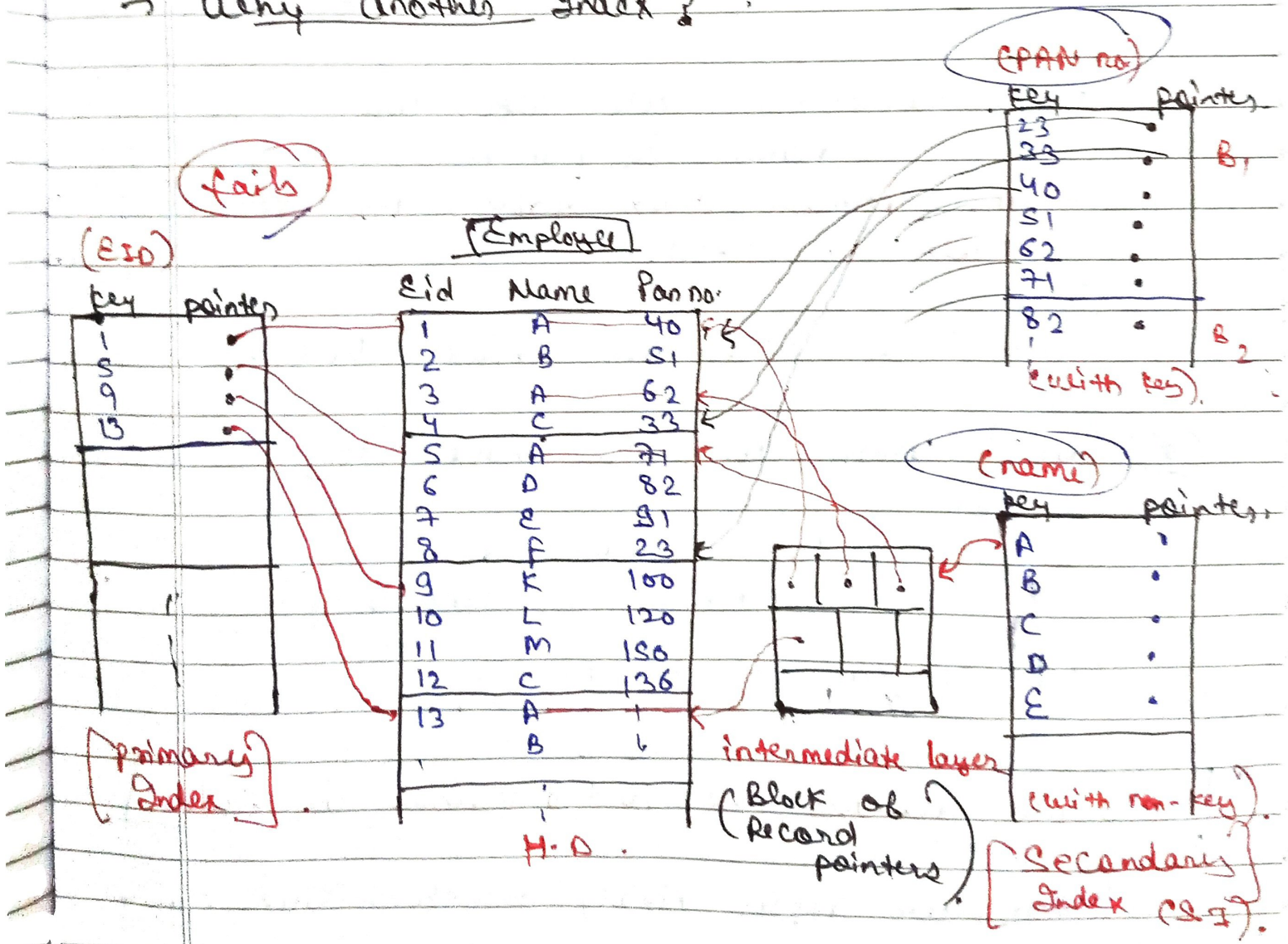
(17.) Secondary Index (S.I.), (Multilevel Indexing):

⇒ (Already there is an Index in database, we have to made another Index).

