17/4/25

DBMS PROJECT

M PRASANNA KUMAR
CS23B1011

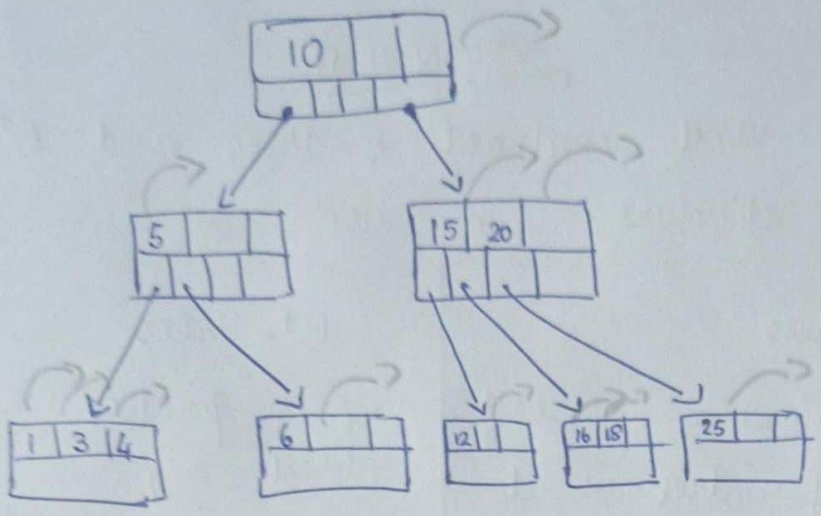
ASSIGNMENT

1) Compare and contrast B-trees and B^+ trees Explain structure and use cases.

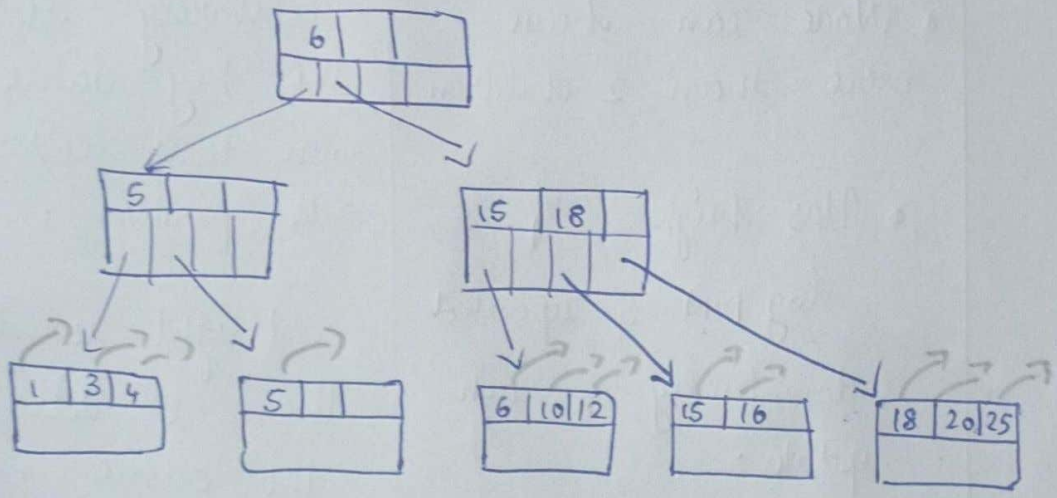
B-Trees	B^+ Trees
• Also known as self balancing tree as its nodes are sorted in the inorder traversal. All nodes have data pointers	• Data pointers are stored only at the leaf nodes of the tree.
• Node can have more than 2 children	• Leaf nodes must necessarily store all the key values along with their corresponding data pointers.
• Has height of $\log_M N$ & updated automatically on each update: where M : order of tree N : number of nodes	• Height is lesser than B trees for the same number of nodes.
• Data is sorted in specific order with low value on left and highest value on the right.	• Duplicate of keys are maintained and all nodes are present at the leaf.

