and no. of blocks needed to store D.B file = $\frac{256}{8} = 32$.

Now assume those 256 records belong to students table with R.NO as a col.

and we are running the following query:-

```
Select * from Students
where R.NO. > 5 and R.NO. < 50
```

→ So as of now :- the query processor don't know exactly that which record is present on which block? ⟹ So it will go through all the blocks. and have to spend a lot of time.

So here the file should be stored in an organised way so that time for searching should be min^m and the performance should be improved.

↳ The storage of the DB file should follow some pattern and it should be accessible directly otherwise we have to search for a single record linearly :- linear search.

* we can have records of :- fixed length or variable length.

These are v. easy to maintain

not easy to maintain

little difficult technique.

saves a lot of space.

28/09/2022 #lesson 09#

→ deletion in fixed length records :-

→ consider we have a students records stored in sorted order of R.No. In a table.

Now if one record has to be deleted then :-

So we have following two options :-

1. Keep remaining records in sorted order :-
   - ↳ shift remaining records up
     - ↳ this is costly ! (time consuming)
   - another thing is :- Just keep those empty places as it is !
     - ↳ but deletion happens very frequently = then it will be problematic.

2. Unordered Roll no.
   - ↳ here the table will be converted into the linked list form.
     - ↳ one column will be added at the end which will give us the next free record of the table

last column :- will be "Pointer to next free address"

#Now Why Indexing :- here the records of a DB table entered in disk blocks don't have their block numbering = that in which block they are stored. So it is very difficult to search for few particular records :- based on some criteria. the problem is that you have to search linearly and you'll have to go through all the blocks where that DB table will be stored.

So that's why we need Indexing!

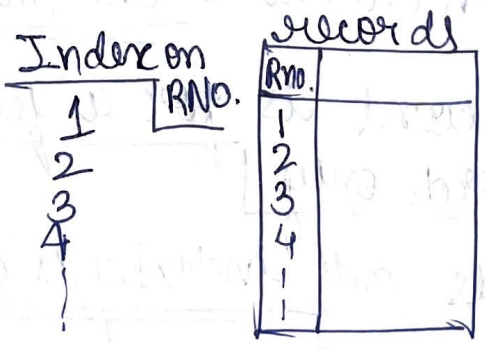
#Index file :- In Secondry memory (disk) dataBase file is stored and to maintain the indexes :- another index file is created :- stored in single/multiple blocks.

→ where index of a table stored.

So index file will have :- the specific records of the table is stored in which blocks.
↳ their addresses.

#techniques :- 01. Clustered Indexing :-
data order and Index order are same to same.

means :-



indexing of roll no. & records stored on basis of roll no. are in same order exactly.

02. Non clustered Indexing :-

Here data order and Index order are not same :-

Indexing on R-No.

RNo.	diskno.
3	
4	
2	
5	
1	

records

RNo.	
1	
2	
3	
4	
5	

orders of both are very different.

both sequences don't match here

→ So these both ~~seq~~ Indexing techniques show the relation b/w record sequence and Index sequence.

Dense V/S sparse Index :-

→ Dense :- Indexing is done for each and every database record (table row).

advantage = we can reach to a specific record in very less time, easily

→ for each row of a table :- there is Indexing done.

Ex: we have 2000 records in Student table then in Indexing also we'll have 2000 records

↳ for each roll no. = one Index we'll have.

→ Sparse :- Index record is for a few DB records only.

↳ for few records only Indexing is done

Ex: for 2000 records in student's table we have only $\frac{2000}{5}$ = 400 indices

↳ means for every 5th record starting from 1 we have a index.

1, 6, 11, 16, - - - -

Advantage :- here we will save a lot of space.

But Sparse Indexing = cannot be used in every case. where as dense indexing can be used.

Indexing techniques (types)

01. Primary Indexing :- Says that :- Indexing done on the primary key or any superkey.

↳ data must be ordered on Index.

↳ it is always sparse Index.

means if indexes are sorted on the basis of primary key then data must also be sorted in that Primary key only

↳ kind of clustered Indexing.

* anchor record :- The starting record of a disk block.

Now consider the following table stored in 3 blocks:

RNO.	S_Name
1	
2	
3	
4	

B_1

5	
6	
7	
8	

B_2

anchor record

9	
10	
11	
12	

B_3

its index block table

1	B_1
5	B_2
9	B_3

Now if we run the following query:-

```
select * from student
where R.NO. = 1
```

then no. of blocks accessed :- 2

- one for accessing the index block
- one for accessing the data block

Now if there were more than 1 index blocks :- then we will do the binary search ⇒ which will take $(\log_2 n)$ time to search our record

So total time = $\log_2 n + 1$ = or total no. of blocks accessed

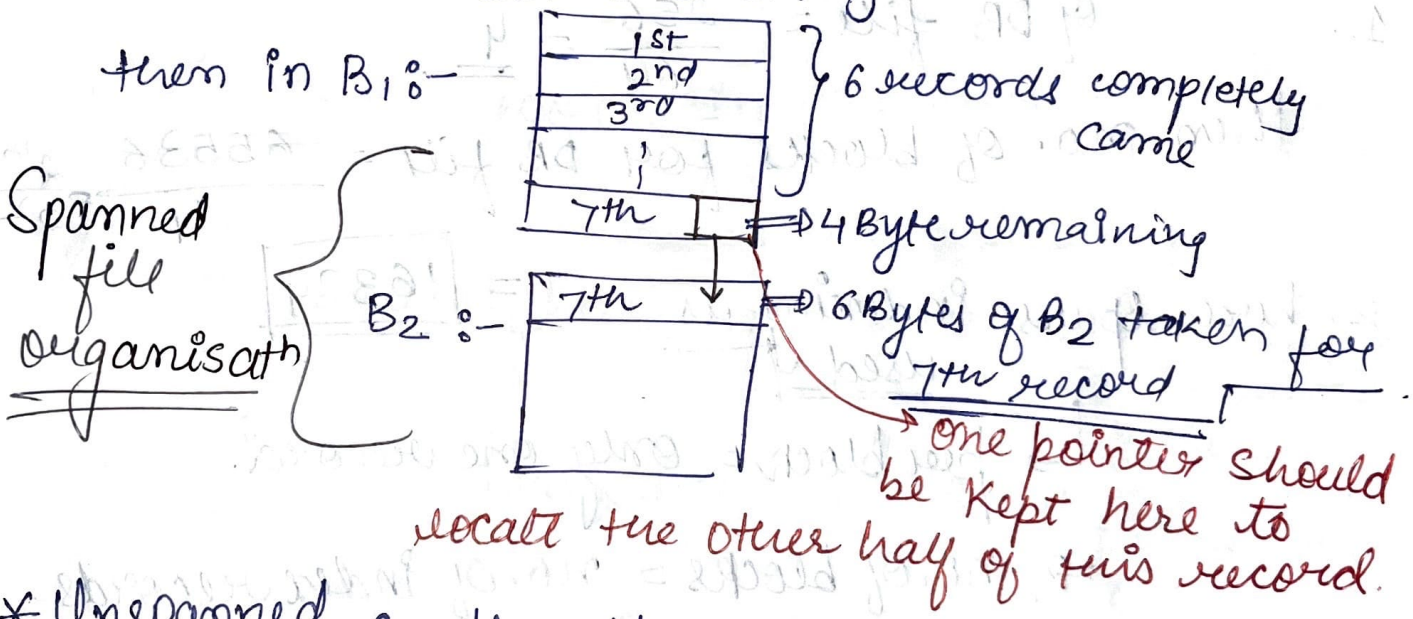
So if index file is stored over n. no. of blocks then to search in index file ⇒

$\log n$ block accesses needed

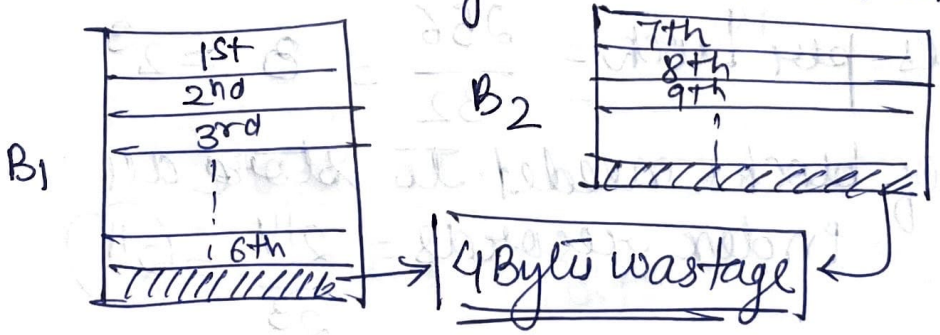
#Spanned V/S Unspanned file organisation:

* **Spanned** :- When one particular record of a DB table/file is split into two separate blocks because in 1st block there was not much enough space to accomodate that record fully.

Ex: consider one disk block of size = 64B
and one DB table = with 100 records
and each record size = 10B



* **Unspanned** :- Here No splitting of any record: either storing a record completly or storing in another block completly only.



Que. Consider a DB file of 65536 records, each of size 64B. Key field is 10B & block pointer is 22B. Assume that block size is 256B

1. the no. of blocks req^d to store file?
2. — " ——————— store the index file for primary indexing?

Soln. In one block of disk, no. of records of DB file = $\frac{256}{64} = 4$.

then no. of blocks for DB file = $\frac{65536}{4} = \frac{2^{16}}{2^2}$

$= \boxed{16384}$

2. here sparse indexing is used.

↳ per block = only one record.

⇒ no. of blocks = no. of index records.

no. of index records = $16384 = 2^{14}$.

size of each index record = $10+22 = 32B = 2^5 B$

no. of indexes per block = $\frac{256}{32} = 8 = 2^3$

then no. of blocks needed to store all index records = $\frac{2^{14}}{2^3} = 2^{11}$

Qu Consider a DB file with 2^{20} records each of size 128B. The indexing is done on primary key of the table and to store p.k. of each record 8B storage needed. The disk block size = 1KB. the disk has total space = 2^{34}B. The D.B file is primary indexed on primary key.

① no. of blocks needed to store D.B. file.

② —— 11 —— index ~~4/12~~

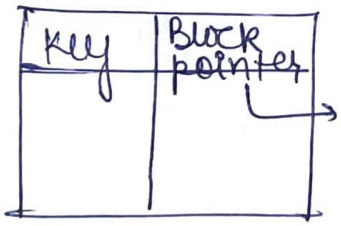
Soln. ① no. of records in one block :- $\frac{2^{10}B}{128}$

$\Rightarrow 2^3 = 8$ records

and no. of blocks needed = $\frac{2^{20}}{2^3} = \boxed{2^{17} \text{blocks}}$

② Size of one index record = 8B + ~~11B~~ = ~~19B~~ 11B

Block = 3 pointer bits



pointer which contains the address of that particular block.

address bits of disk blocks $\Leftarrow \frac{2^{34}}{2^{10}} \Leftarrow 2^{24}$ total disks

↳ 24 bits req'd ~~for~~ for disk

or

3B = for Block pointer.

now each Index record size = 8+8 = 11B.

Now two different values Based on two types of index :-

Spanned

total index file size
$= 2^{17} * 11B$

no. of indexes = total no. of blocks

now blocks :-

$$\frac{2^{17} * 11}{1024B} = 1408 \text{ blocks}$$

unspanned

= no. of indexes in each block = $\left\lfloor \frac{1024}{11} \right\rfloor$

$= 93$

no. of blocks to store indexes = $\left\lceil \frac{2^{17}}{93} \right\rceil = 1410$ blocks

Lesson 10

29/09/2022

02. Clustering Indexing :-

→ Indexing done on Non key field.
→ Data must be ordered on index field.
→ in non key field = we can have the repeated values.
and in index file :- only distinct values will be kept.
So we are not keeping index for each record
↳ hence it is Sparse indexing.

Ex :-

1	A
2	A
3	A
4	B

B_1

5	B
6	B
7	B
8	B

B_2

9	B
10	B
11	C
12	D

B_3

13	E
14	F
15	F
16	G

B_4.

→ Here we have 7 distinct values :-
A, B, C, D, E, F, G

Now the index storing blocks :-

SName	Block pointer
A	B_1
B	B_2
B	B_3
E	B_4

Now if we search for B in index block :- then it shows either B_2 or B_3

So here we will have the indexing of only 7 distinct values

Such that :-

A	B_1
B	B_1
C	B_3
D	B_3
E	B_4
F	B_4
G	B_4

→ Starting block of each distinct value of non key field

Qu. DB file size 1G records = 2^{30} records

record size = 64 B

Block size = 4096 B

Index field = 10 B

Block pointer size = 22 B

No. of distinct values in index file = 16384

Indexing is done on non key, data is ordered on non key and indexing is done on each distinct non key value.

↳ now this is the clustering indexing only.

So find no. of blocks used to store the DB file and no. of blocks used to store the index file??

So no. of records per block = $\frac{2^{12}}{2^6} = 2^6$

(1.) no. of blocks needed = $\frac{2^{30}}{2^6} \rightarrow (2^{24})$

(2.) no. of distinct values in index file = $2^{14} = 16384$

one index size = $10 + 22 = 32\,B = 2^5$

in one block no. of index records per block = $2^7 = \frac{2^{12}}{2^5}$

no. of blocks to store 2^{14} indexes = $\frac{2^{14}}{2^7} = (2^7)$

Secondary Key Indexing :-

→ Indexing done on ~~primary~~ candidate Key or any super Key.

→ Data must not be ordered on index field.

→ it is dense index. ⇒ keeping a block pointer for each record of DB file.

* So this indexing becomes space consuming.