→ Where use S.I. ?

Unordered Data,
↙ ↘
key Non key.

→ Why another Index ?



→ Why Another Index?

2) Let, H.D. has data of Employee table & data is already sorted on basis of Eid (primary key). → [Unique + Not Null]

↳ bcz most of the query are on E-Id.

↓

(2) Find * of Employee where E-Id = ...)

↳ then we use primary Index.
(which is already there, formed on basis of E-Id.)

But, sometimes we also use name & pan no.
then primary Index fails bcz it is made on E-Id.

So,
we use Secondary Index here.

(*) Can I when we also have key.

then
we use PAN as
(unique + unordered)
(As u can see in Diag.)

(#) We always use sorted values in Index.

↳ then, we apply binary search → Time Saving.

* Index is always sorted & unique.

* 2) Secondary Index on key, is always DENSE.

bcz, we don't have any anchor (leader) here like in primary In.

→ These values are not sorted. Hence, we put them sorted in Index & write all the records of H.D. in Index Table. So, Dense.

i.e.

No. of records in Index = No. of records in H.D.

⇒ Search time: $\log_2 N + 1$

where,

N is the no. of blocks in Index Table.

(3) Case II When we have Non key with unordered data.

It is the worst case.

then,

we use NAME,

↳ (neither ordered, nor key) (unique).

i.e. (Secondary Ind in D.bg.)

⇒ In Index, we don't have to take multiple values of A... (only once).

But, A is many times in Hard Dsk.

So,

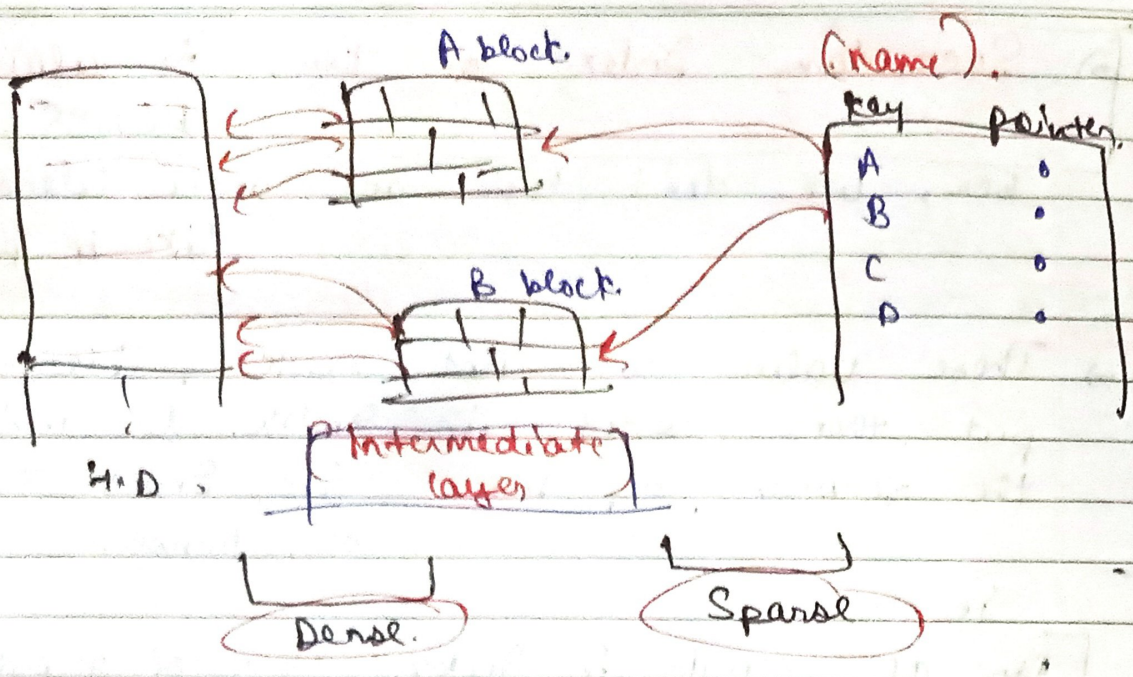
here we made an Intermediate layer.