Structure: M=4

B TREES



B+ Trees



Use cases:

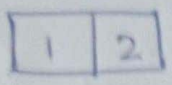
B- Trees	B+ Trees
1) Databases	1) Databases
2) Search engines	2) Geospatial databases
3) File Systems	3) Range Queries
4) Tele-Networks	4) Disk Storage

CS23B1011

2) Explain insertion operation in B tree with an example. How does it maintain balance?

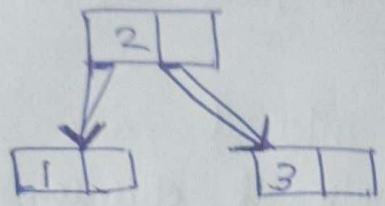
Insert 1 2 3 4 5 6 7 and lets take order of B-Tree as 3

Insert 1, 2



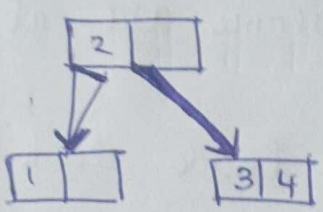
Create root node and insert 1, 2

Insert 3



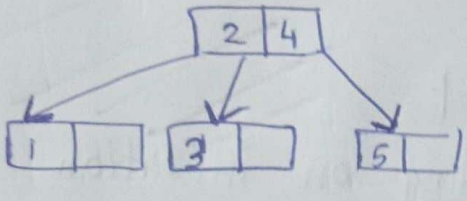
As we have no space for insertion of 3, nodes are in overflow, we pick 2 and create 2 child nodes.

Insert 4



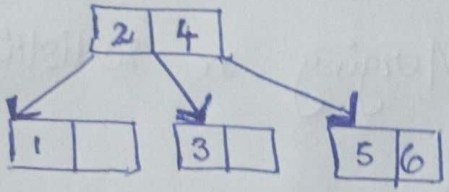
Insert 4 in the space

Insert 5.



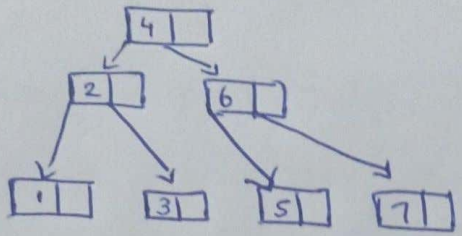
We choose 4 as the median and split the node containing 3 and 4.

Insert 6



Insert 6 after 5

Insert 7



Lastly we have 7, we have to split node containing 5,6 Again split parent node (2,4) & choose 4 as median.

B tree maintains balance through structural Rules and ensure tree remain shallow

B tree of order (m) is self Balancing search tree where

* each node can have atmost (m) children
* each node (except root) must have atleast $\lceil m/2 \rceil$ children
* each internal node with k children has (k-1) keys
* All leaves are at same level
* Keys inside each node are Sorted and guide the search

On each step

1) Splitting on insertion
2) Merging or redistribution on deletion

3) [CS23B1011]

Describe deletion process in B+ tree. What challenges are faced and how are they resolved?

S_1: Look to locate deleted key in the leaf nodes.

S_2: Delete the key and it's associated value if key is discovered in leaf node

On deletion if number of keys is less than half the maximum allowed

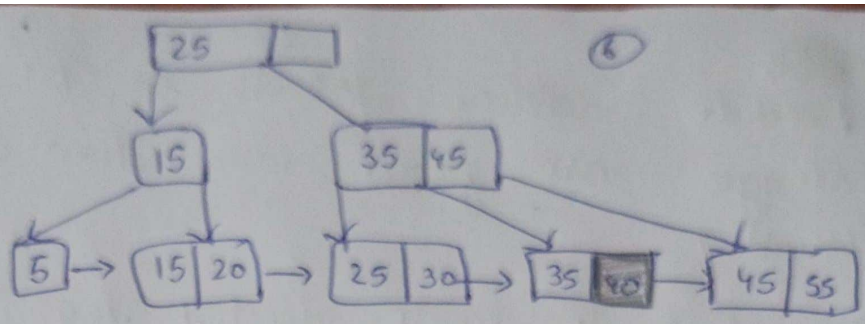
1) Get a key by borrowing it from a sibling node if it contains more keys than required minimum.

