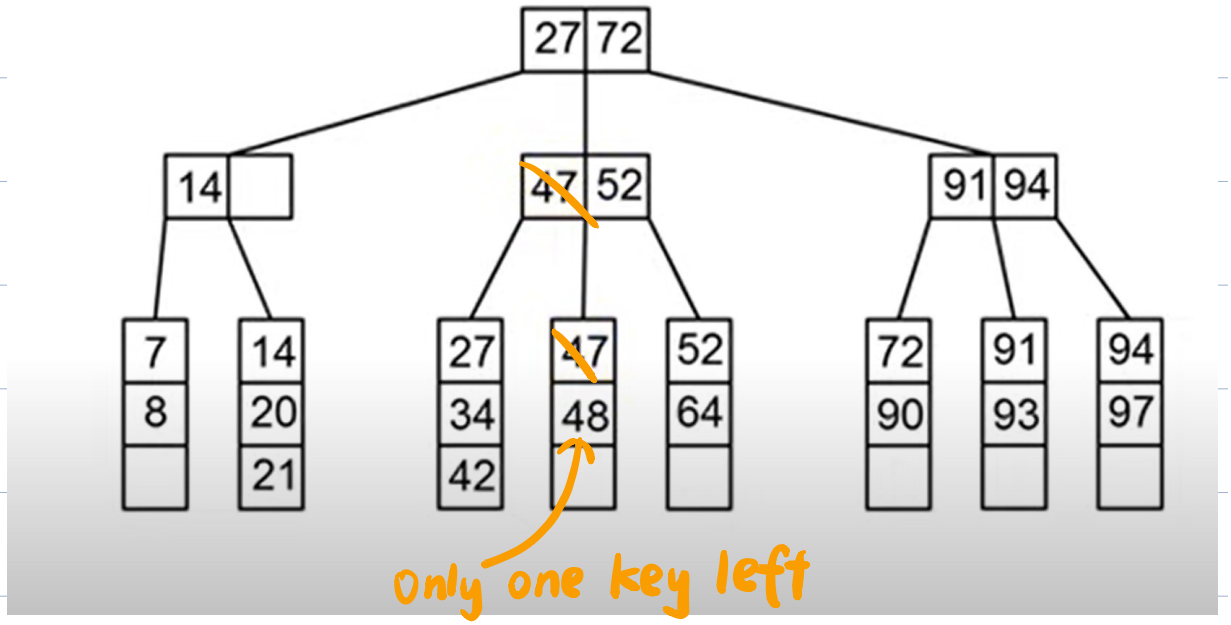

27 72
14
43 52
91 94
7 8
14 20 21
27 34 42
43 47 48
52 64
72 90
91 93
94 97


traverse to find 43
27 72
14
43 52
91 94
don't forget to update this
7 8
14 20 21
27 34 42
43 47 48
52 64
72 90
91 93
94 97

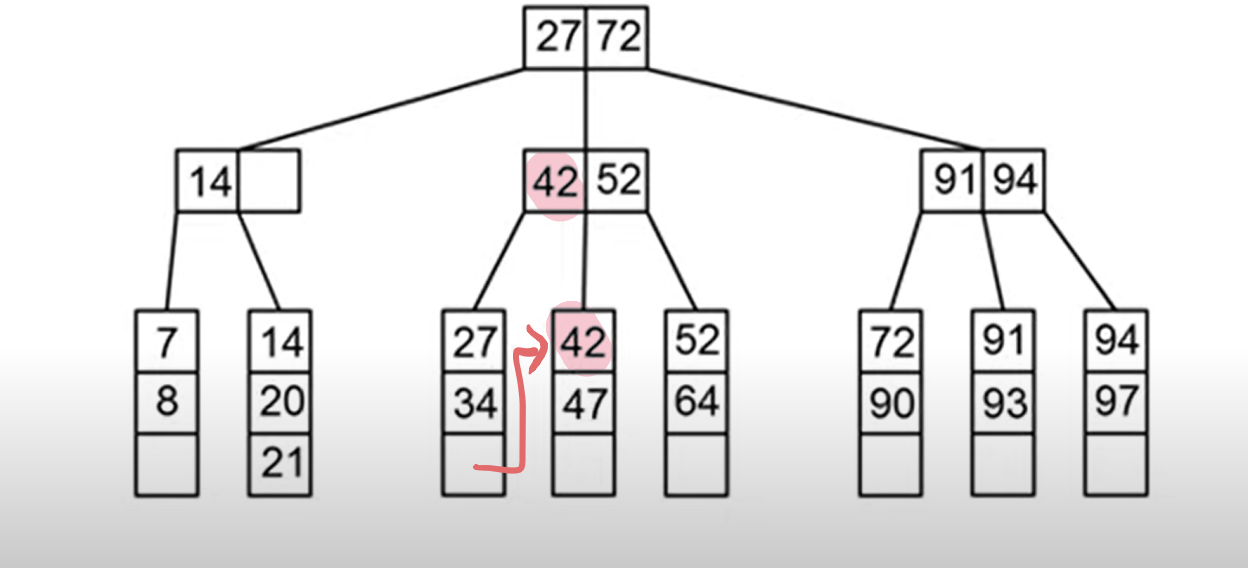
Result:



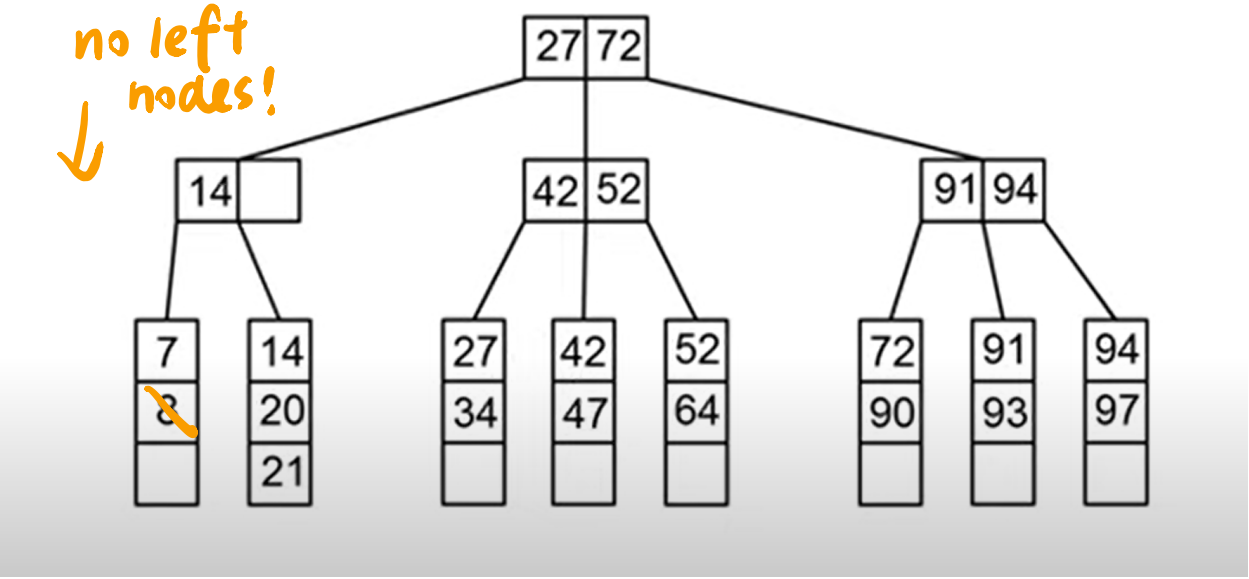
Then let's delete 47.



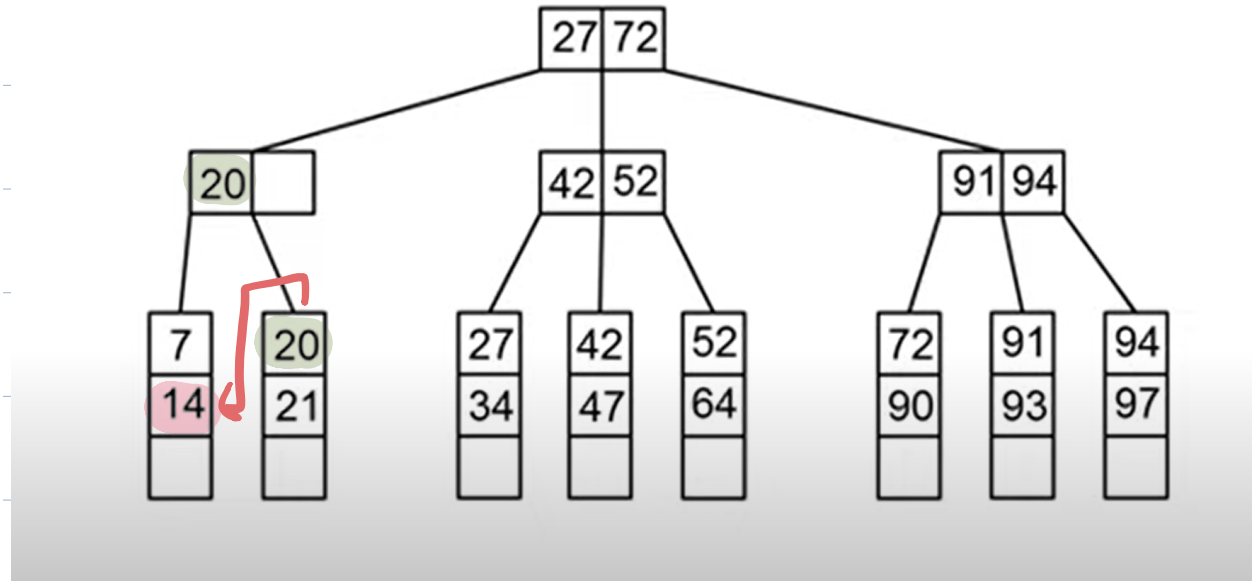
Borrow the largest from left!