(It is a block of Record pointers).

book में पहले secondary Index (S.I.) मिल जाता है।

Date / /
Page No.

Ex!,→



⇒ Hence, It is Mix of Dense & Sparse.

So,

⇒ We can say it DENSE Overall.

⇒ Time Complexity : $\log_2 N + 1 + 1 + \dots$

(extra for intermediate layer)

(And this is only for 'A'. It may be more than $\lceil \log_2 N + 2 \rceil$)

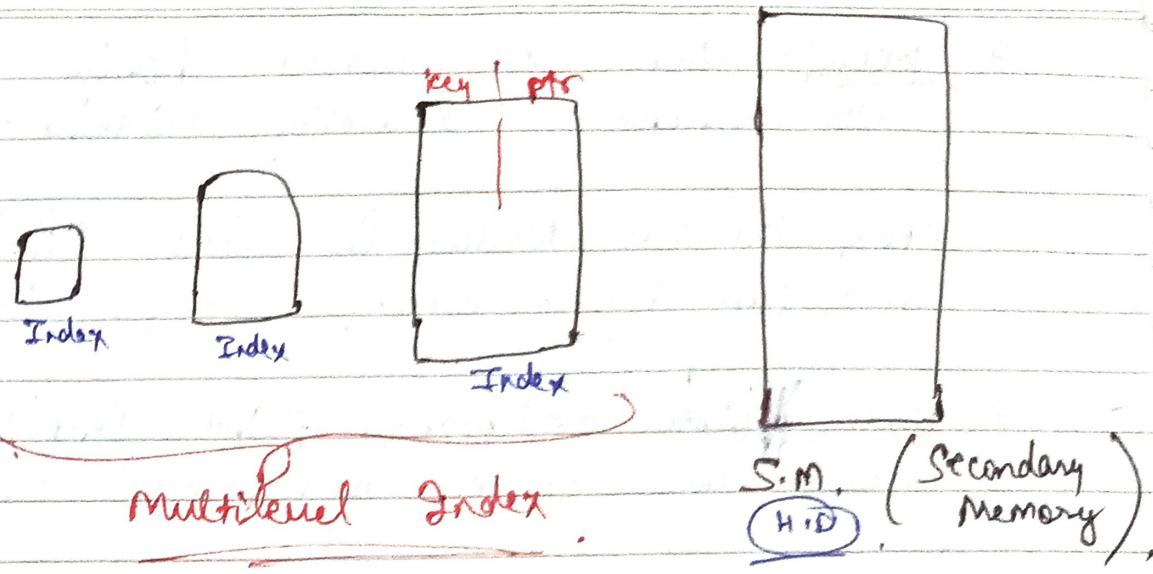
(*) Secondary Index is always dense.

(9th) Intro to B-Tree & its Structure :→

→ B- Tree (Dynamic Multilevel Index) :-

(Balanced Tree)

graph → cycle
Tree → acyclic.

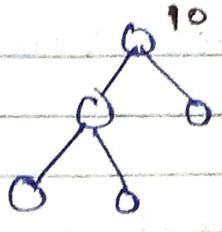


Multilevel Index

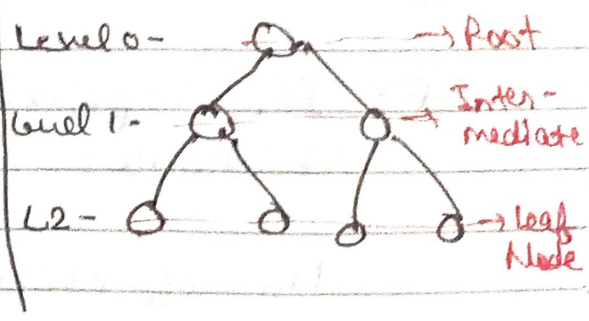
⇒ In this, it is hard to manage. bcz, If we insert data in S.M. then we have also insert in all the Indexes & same for delete...