Ex :- the index blocks :-

R.No.	Block pointer
1	B_1
2	B_2
3	B_3
4	B_1
5	B_4
6	B_4
7	B_3
8	B_2
9	B_1
10	B_4
11	B_1
12	B_2
13	B_3
14	B_3
15	B_4
16	B_2

R.No. Name

1	
4	
9	
11	

B_1

16	
2	
12	
8	

B_2

7	
14	
3	
13	

B_3

15	
10	
5	
6	

B_4

⇒ data and the index field are not ordered ⇓ which is acc. to defn

Qu: DB file size = 2^{40} B, record size = 128B, Block size = 4096B, Index key field = 10B, record pointer size = 22B, using unspanned file organisation

~~Indexing~~ of

↳ Secondary key Indexing

Indexing is done on any key. data is Unordered on key and indexing is done for each key value.
↳ dense

then find :-

1. No. of blocks required to store database file
2. No. —— " —— index file

Soln no. of records in dB file = $\frac{2^{40}B}{2^7B} = 2^{33}$

no. of records per block = $\frac{4096B}{128B} = 2^5 = 32$

(1) no. of blocks = $\frac{2^{33}}{2^5} = 2^{28}$

(2) no. of records = no. of index records = 2^{33}

each index record size = 10+22 = 2^5B

no. of index records per block = $\frac{4096B}{32B} = 2^7$

no. of blocks need to store indexes = $\frac{2^{33}}{2^7} = 2^{26}$

Secondary Non Key Indexing :-

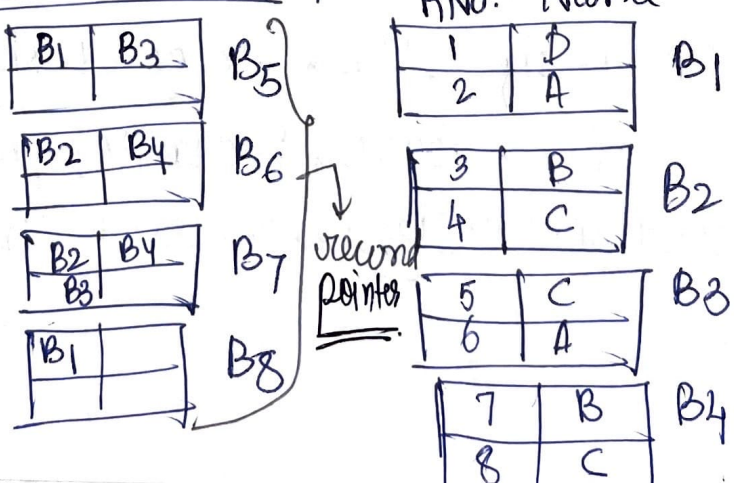
↳ Indexing done on non-key field.

↳ data must not be ordered on index field

↳ it is sparse index.

Ex: Index for each distinct value

Name	BP
A	B5
B	B6
C	B7
D	B8



Recap

Ordering	Key/Non Key	Type
1. ordered	Key	Primary + Sparse
2. ordered	Non key	Clustering + sparse
3. unordered	~~Non~~ Key	sec. key + Dense
4. unordered	Non key	sec. key + Sparse.

Ques. DB file size = 10^8 records
record size = 400 B
Block size = 4096 B
Index key field = 16B
index pointer size = 4B
unspanned file organisation
Indexing is done on key; data is unordered on key and indexing is done for each key value.

Secondary key

1. No. of blocks req^d to store DB file.
2. No. of blocks req^d to store index file.

Soln. no. of records per block = $\left\lfloor \frac{4096B}{400B} \right\rfloor = 10$

①

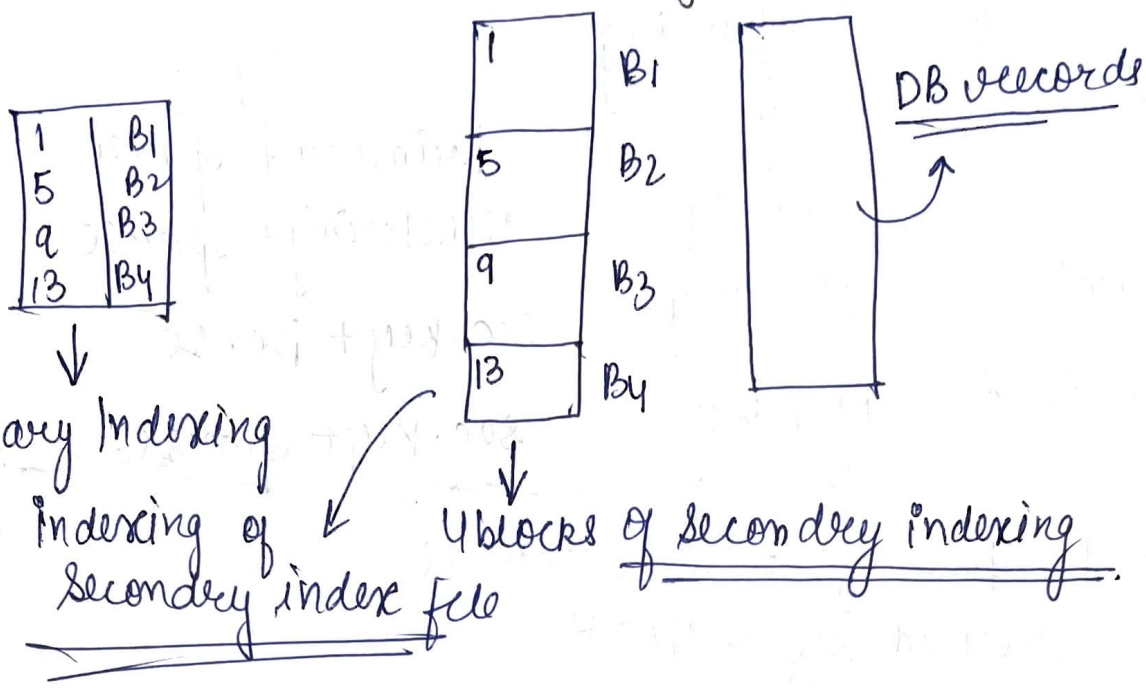
no. of blocks for d.B file = $\frac{10^8}{10} = 10^7$ blocks

② Index record size = 16+4 = 20 B

no. of indexes per block = $\left\lfloor \frac{4096}{20} \right\rfloor \frac{B}{B} = 204$

no. of blocks needed for index file = $\left\lceil \frac{10^8}{204} \right\rceil = 490197$

#multi level Indexing#



now if secondry indexing is very large then its Primary indexing will also be large then one more level of indexing or more than one level also needed.

↳ 3 or 4 or 5 - - - level Indexing it would be !

* In multilevel indexing of n-levels :-
no. of blocks accessed to get record = n+1
↳ for index blocks ← for DB records

* In general it is = $\log_2$ (no. of blocks to store index) + 1

29/09/2022. #Lesson 11#

B-Tree :- used to use the No. of accesses of index blocks drastically.

↳ Tree Based indexing

↳ Dynamic Indexing technique.

↳ Based on insertion and deletion, the tree automatically adjusted

↳ Self Balancing Search Tree.

↳ means every node will have its height of right sub tree = height of left sub tree.

→ Binary Search Tree :- max^m 2 children of each node and each node can contain only one 1 Key.

→ if indexes are stored using BST :- and we have 2^{30} nodes in that BST :-
then searching will require :- $\log_2(2^{30})$
= 30 blocks.

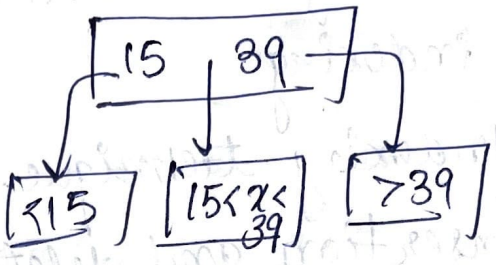
So disadvantage of BST :-
tree grows only vertically hence $\log_2 n$ = larger value for block accesses.

So the soln :- tree should grow horizontally & vertically.

So n-ary Search tree :- max^m n children

now if we have 3 ary search tree :-

max^m. 3 children per node and 3-1 = 2 keys per Node



now B-tree :- → p-ary search tree = max^m p children per node

An order-p B-tree :-

1. Every node other than root should have atleast $\lceil p/2 \rceil - 1$ keys.

2. In every node there are atmost (p-1) keys and p tree pointers (children)

Ex $n=3$ → max^m children = 3 | min^m = 2

max^m keys = 2 (3-1) | $min^m = \lceil \frac{3}{2} \rceil - 1 = 1$.

3. root can have min^m 1 key.

4. All leaves should appear on same level.

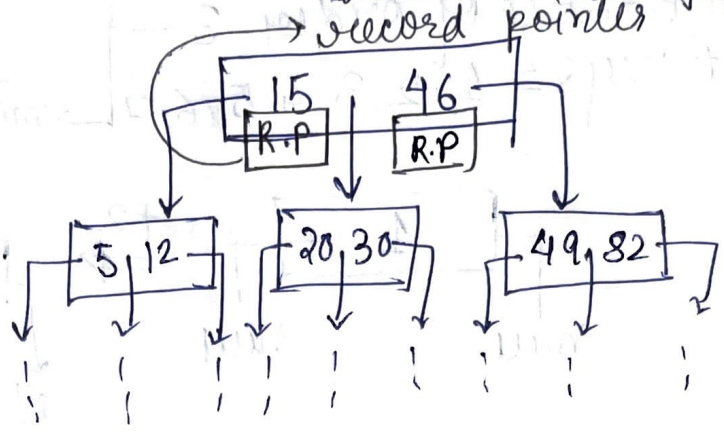
Ex: $n=5$

	Children	Keys
max	5	4
min (not on root)	3	2

Consider the student's DB :-

R.No.	Name
1	A
2	B
3	B
4	C
5	D
⋮	⋮

now using order 3 B-tree for indexing :-



→ So here along with key :- we will also keep one record pointer = which will denote that where in our DB or disk → this particular key value is located.

→ So on a particular Node a B-tree :-
we are keeping following three things :-

1. Key = field value on which indexing is done
2. Record Pointer = pointer to record associated with key.
3. Tree pointer :- pointer to next node in B-tree.

→ So in a B-tree of order p :-

max^m on a node :-

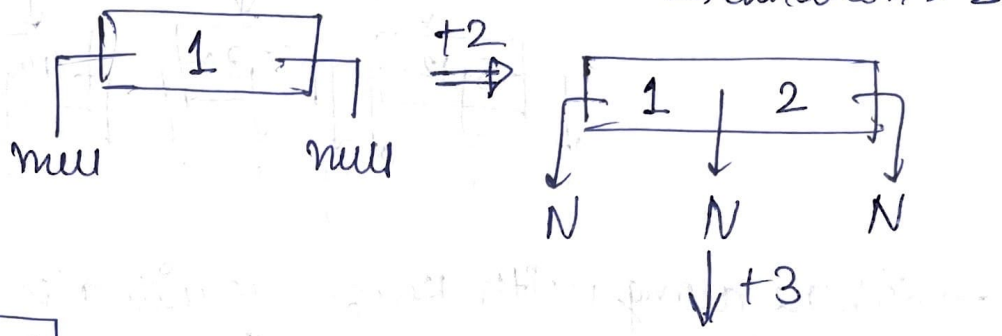
↳ Key : p-1

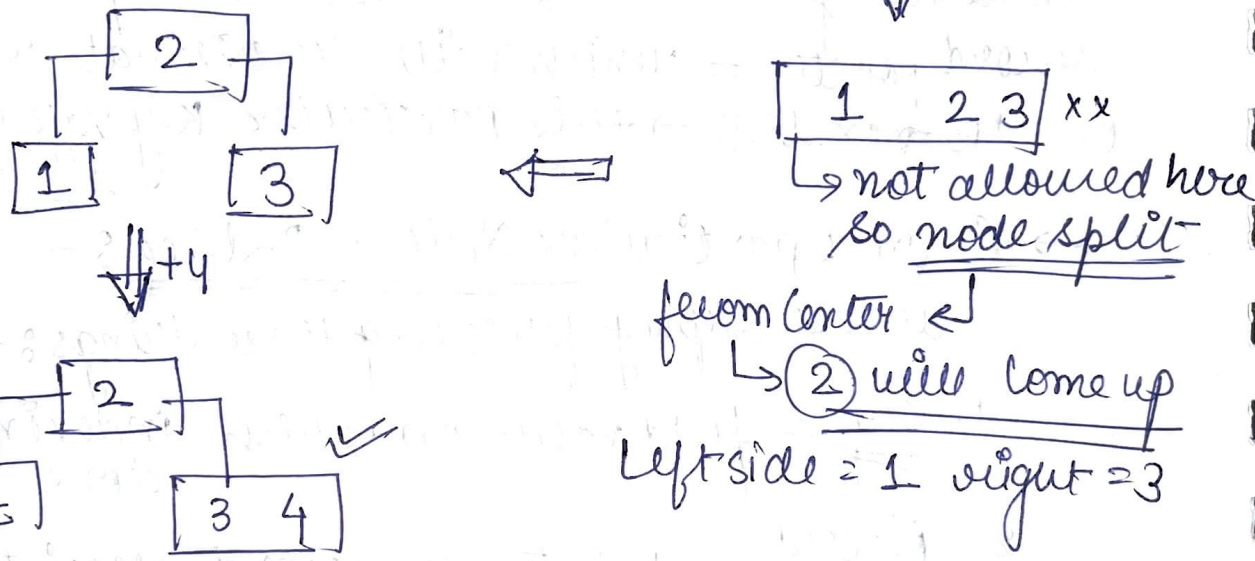
↳ Record Pointer : p-1

↳ tree pointer : p

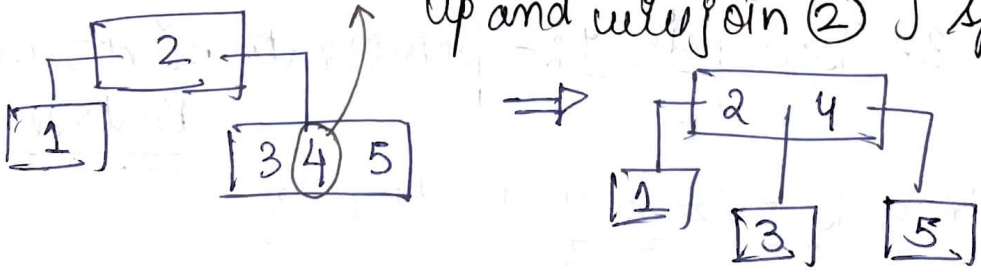
Insertion in B-tree :- always on leaf node