2) If minimal number of keys is not by all of the sibling nodes, merge the underflow node with one of its siblings and modify parent node as necessary.

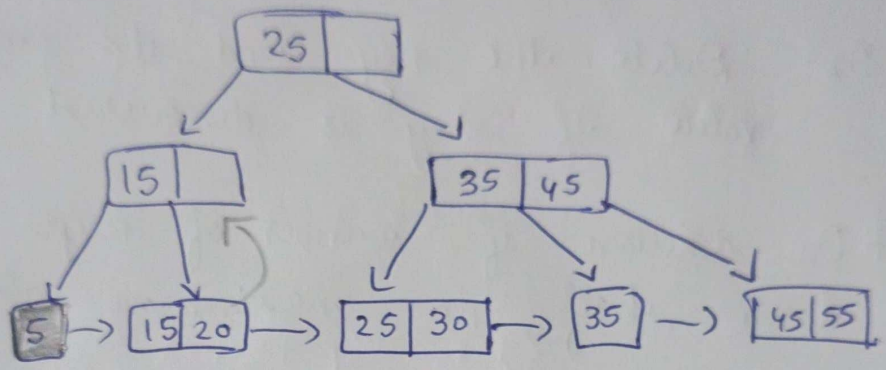
S_3: Remove all references to the deleted leaf node from the internal nodes of the tree.

S_4: Remove the old root node and update the new one if the root node is empty.

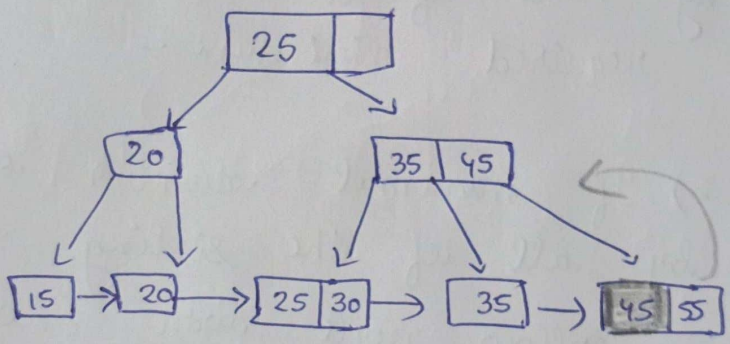
Let (m=3)



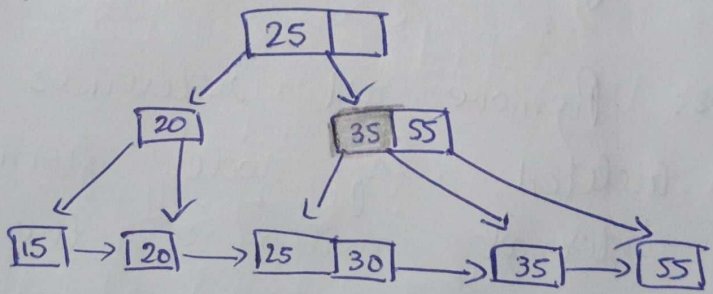
Delete 40 :



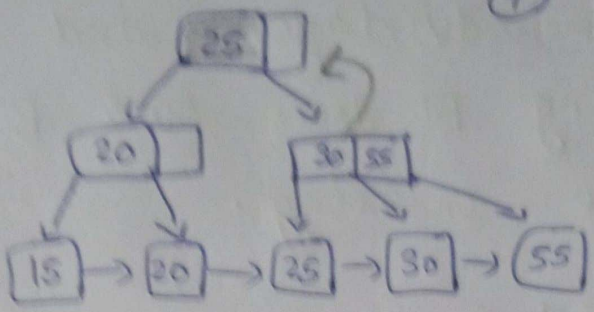
Delete 5 :