(#) Balanced Tree (B-Tree!→)

→ means, all the Elements are at the same level of leaf Node.

⇒ 

⇒ Not Balanced.



⇒ Block pointer (B.P.) or Tree pointer! →
When node denotes his child. Then, we use Block pointers (B.P.).

⇒ Note! → A node can contain multiple values here.

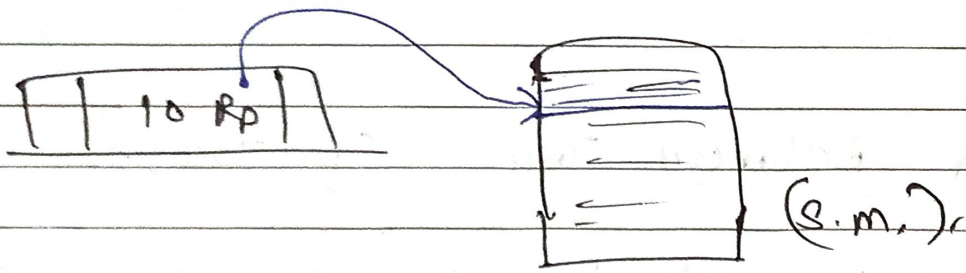
Date / /
Page No.

⇒ **Keys**:- that on which basis we have to search. Searching Criteria - key.

(key को इन Nodes के अंदर रखेंगे।)
(We can insert multiple keys in a node).

⇒ **Data pointer (or) Record pointer (R.P.)**:→ (D.P.)
These are with correspond to keys.

→ This record pointers points to in the Secondary Memory (Hard Disk) where are record is present of that key.



(#) keys = Record pointers.

(#) Block pointer depends on the how many childrens are there in a Node.

No. of children = No. of B.P.

(#) Order = p (of a B-tree)
: Max. no. of Block pointers.

Order = p = Max no of children a node can have.

(#) Keys = p-1 (Max)
Rp = p-1

Min keys = $\lceil \frac{p}{2} \rceil - 1$

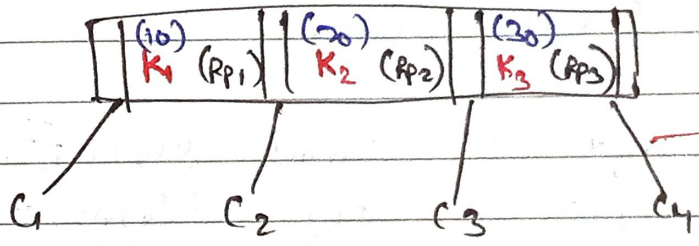
Table :-

Children		Root	Intermediate Node.
	max	p	p
	Min	2	$\lceil \frac{p}{2} \rceil$

↳ ceiling value

Ex :-

Let $p = 4$ = order of a Tree.



(#) We always insert keys in the sorted Order. (bcz, it follows the prop. of Binary Search Tree).

---

99. Insertion in B-Tree :-→

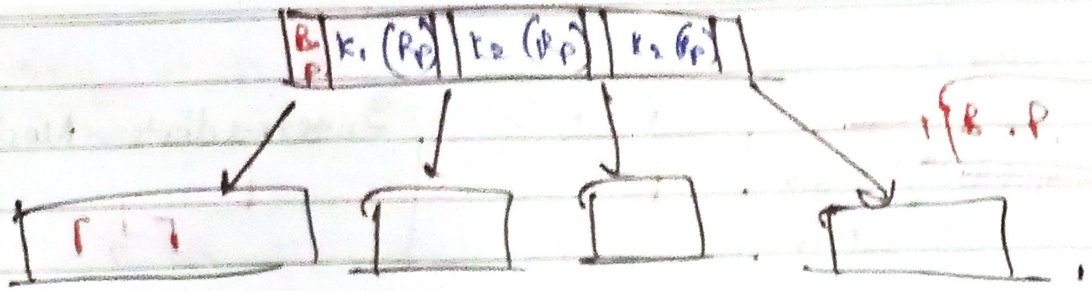
Q:- Insert the following keys into B-Tree, if order of B-tree = 4.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

⇒ Order = 4 = max. no of children.