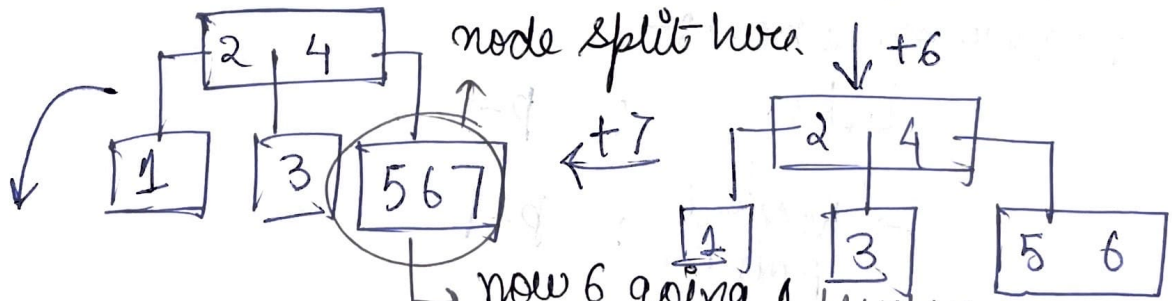
→ So B-tree of order 3 and insert keys = 1, 2, 3, 4, 5, 6, 7
- max^m → key = 2, child = 3
- min^m → key = 1, children = 2



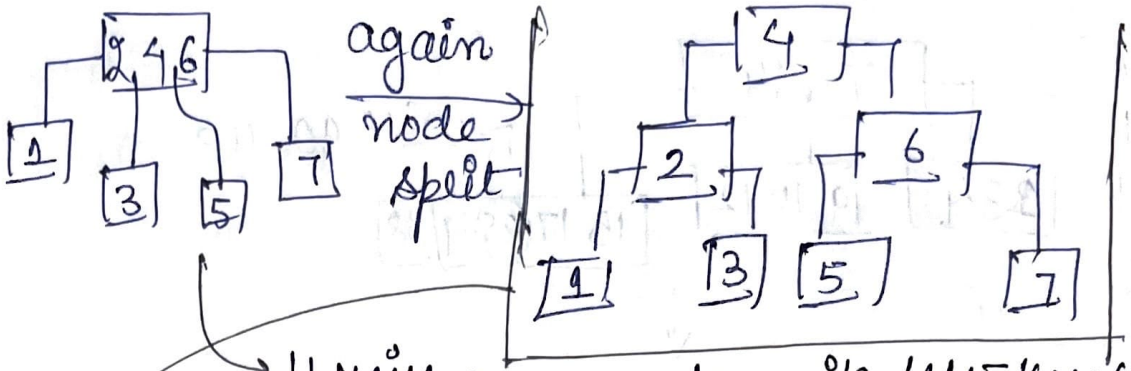


Now this will go one level & node up and will join ② & split.





now 6 going 1 level up its left will be (5) left child and its right will be right child (07)



4 will go up :- then its left key (2) will become its left child and then its right key (6) will be its right child.

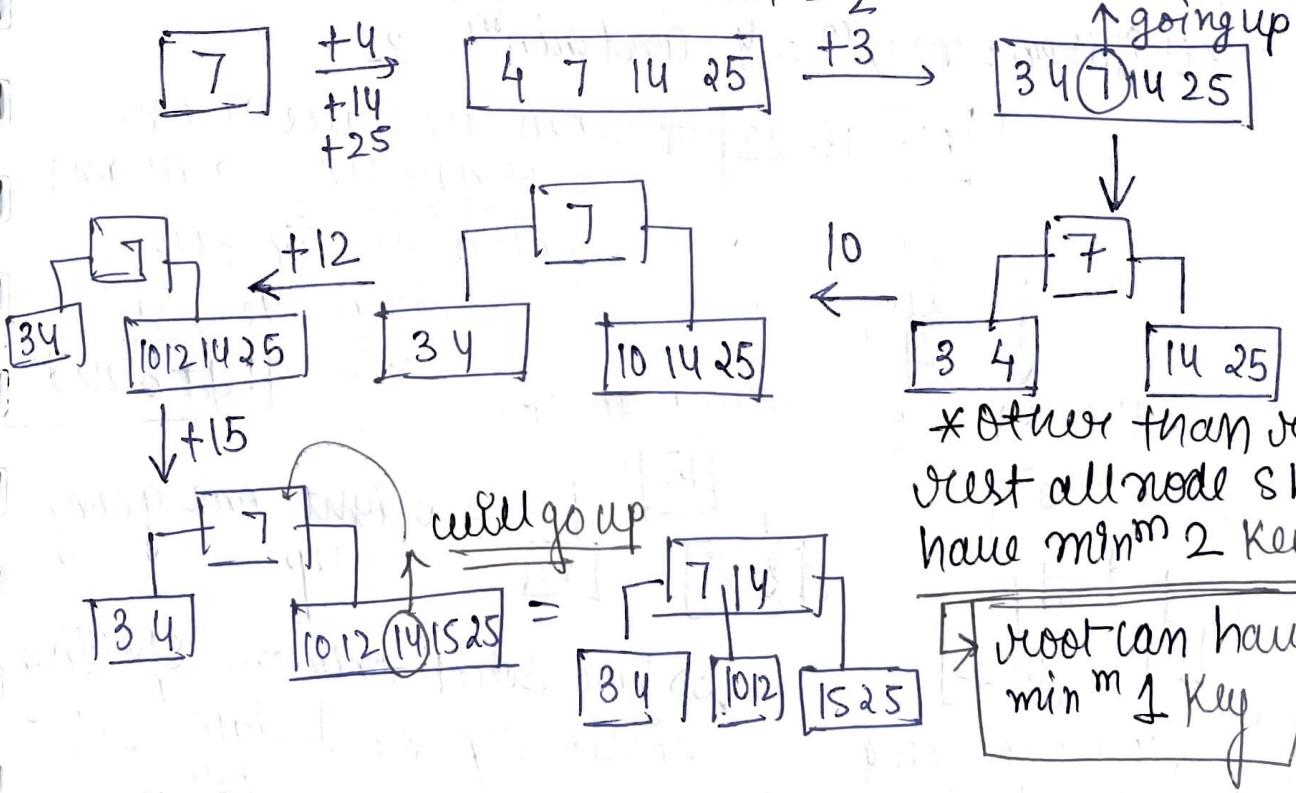
And final B-Tree.

⇨ Insert the following keys :- in B-tree of order = 5

Keys = 7, 4, 14, 25, 3, 10, 12, 15, 17, 9, 29, 1, 38, 8, 11.

Soln.

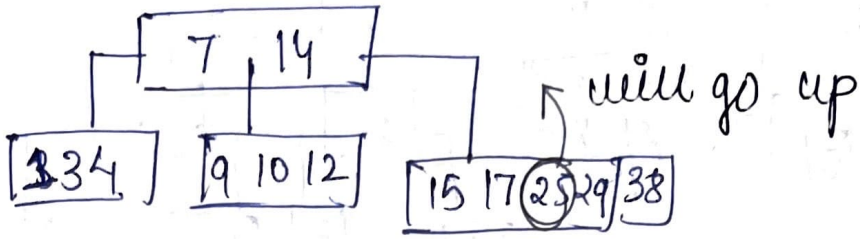
Order = 5
- children
  - max = 5
  - min = 3 = ⌈5/2⌉
- keys
  - max = 4
  - min = 2

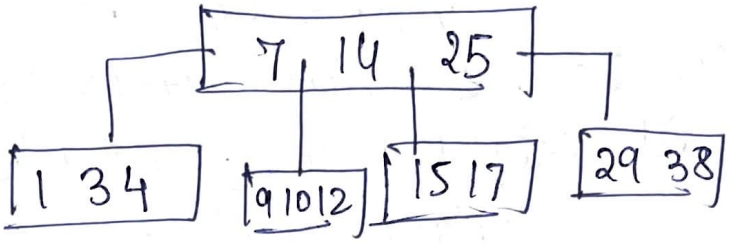


* Other than root rest all node should have min^m 2 keys

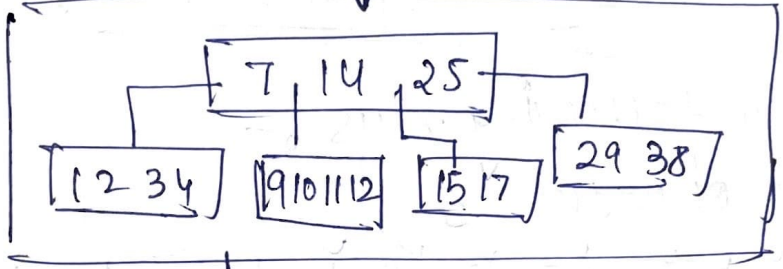
root can have min^m 1 key

+17 → +9 +29 +1 +38





+2, +11



→ This is our final tree !

Ex:- B-tree of order = 4 → max^m keys = 3 and min^m = 1

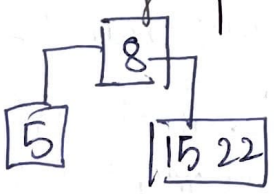
Insert keys = 15, 5, 8, 22, 10, 1

→ children = max^m = 4 and min^m = 2

5 8 15 22 ⇒ now we have even order here → means not a center key

2 ways to split or

either (8) will go up

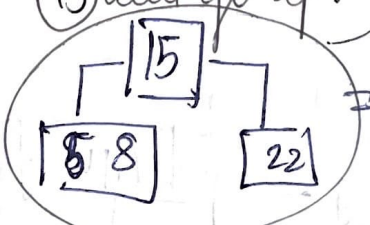


left one going up

right side more keys → right biased

(15) will go up.

left side more keys = left biased

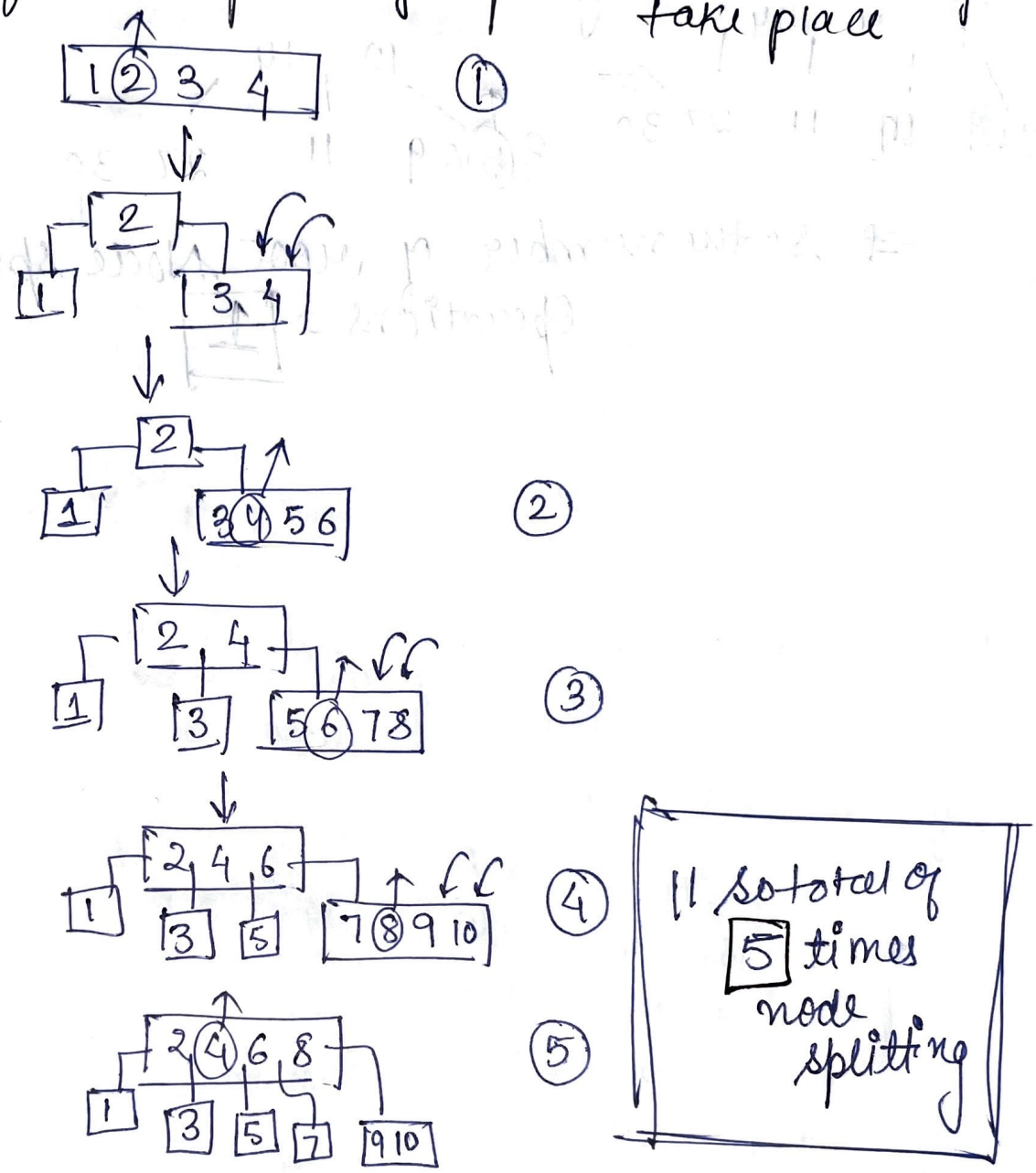


= right one going up

So here select only one splitting either left or right and throughout the question follow that splitting only.