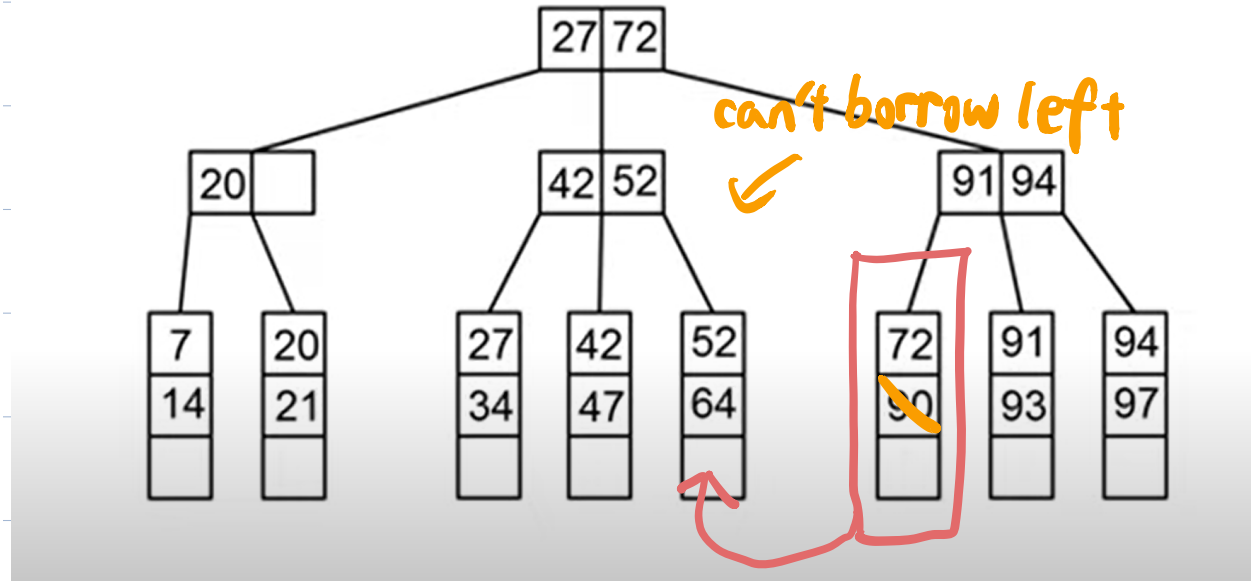
Let's delete 8.



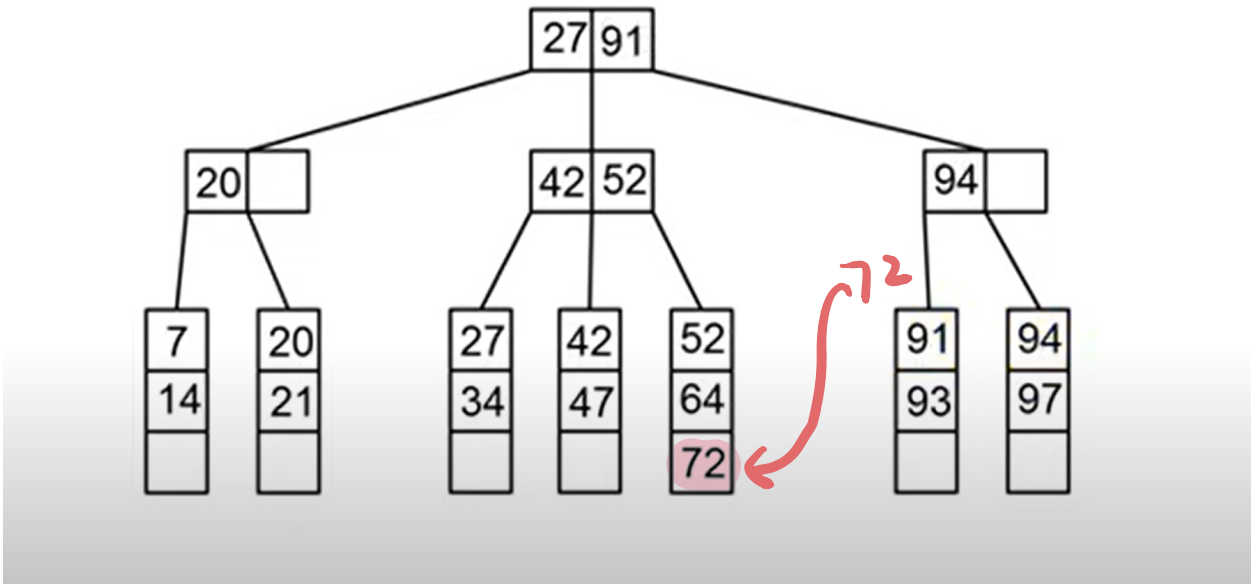
borrow the smallest from right!