Max keys = $p - 1 = 3$

Min keys = $\lceil \frac{p}{2} \rceil - 1 \Rightarrow 1$



⇒ We follows the prop of Binary Search Tree in Insertion.

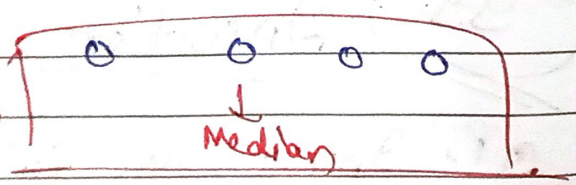
(#) In Binary Search Tree,

Root → Left = Small Element
Root → Right = Big Element.

(#) Overflow होते ही, Median किसी तरह break करना पड़ेगा,

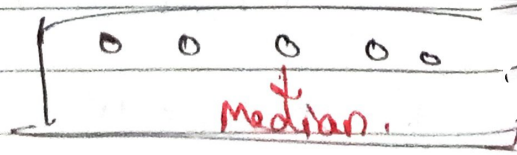
→ When, n - Even ⇒ $\frac{n}{2}$ ⇒

Let, 4 ⇒ $\frac{4}{2}$ ⇒ (2)



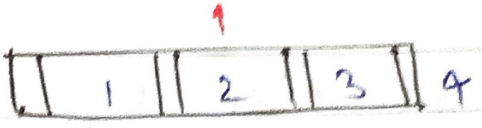
→ When, n - odd ⇒ $\frac{n+1}{2}$ = Median

Let, 5 ⇒ $\frac{6}{2}$ ⇒ (3)

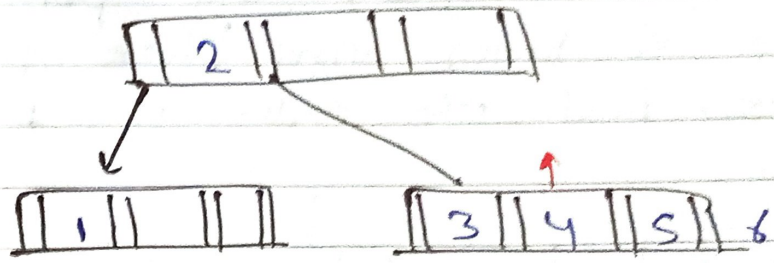


⇒ and, Shift Median to upward (↑), & उसके Left & Right Elements उसमें नीचे आ जाएंगे। & ये, how we break.

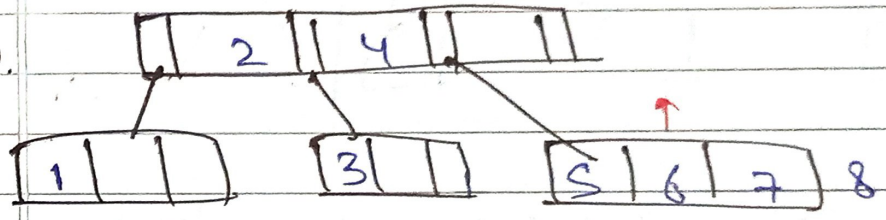
(Ans), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

a) 