Qu. A B-tree of order 4 is built from scratch by 10 successive insertions. What is the max^m number of node splitting operations that may take place

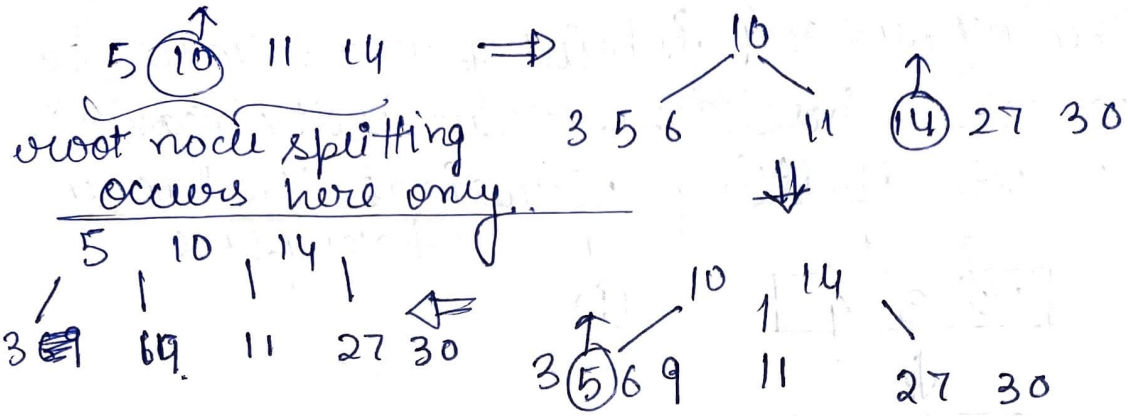
Soln.



Qu. A B-tree of order 4 is built from scratch by successive insertions of following keys in the given order:-

10, 5, 14, 11, 3, 6, 30, 27, 9

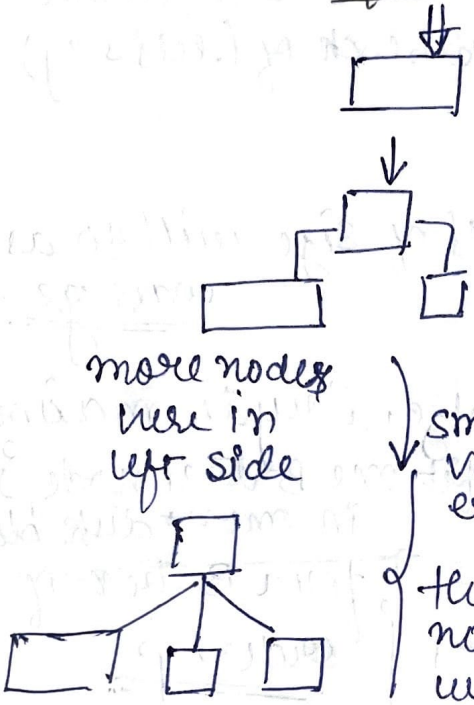
what is the no. of root node splitting operations that may take place with right biasing.



⇒ So the number of root Node splitting operations = 1

30/09/2022 · # Lesson 12 #

⇒ now more splitting happens in which :- left biasing or right biasing?



So in left biasing :- insertion of decreasing sequence will cause more node splits.

→ in left-biasing if we insert elements in ↓ing order then from LHS only more splitting will be there.

Similarly here in right biasing :- max^m split if insertion done in increasing order.

⇓

when you insert the elements which are larger and more larger and more larger ------ then elements will be entered at right most leaf child :- then splitting will also be max^m = ~~max~~ in case of right Biasing.

practical Implementation of Node on Blocks :-

→ The B-tree is also stored in disk Blocks such that one Node of the B-tree is stored in one disk block.

now consider if B-Tree is of order = 3
then max^m key = 2
max^m Record pointer = 2
and max^m tree pointer = 3
} Total 7 values stored in one disk block of (let's say) 1024B size

can't utilize block size upto fullest.

remain is very large for just one node of a B-tree

↳ these will go as wastage.

So so --- what to do?

↳ to utilize Block size, try to maximize the order of B-tree to fit one B-tree node size in one disk block

we know that :- max^m keys = p-1
max^m Rec. ptr = p-1
max^m tree pointer = p
} for a B-tree of order p.

now total size stored in one node ≤ [(p-1) × key size] + [(p-1) × record pointer size] + [p × tree pointer size] ≤ Block Size.

* tree pointer == next block pointer.

Que: Key size = 16B
Block pointer size = 32B
record pointer size = 48B
Block size = 8192B

↳ If a B-tree of order p is implemented then what is the max^m p value?

Soln ⇒ (p-1) × 16 + (p-1) × 48 + p × 32 ≤ 8192

$$p \leq \left(\frac{8192 + 64}{96}\right) \Rightarrow p \leq 86$$

Ans ⇒ $p_{max} = 86$

⇨ Height of the B-tree :-

P order B-tree :-

and total Nodes = n

then Hmin = $\lceil \log_{p}(n+1) - 1 \rceil$

Hmax = $\left\lfloor \log_{\lceil p/2 \rceil} \frac{n+1}{2} \right\rfloor$

---

deletion in B-tree :- two Cases

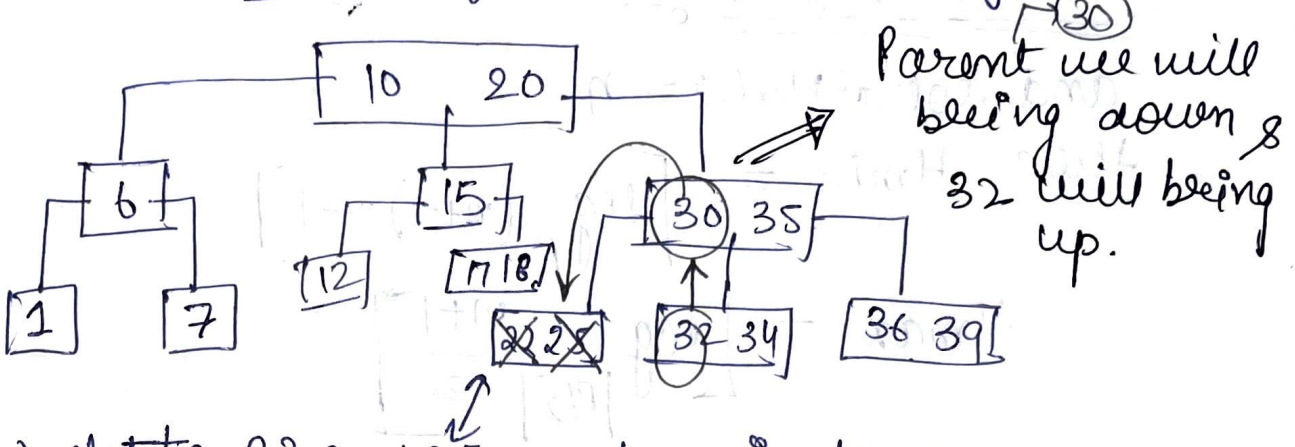
Case ① deletion in leaf node :- here also we will have 3 cases :-

1. if after deletion :- then node have more than or equal to the min^m no. of allowable keys = then no problem at all.
   ↳ will not do any adjustments!

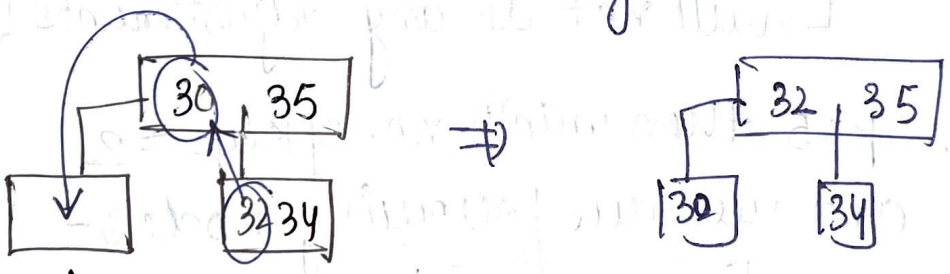
   Ex :- p=5 then min^m no. of keys = 2
   and we have following node :-
   [15 19 20 ~~25~~] → after deleting this we have 3 keys in the Node :- which is not a problem!

2. Now if after deletion : there is a violation of min^m no. of keys :- then borrow a key from sibling node & do the rotation with parent

now consider the following 3 order B-tree
max^m keys = 2 and min^m keys = 1.

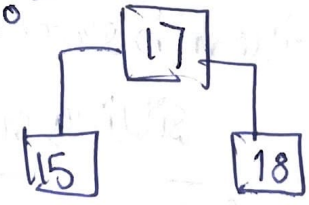


now delete 22 and 25 :- then in that node
no. of keys = 0 = violation
of min^m no. of keys
then Borrow a key from sibling Node (which is closest) = 32 we will borrow from right sibling
↳ since it has 2 keys
So 1 key can be taken.



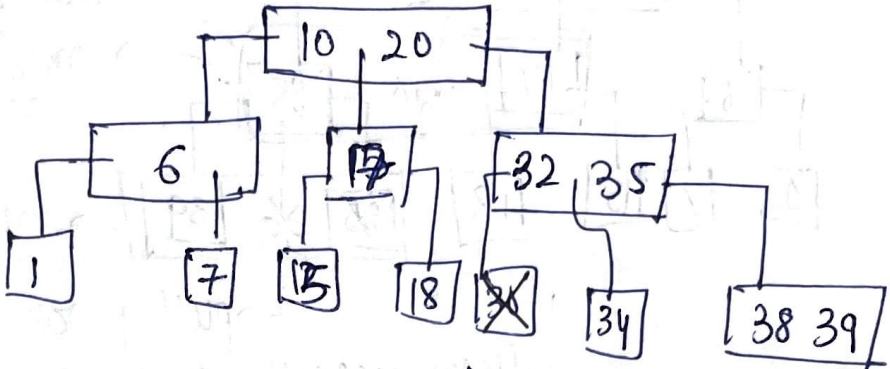
↳ blank after deletion.

* now we will delete 12 :- then from its right sibling we will borrow such that 17 will go up and 15 will come down.
such that :-



now our tree :-

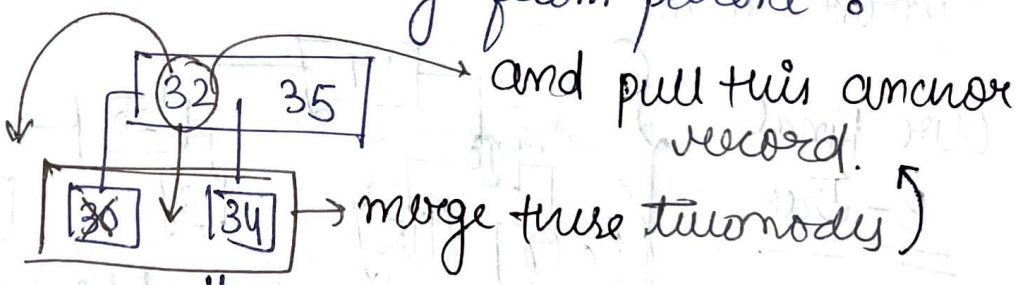
③

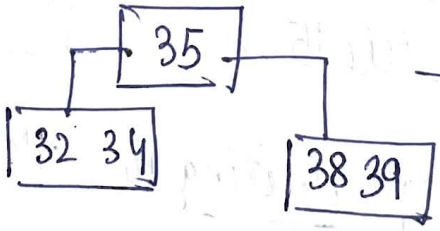


↓ now deleting 30

here from Sibling the borrow will not be possible because :- 34 this node also have minm no. of key = 1 key only.

then :- Merge the Node with sibling and pull down the anchor key from parent :-

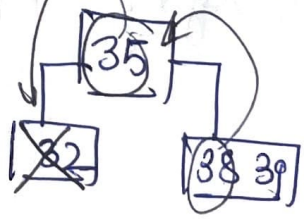


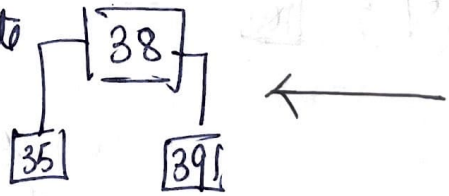


delete 34 → No problem

↓ delete 32

↓ violation here

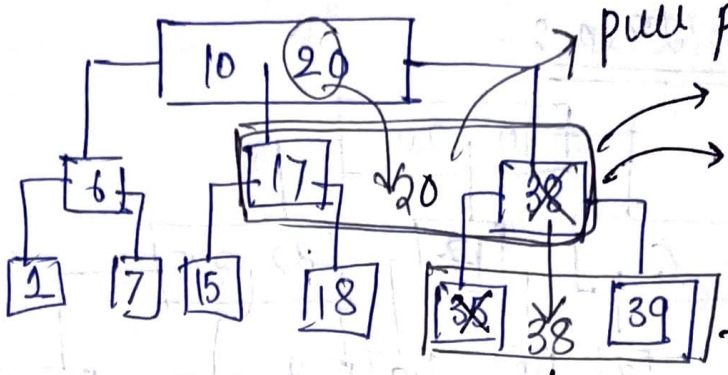




Now delete 35

we cannot take from Sibling :- then merging & pulling down the anchor key from parent.

⟹



pull parent node

→ merge

→ min^m cond^n violated.

→ merge

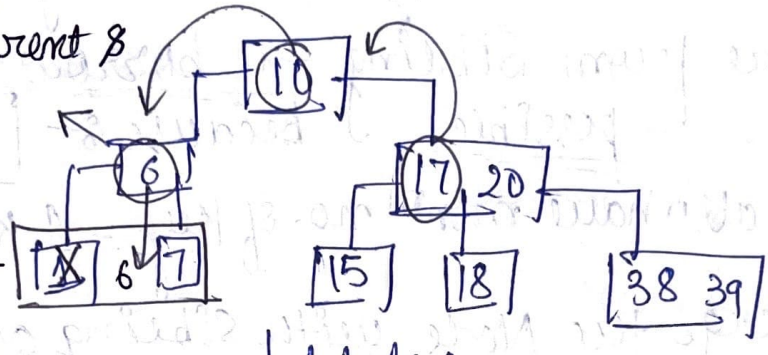
↳ pull parent down

↓ then resultant ~~tree~~ tree :-

. then for this :-

root^n with parent & right sibling!