Let's delete 90.

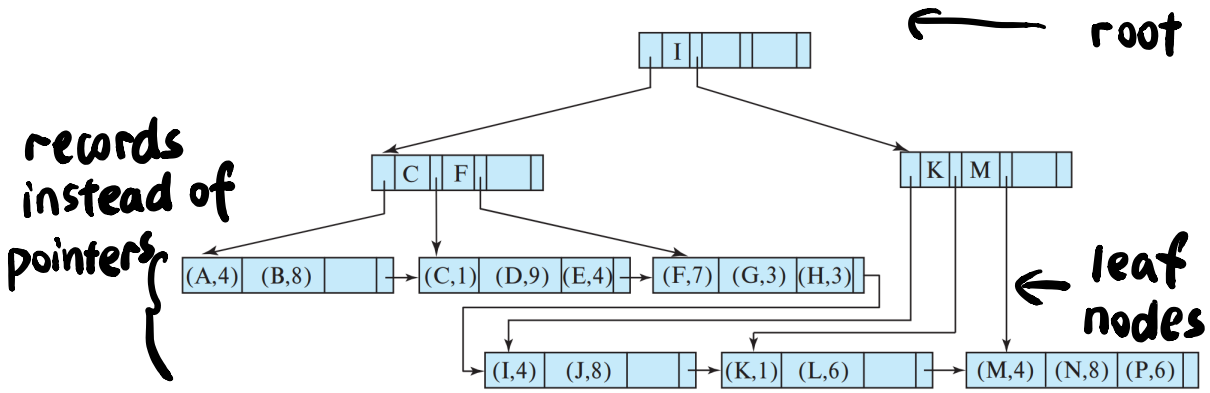


Merge with left node!



B^+ Tree Extensions

- **B^+-Tree File Organization**
  Not only as an **index**
  but also **organizer for records** in file.
  - The leaf nodes store **records**, instead of **pointers to records**.
- **Insertion** and **deletion** works similar to B^+-Tree Indexes.



When a record with key value v is inserted:

① searching the B^+ tree for largest key $\leq v$

② If the block has enough space:

stores the record in the block;

If not enough space:

split the block into two blocks and reassign the records.

Deletion works similar by

① borrowing or ② merge blocks

— B-Tree Index Files

- Similar to B^+ Tree
- But any node can store pointers to data

Parameters	B+ Tree	B Tree
Structure	Separate leaf nodes for data storage and internal nodes for indexing	Nodes store both keys and data values
Leaf Nodes	Leaf nodes form a linked list for efficient range-based queries	Leaf nodes do not form a linked list
Order (n)	Higher order (more keys)	Lower order (fewer keys)
Key Duplication	Typically allows key duplication in leaf nodes	Usually does not allow key duplication
Disk Access	Better disk access due to sequential reads in a linked list structure	More disk I/O due to non-sequential reads in internal nodes
Applications	Database systems, file systems, where range queries are common	In-memory data structures, databases, general-purpose use
Performance	Better performance for range queries and bulk data retrieval	Balanced performance for search, insert, and delete operations
Memory Usage	Requires more memory for internal nodes	Requires less memory as keys and values are stored in the same node

Most database-systems use B^+ tree but call it B-tree.

- Hash Indices (in main memory)
  1. Hash maps are BAD on disk.
  2. RAM are expensive and small to keep indexes.
  3. No range queries.

references

Youtube @Douglas Fisher

Youtube @Stephan Burroughs

Youtube @Jordan has no life

geeksforgeeks introduction-of-b-tree