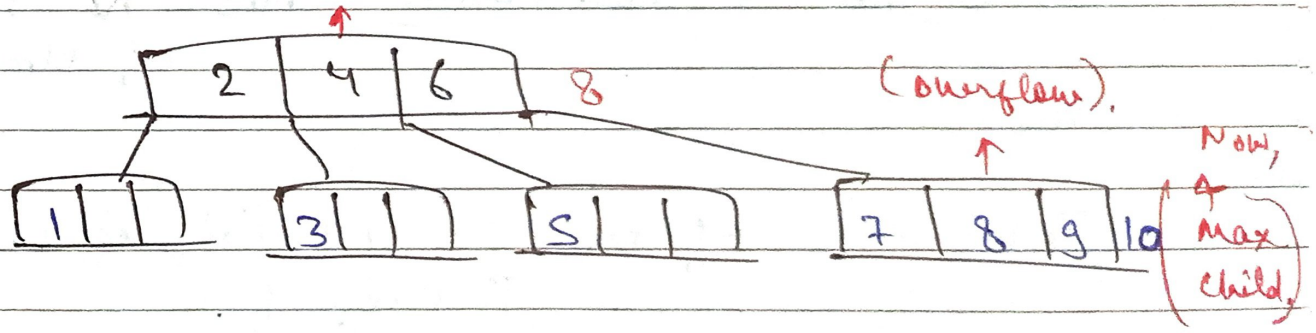
(overflow). ⇒ $\frac{n}{2} \Rightarrow \frac{4}{2} \Rightarrow 2$

⇒ 

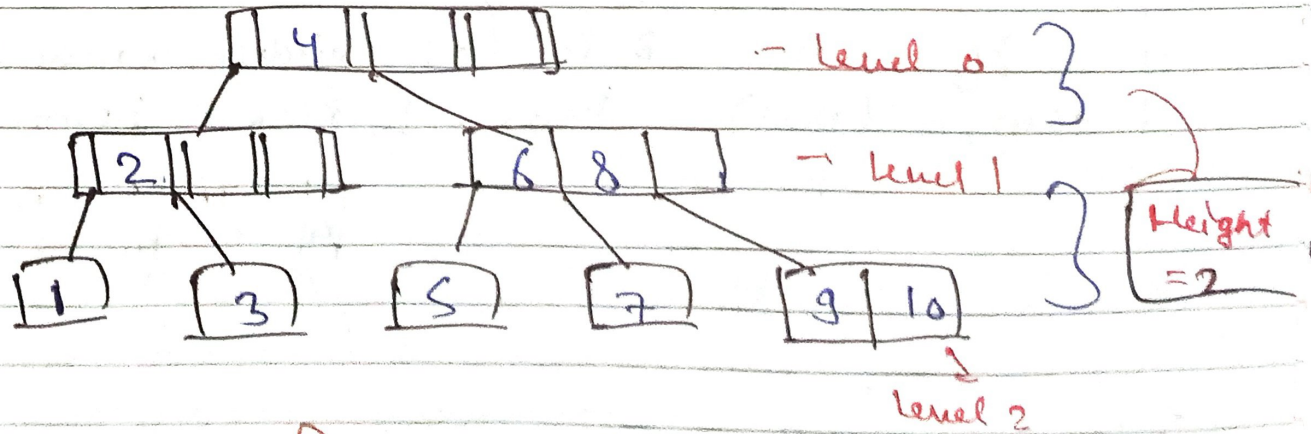
(overflow).

c) 

(overflow).

⇒ 

(overflow).