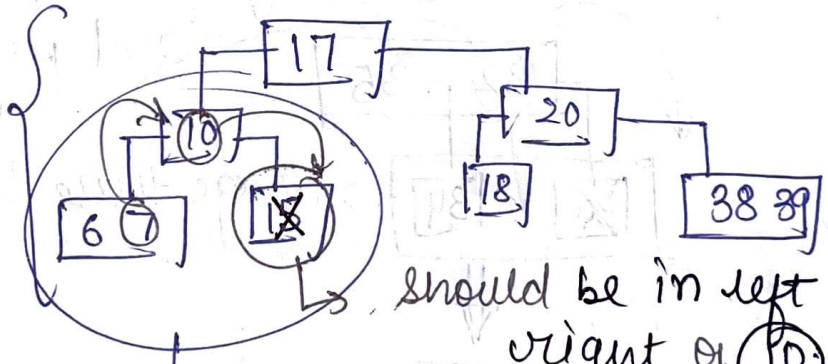
merging two nodes and bringing parent down



our final B-tree

↓ delete 1.

Our final B-tree

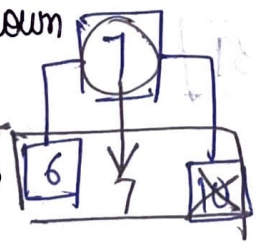


should be in left of 17 and right of 10.

↓ delete 15

Pulling down the parent key

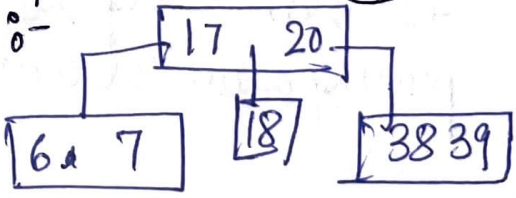
merging →



deleting 10 →

then (7) node will again become empty so (20) ke sath merge ho jayega :- and we will pull down (17)

So resultant B-tree :-



Case II :- Deletion in B-tree :- deletion in Internal Node :-

Here also we have two cases :-

1. when after deletion :- No any problem is there then well and good!

2. if after deletion :- there is violation of min^m no. of keys in the Node :-

then :- (i) Replace the deleted value with inorder success or (just ~~smaller~~ larger key) or inorder predeccor (just smaller key).

↳ these would be from Leaf Node only

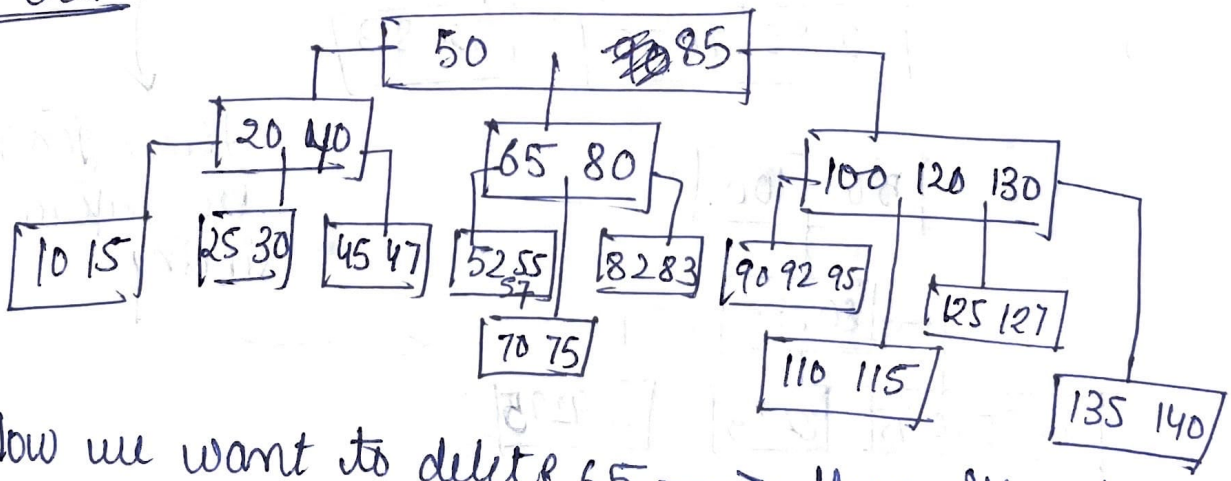
(ii) then follow the rules of deletion of key from Leaf Node.

Ex. B-tree of order 5 :-

max^m keys = 4 and

min^m keys = 2.

Now tree :-



Now we want to delete 65 → then either the inorder predeccor = 57 or inoder successor = 70 will go up in place of 65

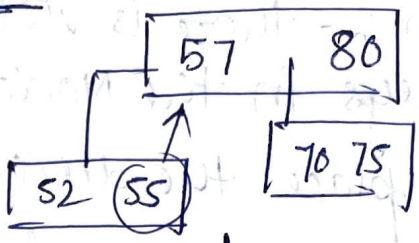
So here replacing 65 with 57 :-

now same rule will be applied for 57 deletion

↓

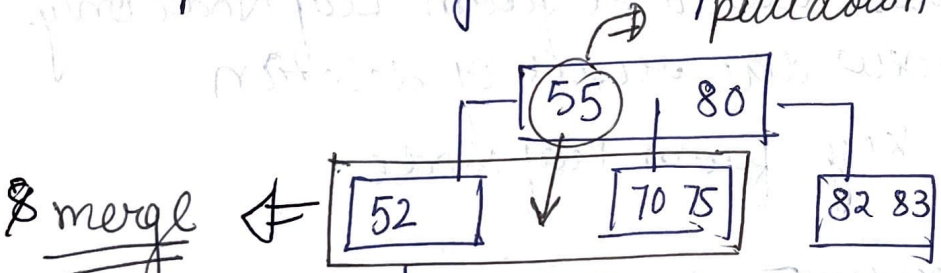
when 57 up :- no any violation in leaf node

So subtree :-



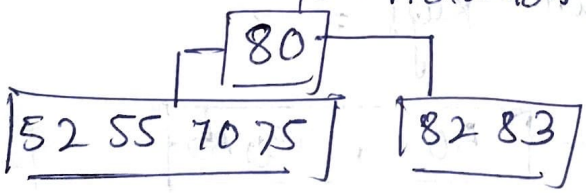
↓ now delete 57

replace it by 55 = predecessor pulldown the anchor 55

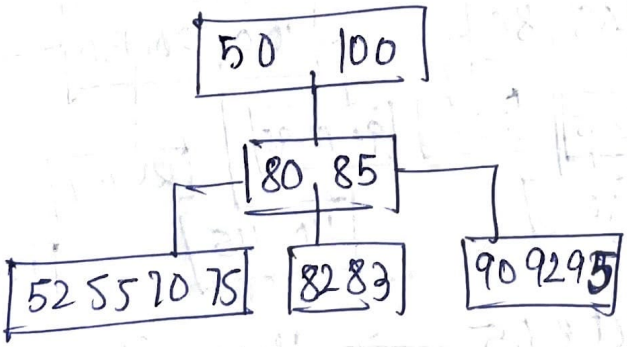


& merge ← violation here, and the sibling cannot give = so merge two nodes

↳ violation of min^m key here.



then 100 = will be taken from sibling

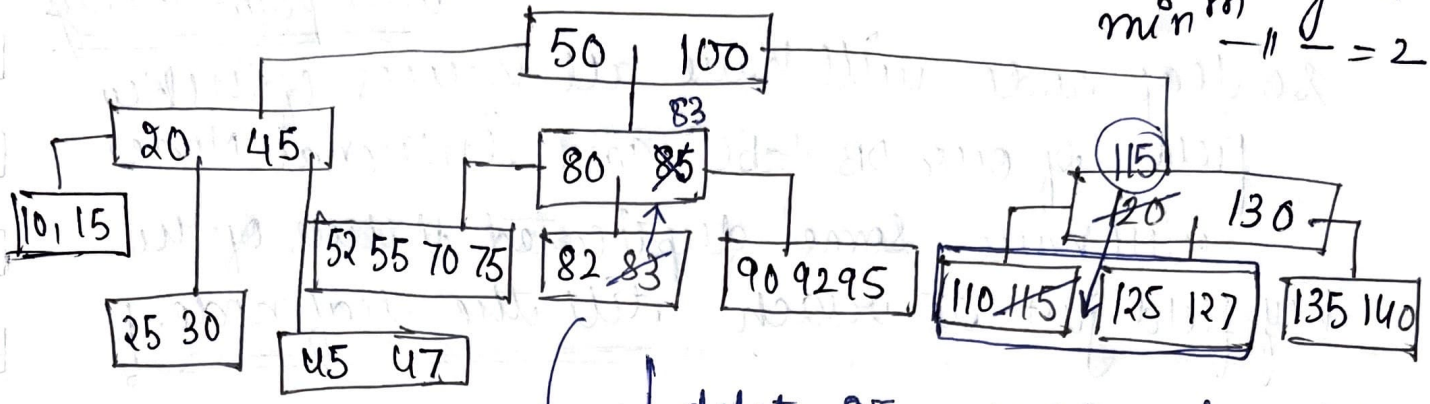


30/09/2022

#lesson 13#

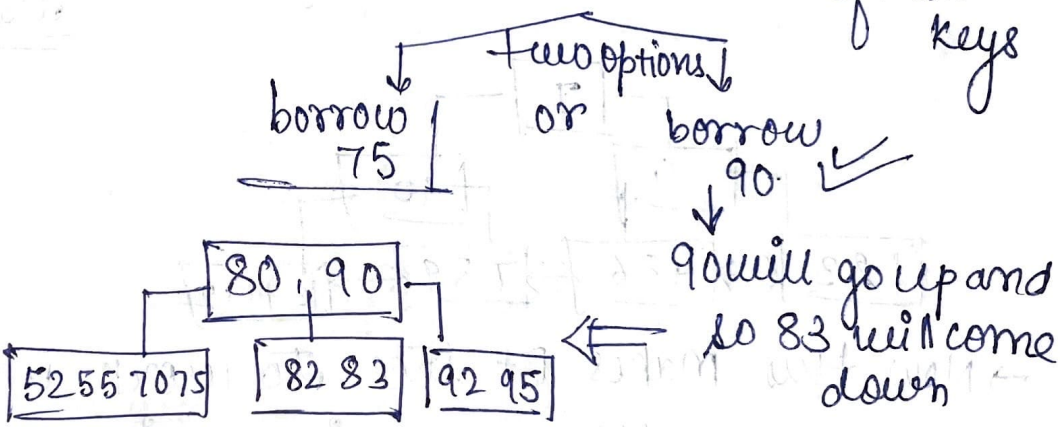
Que. delete the internal nodes one by one from the following tree :-

order = 5 max^m keys = 5
min^m " = 2



↓ delete 85 = replace it with 83

→ now here violation of min^m keys



→ Now deleting 120 :- replace by (15)

then for leaf node :- merge with sibling node.

and pull down the anchor

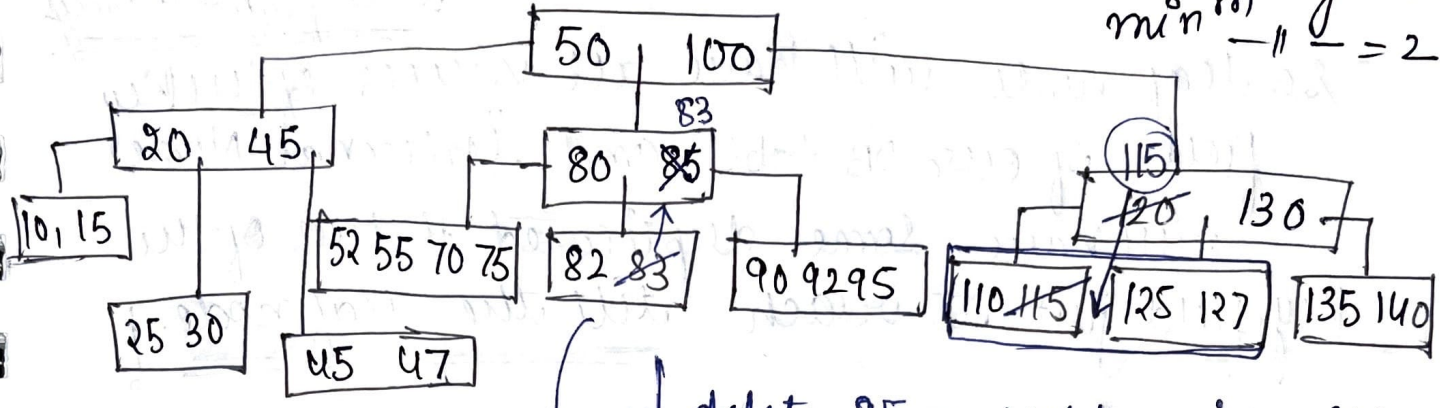
then 130 will become alone :- then merge him with its sibling and pull down the anchor key from parent node

#lesson 13#

30/09/2022

Que. delete the internal nodes one by one from the following tree :-

order = 5 max^m keys = 5
min^m " " = 2



↓ delete 85 = replace it with 83

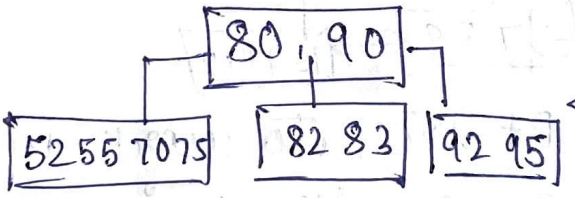
→ now here violation of min^m keys

two options ↓

borrow 75 | or | borrow 90 ↙

90 ↓

90 will go up and so 83 will come down



→ Now deleting 120 :- replace by (15)

then for leaf node :- merge with sibling node.

and pull down the anchor

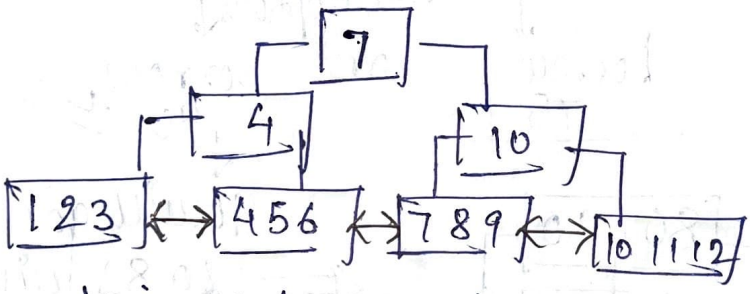
then 130 will become alone :- then merge him with its sibling and pull down the anchor key from parent node.