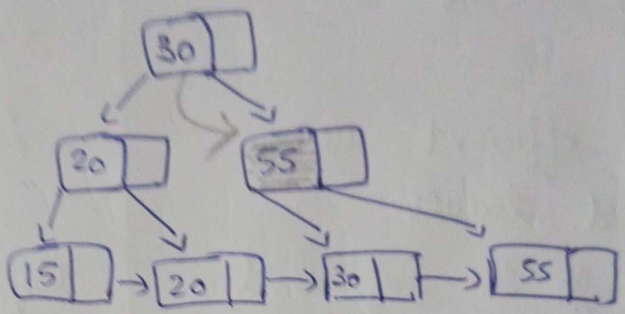
Delete 45:



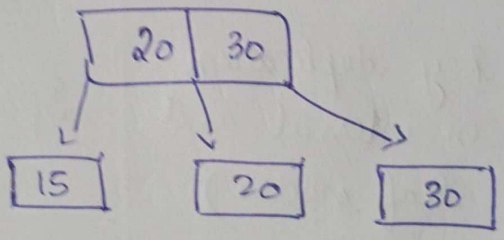
Delete 25 :



Delete 25:-



Delete 55:-



challenges resolved :-

1) Borrowing from Siblings
2) Merging leaf nodes
3) Fixing internal nodes
4) Update leaf links

4) Describe advantages and dis-advantages of using B+ trees over B trees in database indexing. ⑧ LCS23B1911

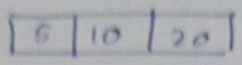
	B+ Tree	B Tree
DATA STORAGE	Only in leaf nodes	Both internal and leaf nodes
RANGE QUERIES	Very efficient using linked leaves	Less efficient - Scattered data
POINT QUERIES	Must go to leaf level always	May find key in internal nodes earlier
TREE HEIGHT	Shorter	Taller
MEMORY USAGE	Higher (key duplication in internal and leaf nodes)	Space efficient
DISK Performance	Better due to fewer levels	Slightly worse
Con-currency	Easier to implement with leaf level locking	More complex
Complexity	More complex in deletion & updation	Simpler during deletion and updation
Use Case	Preffered in large databases (MYSQL, POSTGRESQL)	Suitable for in-memory data or small scale apps.

6) Construct B+ trees of order (m = 4) ⑨ CS23B1011 Prasanna Kumar

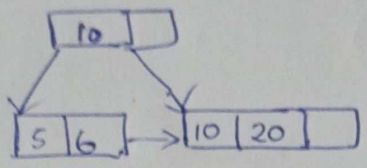
10, 20, 5, 6, 12, 30, 7, 17

Solution:

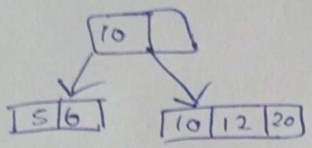
Insert 10, 20, 5



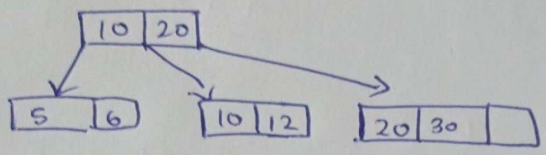
Insert 6



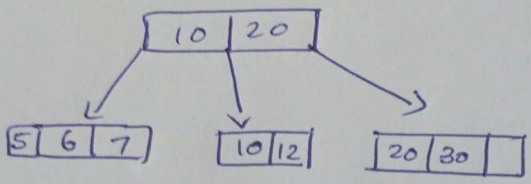
Insert 12:



Insert 30:



Insert 7:



Insert 17