B+ tree :- Here we will store the Keys and record pointer only on the leaf nodes

In and internal nodes :- we will store Keys and tree pointer only.

So leaf node will have all values of the Key field of our DB table and Internal Node will have some duplicated values of the Key field just to reach till the leaf node.!

* now here in B+ tree :- All the leaf nodes are connected through the doubly linked list form :-



→ Now this makes B+ tree to work well with range queries ⇒ in which some values within a range has been asked

and so the no. of node access will be less here as compared to B-tree!

→ B+ tree will be larger (with little extra height) as compared to B-tree for same keys! just because on internal nodes of B+ have the duplicate values of the leaf nodes.

→ B tree :- beneficial when a specific value to be accessed !

↳ Because we don't have to search till leaf node everytime.

* Now Consider :-

Internal node order = p
and
Leaf Node order = q.

then :-

1. Every Internal Node other than root should have atleast $(\lceil p/2 \rceil - 1)$ keys or $\lceil \frac{p}{2} \rceil$ pointers.
2. Every internal Node can have max^m $(p-1)$ keys or 'p' pointers.
3. Every leaf Node should have atleast $\lceil q/2 \rceil$ keys and max q keys
4. All leaves are on same level
5. The leaves are connected using linked list (singly or doubly)

Now, what if order - 4 B+ tree given in question?

then internal nodes = order = p = 4

Leaf —"— —"— = q = 4

then max^m keys → internal = 4
→ Leaf = 3

min^m keys → internal = 1
→ Leaf = 2.

Insertion in B+ tree :-

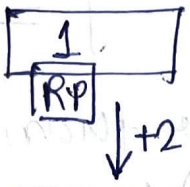
	keys min	keys max
m	1	2
m	1	2

Given :- Internal nodes order = 3
Leaf nodes order = 2

Insert = 1, 2, 3, 4, 5 using Node splitting.

→ Here also we will insert the values on leaf node only = as we did in B tree.

Insert 1 :



= root as well as leaf.

+2

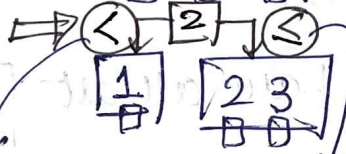
consider these empty boxes as RP

here we have to go with either left biasing or left biasing

+3

now split it since violation of max^m no. of keys happened!

duplicated (2)



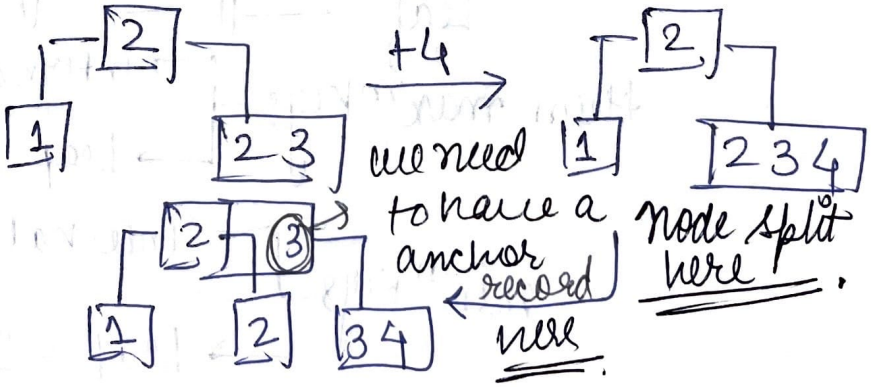
Strictly smaller value in left side.

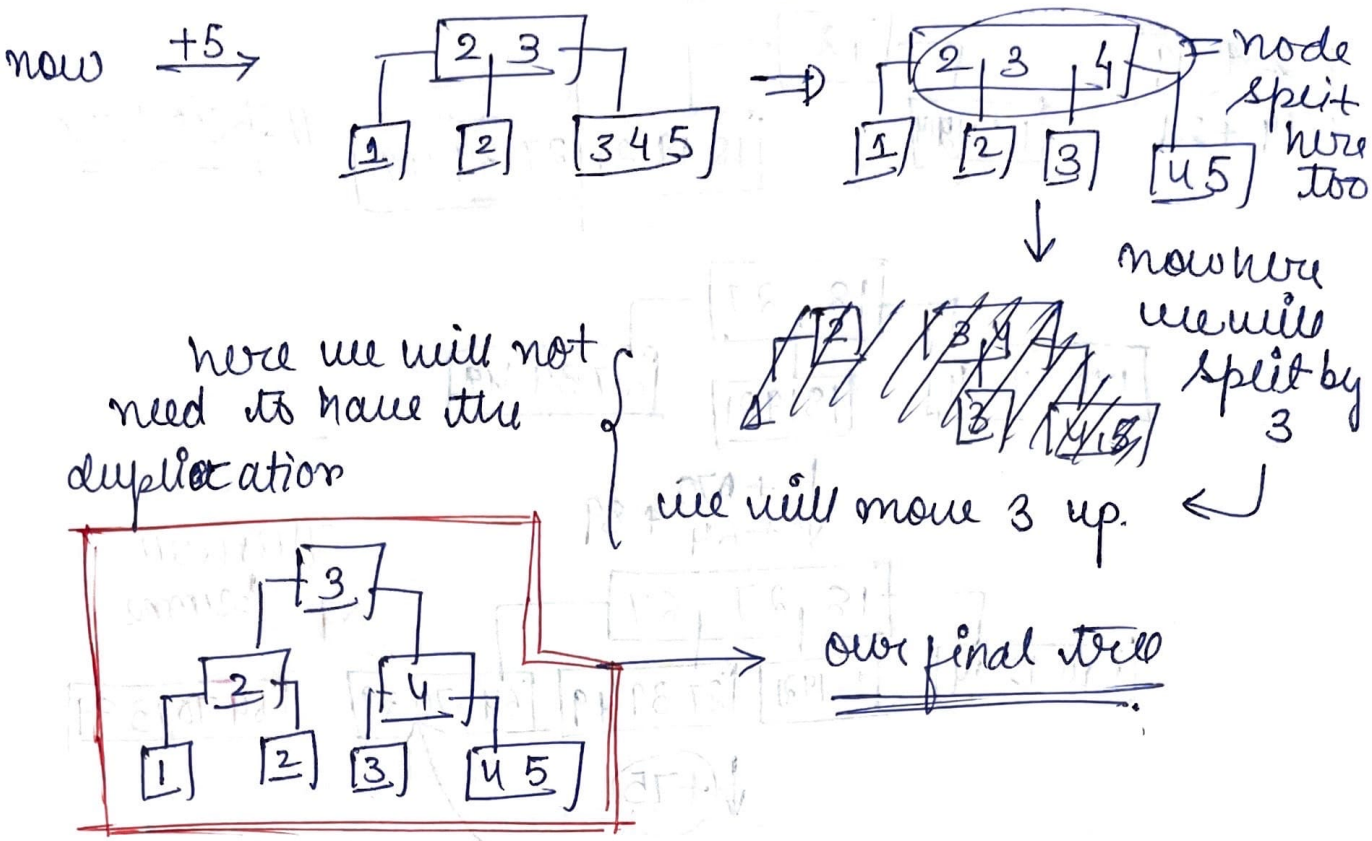
greater than & equal to values in right side

⇒ here we will not have to delete any key from leaf node. and we also have to duplicate one of the key as a internal node

Now here equal value will be kept in RHS and in LHS also we can keep = that depends on how it is mentioned in the question = both are correct!

now our tree =





So if leaf Node split happens ⇒ copy key to internal node also
↳ duplication will be there.

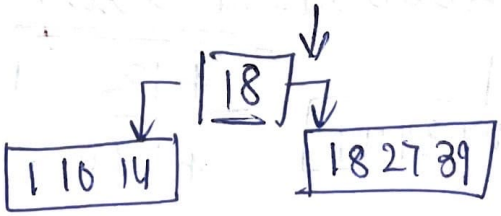
and in internal node split = then only split and no any duplication.

Now do the following insertion in B+Tree of order = 5

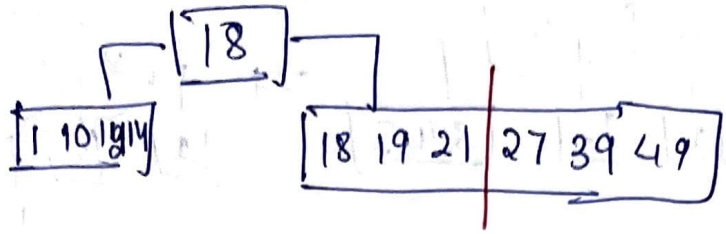
Keys : 10, 14, 1, 18, 27, 39, 49, 12, 19, 21, 70, 64, 89, 75.

Soln Now we have order = 5 then max^m keys = 5 in leaf
and min^m keys in leaf = 3
and in internal node :- max^m = 4
and min^m = 2

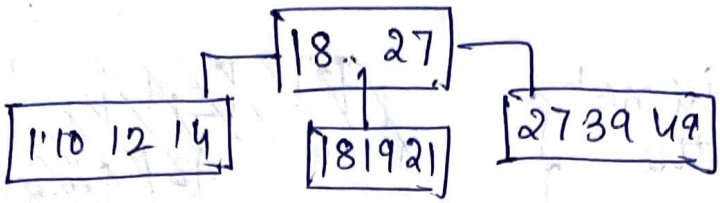
1 10 14 18 27 +39 → 1 10 14 18 27 39 = now node equal splitting division



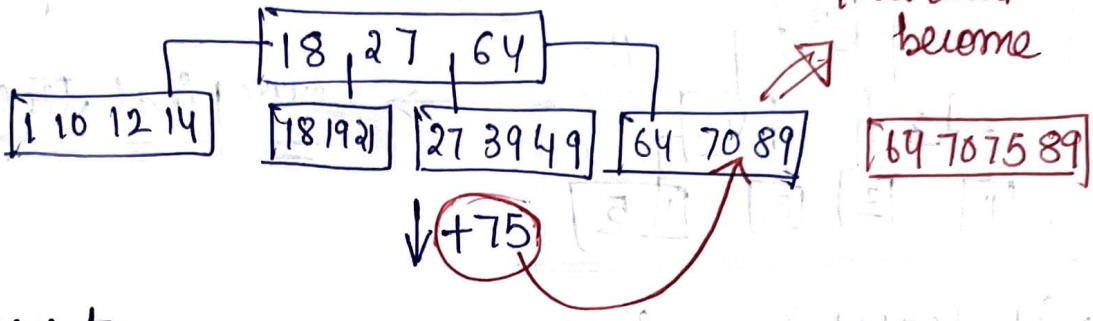
+12 +49
+19 +21



// Split here



+70 +64 +89



Now Insert

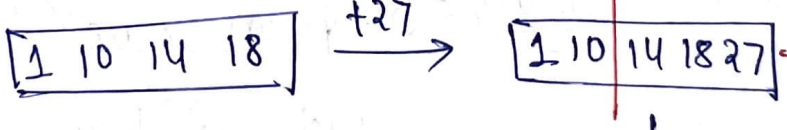
↳ 10, 14, 1, 18, 27, 39, 49, 12, 19, 21, 70, 64, 89, 75

B+ tree :-

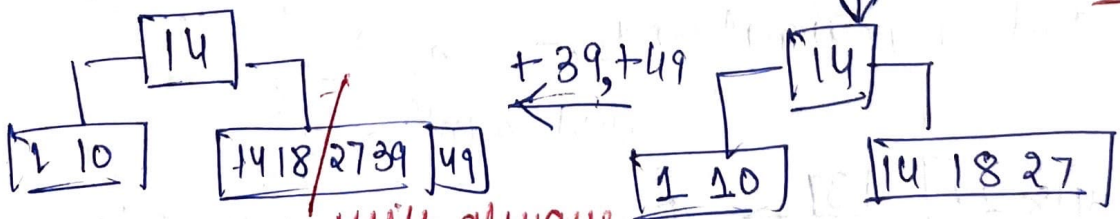
	maxm	minm
p = 4	3	2-1 = 1
q = 4	4	2

↳ here just the orders have been changed & the sequence is still same as it is

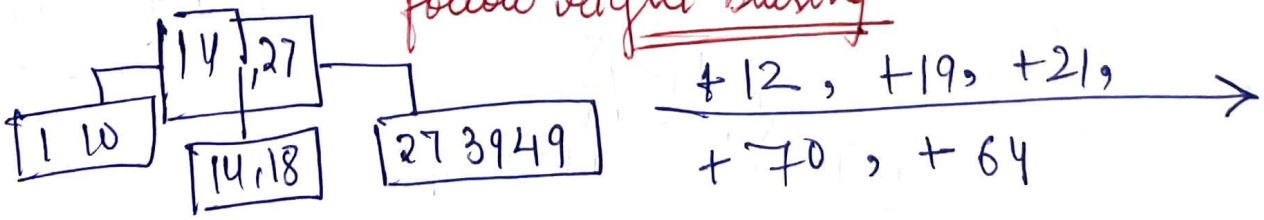
+10 +14 +1
+18

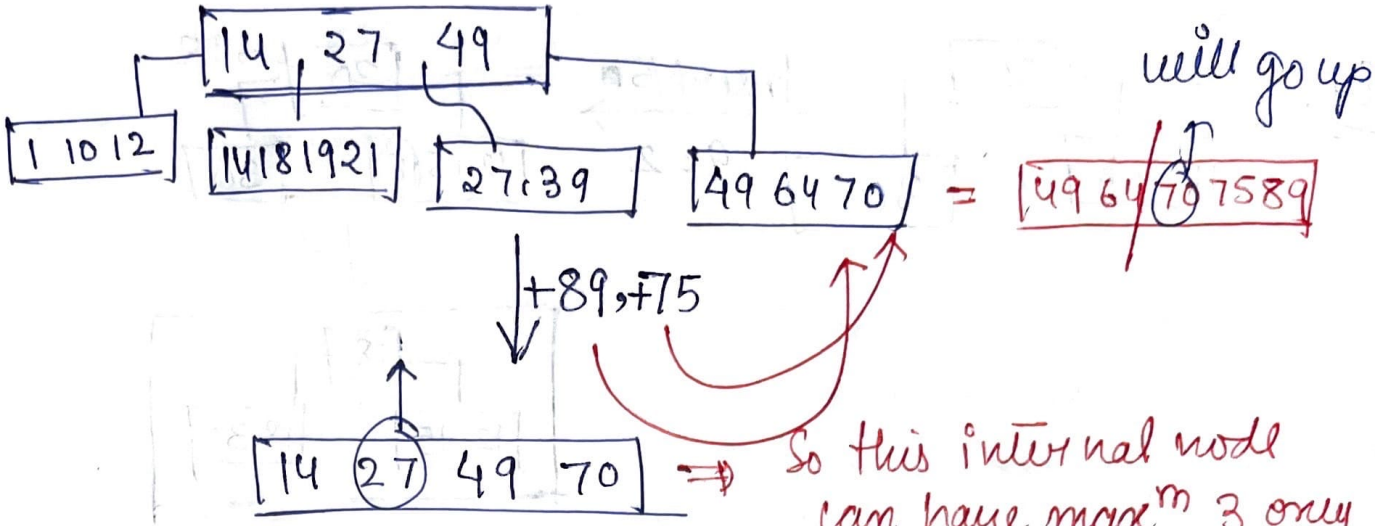


↳ node splitting going for right biasing



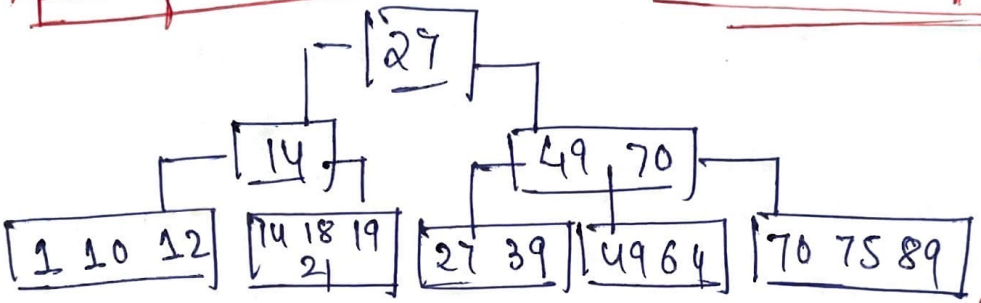
will always follow right biasing





So this internal node can have max^m 3 only so again splitting ← either 27 or 49 will go up !

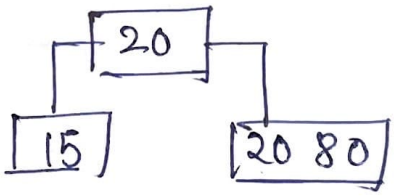
Just like the normal split of a node.



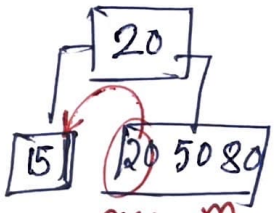
This is our final req'd Tree !

→ Problem with Node split method is :-
when insertion is done = max^m no. of node split = height of tree

So using :- Key distribution



Insert 50 with q=2

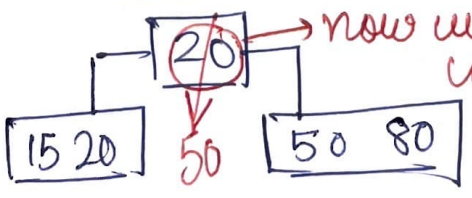


max^m two keys allowed here

Yes we have :- So we will distribute 20 to Left Sibling.

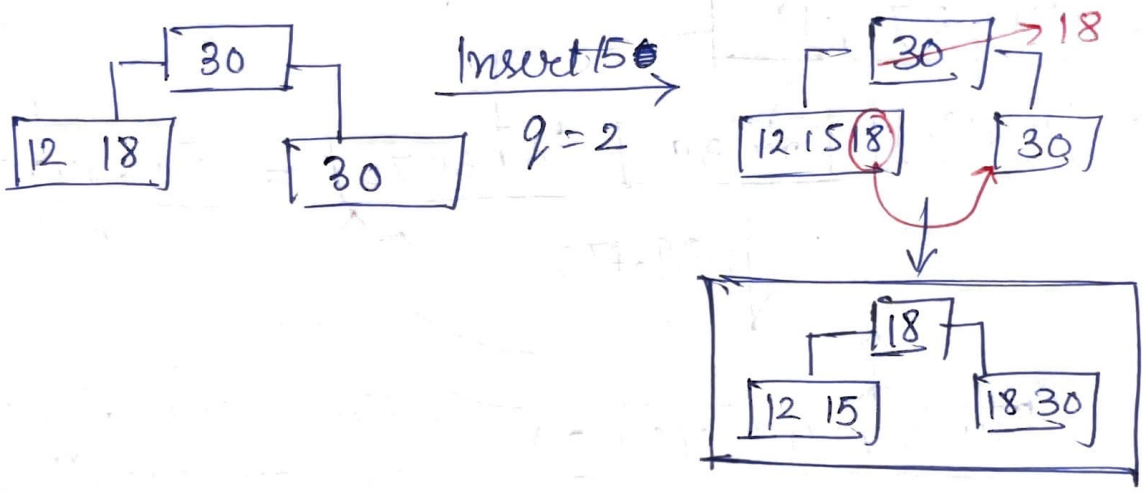
So here look for the Sibling who have keys lesser than max^m no. allowed

↳ distribute one key. = 20 is copied to left sibling.



now we have to change this to 50

Ex.



03/10/2022. #Lesson 15#

→ Cascadeless schedule.

* You cannot have a dirty read.

T_1	T_2
R(A)	
W(A)	
	W(A)
commit	
	R(A)

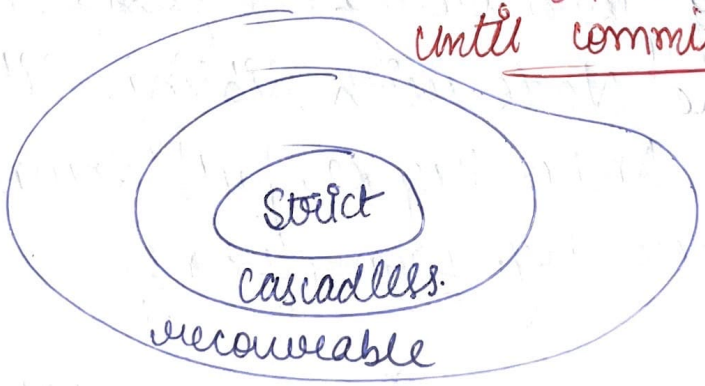
Here we can allow the blind write but not the dirty read

Strict Schedule.

* No read or write allowed until the 1st transaction has done commit

T_1	T_2
R(A)	
W(A)	
commit	
	W(A)
	R(A)
	Commit.

Here both read & write both are not allowed until commit happens.



Now coming back to B+ trees :-

Two different scientist gave two separate definitions of B+ trees :-

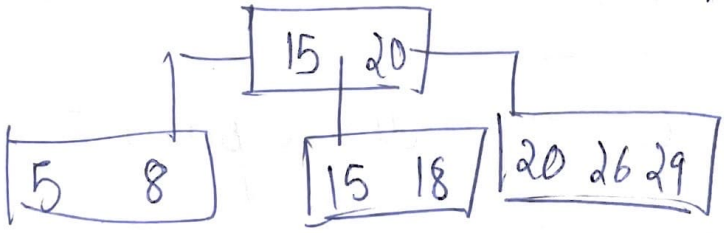
B+ tree of order 5
then internal node
→ 2 and 4
and leaf node
→ 3 & 5

B+ tree of order = 5
here Internal & leaf node will have same min^m & max^m no. of keys = 2 and 4

deletion in B+ tree :- almost same as that of B tree deletion.

1. after deletion if no violation of min^m keys, then no changes in tree.
2. if violation of min^m keys, then borrow key from sibling.
3. if borrowing from sibling not possible, then merge the Node with sibling. Either update the Anchor key or pull down the anchor key from parent.

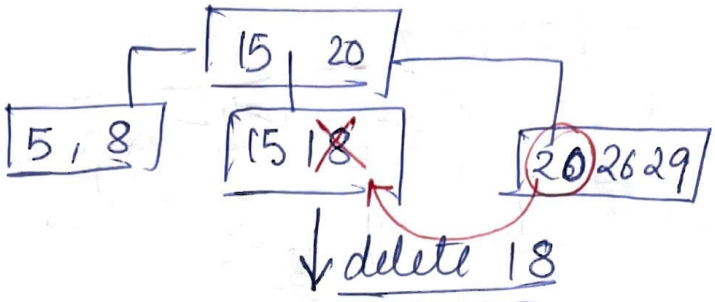
Ex ① order = 5 and min^m keys = 2, max^m keys = 4 } for all non-root nodes



delete 26 → 20 29

Yes we have min^m no. of keys

Ex ②

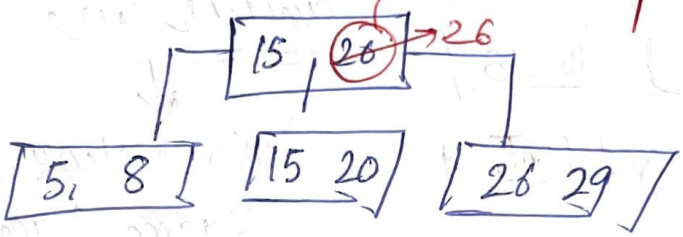


↓ delete 18

[15] = min^m no. of keys is violated

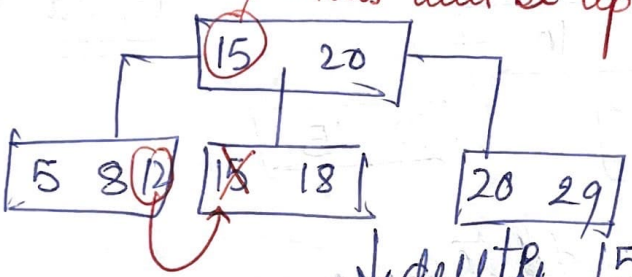
→ so from right sibling we can take.

now update this to 26.

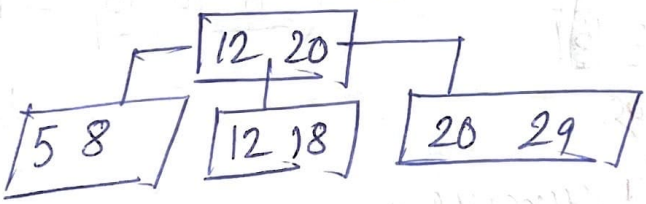


Ex ③

this will be updated to 12

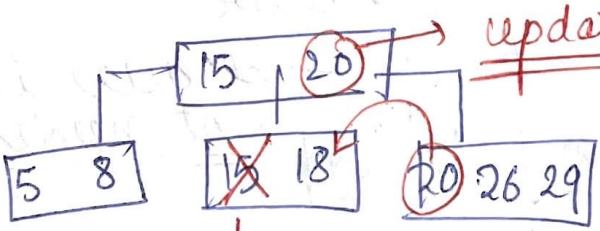


↓ delete 15



Ex ④

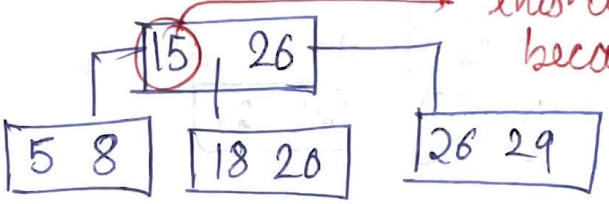
update to 26



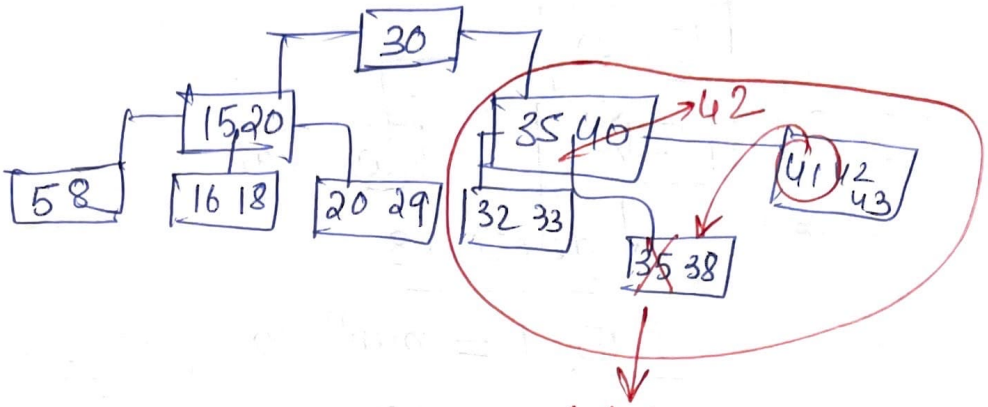
↓ delete 15

So here we can shift 20

this will not be changed because an anchor key is used only for direction purpose

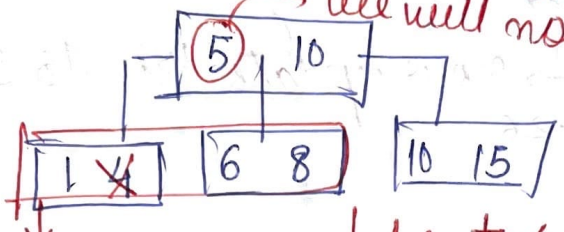


Ex (5)



delete 35

Ex 6-

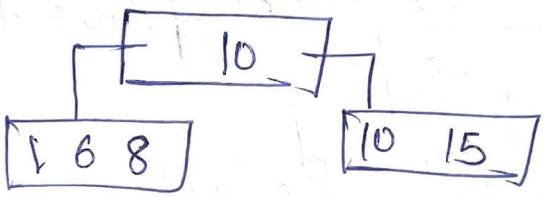


we will not pull down this anchor here

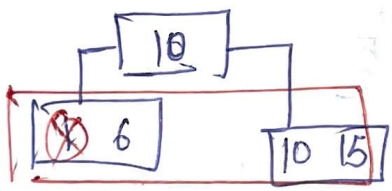
merge these two nodes

delete 4

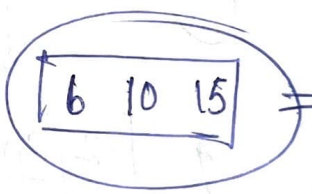
w/o anchor also kaam ban raha hai



Ex (7)



delete 1.

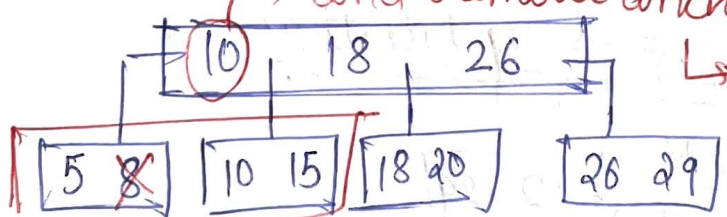


this only thing left

then merge

and remove anchor

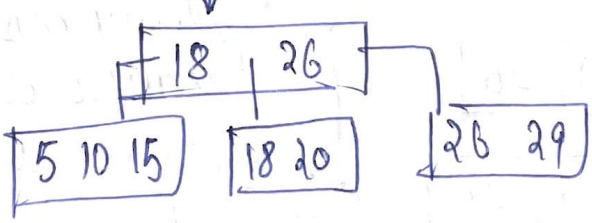
Ex 8.



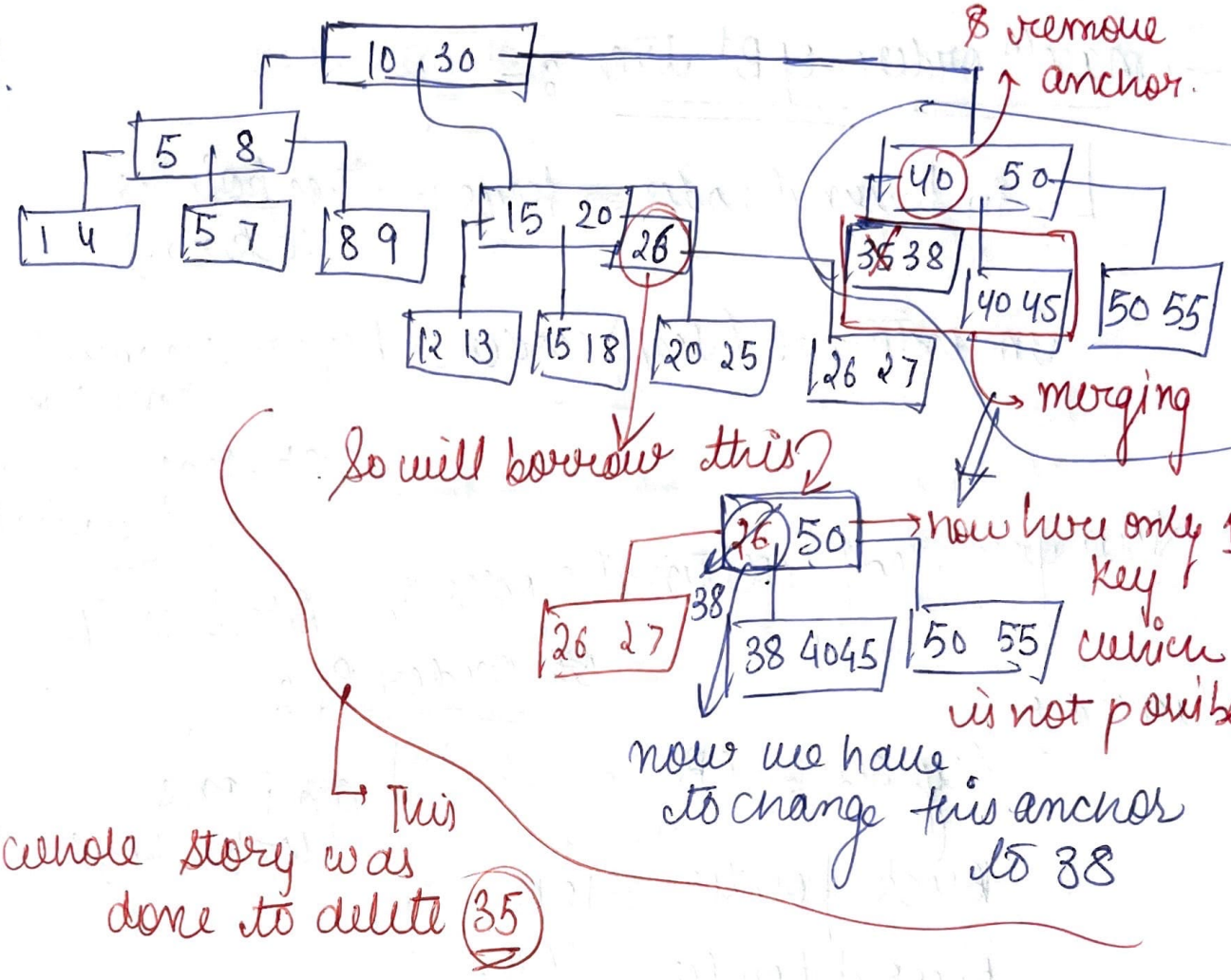
↳ because it is not needed

merge these two

delete 8

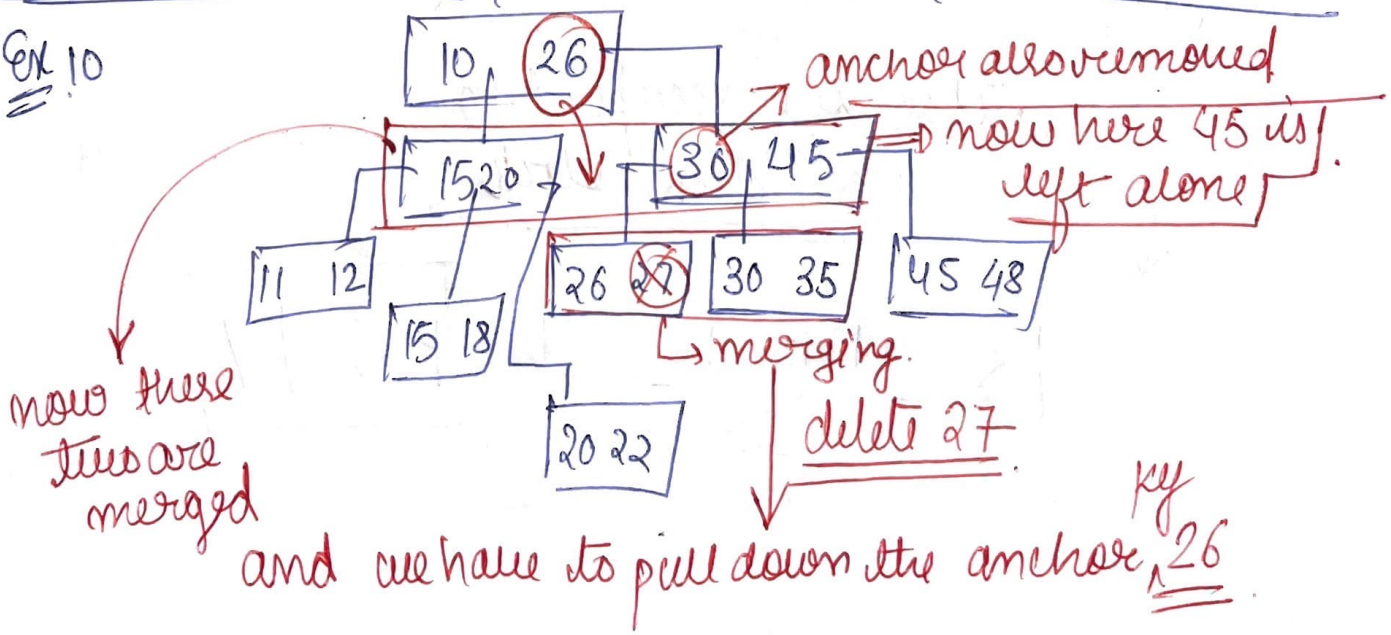


Ex 9



→ now one good approach will be :- Shifting 26 with a rotation with parent 30 just like we did in B tree.

Ex 10



→ max^m order of B+ tree :-

↳ in Internal nodes = keys + tree pointer (block no.)

in external (leaf) nodes = keys + record pointer + next leaf node pointer

Now for leaf/external nodes in B+ tree of order p :-

Consider Key size = 6B
Block pointer = 12B
Record pointer = 16B.
and one block size = 1024 B

max^m keys = (p-1) at each leaf node.

$$\underbrace{(p-1)\times 6 + (p-1)\times 16 + 12}_{\text{total size occupied in one disk block}} \leq 1024$$

$$p \leq \frac{1012+16+6}{22} \Rightarrow \boxed{p \leq 47}$$

04/10/2022.

Lesson 16#

#Relational Calculus

↳ also the mathematical interpretation of the query language.

two types we have

Tuple relational calculus (TRC)

domain relational calculus (DRC)

tuple relational Calculus :- A Non procedural query language

↳ TRC and DRC are more powerful than relational algebra & SQL

↳ predicate/condn

= { t | P(t) }

mention row/tuple/record that you want.

↳ tuple variable

↳ Here we will specify the condn

This is the very Basic way to write a TRC

Now Consider the student table = {RNO., Name, DOB, marks}

now if you want only the RNO. → a specific attribute, then you have to mention :- t[RNO.] or t.RNO.

t[attribute1] or t.attribute1

$\{ t \mid P(t) \}$

t → Select of SQL

$P(t)$ → from, where clause of SQL → after the straight line

Ex now consider the table :-

Shopkeeper { F_name, L_name, rating }

Now ① $\{ t \mid t \in$ shopkeeper $\}$ = This is actually only distinct ⟺ Select * from shopkeeper

② $\{ t \mid t \in$ Shopkeeper ~~AND~~ $\wedge$ $t.rating > 8 \}$

Symbols of predicate logic will be used

or

$t[rating] > 8$

* TRC eliminates the duplicates.

→ means will give only the distinct values

Now consider the following :-

→ $\{ t.FName, t.LName \mid t \in$ shopkeeper $\wedge$ $t.rating > 8 \}$

$\{ t[FName], t[LName] \mid t \in$ shopkeeper $\wedge$ $t[rating] > 8 \}$

These both queries are same

Now Select FName, LName from Students where gender = 'Male' and marks < 20

from students (RNo., FName, LName, gender, marks)

→ so the reqd query will be :-

$\{t.FName, t.LName \mid t \in Students \wedge t.gender = 'male' \wedge t.marks < 20\}$

Now find all such students who's marks are greater than 40.

Soln. $\{t \mid t \in Students \wedge t.marks > 40\}$

* if you access any specific attribute or any tuple which is not present in the table :- then that query will give you NULL !

now using the Quantifiers :-

→ Existential :- $\boxed{\exists t \in r\,(P(t))}$ condition.

↳ if there is atleast one tuple which belongs or satisfies the given/mentioned cond^n → then that tuple will be in result set.

→ Universal :- $\boxed{\forall t \in r\,(P(t))}$

↳ here all the tuples should satisfy the mentioned condition.

Tuple Variables :- 1. Free :- if you are writing w/o using quantifiers, then those are free variables.

2. Bound :- tuples bounded by Quantifiers.

Ex: Students (FName, LName, DOB, Marks, Gender, DNo.)

Dept (DNo., DName, HOD)

Now find all such female students who studies in CSE dept.

Soln.

Then our query :- here we have to join the table.

↳ by using the existential quantifier.

↳ $\{t \mid t \in Students \wedge t.gender = 'F' \wedge \exists s \{ s \in dept. \wedge t.dNo. = s.dNo \wedge s.dname = 'CSE' \}\}$

↳ this will be our reqd query

Now find all such department names where there is no any male student?

Soln So here :- do the negation of :-

find all such dName where Male Student is there

↳ Negation of this will give us the reqd thing ↳ (~)

$\{ d.Dname \mid d \in dept \wedge \nexists S \{ S \in Students \wedge S.gender = 'm' \wedge d.DNo. = S.dNO \} \}$

Ex consider :- Drivers (did, dName, rating)
Cars (cid, cmodel, cColour) = C.
drives (Did, cid, date) = d

Now find all Models of cars driven by 'Micheal'.

Soln. $\{ c.model \mid c \in Cars \wedge (\exists t)(\exists d) \{ t \in drivers \wedge d \in drives \wedge t.did = d.did \wedge d.cid = c.cid \wedge t.dname = 'Micheal' \} \}$

* Here you can write there exist in Nested manner also.

↳ $\{ c.cmodel \mid c \in Cars \wedge (\exists d) \{ d \in drives \wedge c.cid = d.cid \wedge (\exists t) \{ t \in drives \wedge t.did = d.did \wedge t.dName = 'Micheal' \} \} \}$

domain relational calculus :-

↳ Here Rather than writing a tuple Variable, now we will mention here column Names.

$\{ \langle C_1, C_2, C_3, \ldots, C_n \rangle \mid P(C_1, C_2, C_3 \ldots, C_n) \}$ → Predicate or cond's

where $C_1, C_2, C_3 \ldots C_n$ = are domain Variables

Ex Student (rNO, FName, marks)

then the query :- {<rNO, FName, marks> |

<rNO, FName, marks> ∈ Students}

male

Ex Select Names of all those students who have scored less than 50 marks.

Soln {<Name> | <r, n, m, g> ∈ Students ∧ <g> = 'm' ∧ <m> < 50 }

domain variables.

r = for roll No.

n = for name g = gender

m = marks

Ex Select name & marks of student with roll NO. = 5.

Soln {<n, m> | <r, n, m, g> ∈ students ∧ <r> = 5}

we can write above query as :-

{<n, m> | <5, n, m, g> ∈ Students}

sequence matters here

Here writing directly the value of r

→ this will work only for equal to