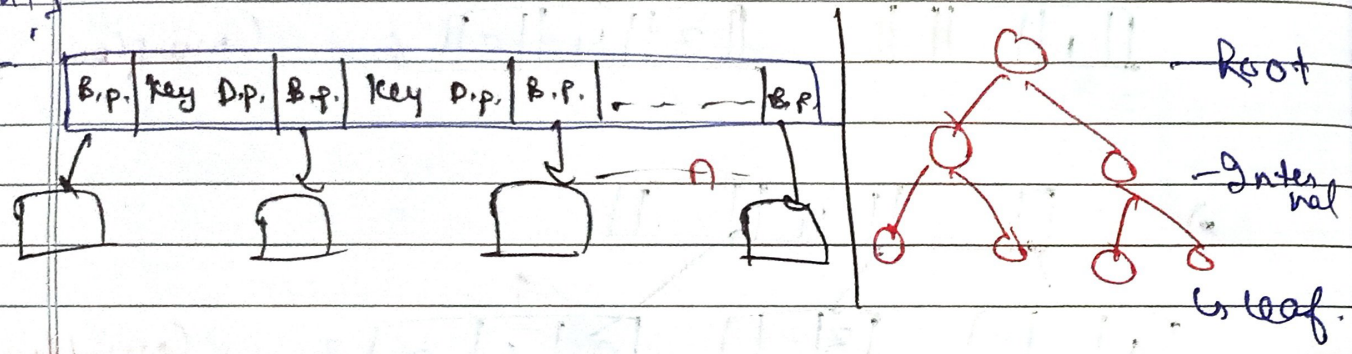
⇒ 

⇒ height - 2
levels - 3

100. How to find Order of B-Tree :→

Q:- Consider a B-tree with key size = 10 bytes, block size 512 bytes, data pointer is of size 8 bytes and block pointer is 5 bytes. Find the order of B-tree?

Sol^n :→



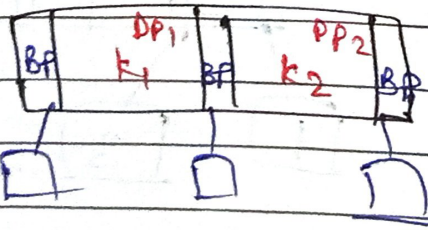
→ Let, In a node / Block has 'n' no. of Block pointers.

So,

Total size of B.P. = n × (size of 1 B.P.)

= n.Bp

→ If n B.Ps (or children a node can have) then, (n-1) keys, & Record pointer.



$n \times Bp + (n-1)$ key size $+ (n-1) Rp \leq$ Block size (or) Node size.

2). $n \times 5 + (n-1)(10+8) \leq 512$.

$\Rightarrow 5n + 18n - 18 \leq 512$

$\Rightarrow 23n \leq 530$

$\Rightarrow \boxed{n \leq \frac{530}{23}}$.

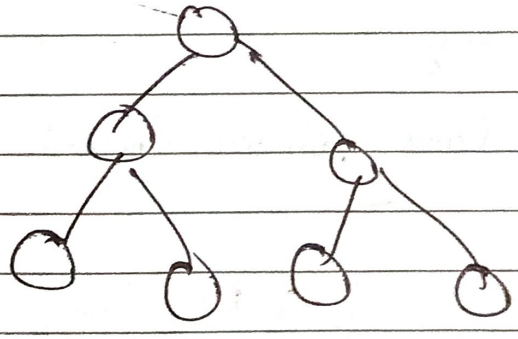
$\Rightarrow n \leq 23.04$

$\therefore$, $n = 23$ ✓. — Max. 23 children.

$\Rightarrow$ Every B.P. represent a children.

SO, Max. Order = 23. ✓

$\Rightarrow$



Max	Min children
23	2
23	$\lceil \frac{23}{2} \rceil \Rightarrow \lceil 11.5 \rceil$ ✗ 12.
Leaf has no children.	

✓

- Keys = Order - 1 ⇒ 22 ✓.

✗ ✗

101. DIB — B-Tree & B+ Tree :→

* B-Tree :→

1) Data is stored in leaf as well as internal nodes.
2) Searching is slower, deletion complex.

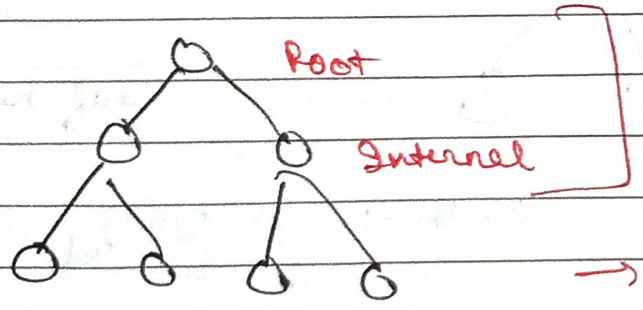
3.) No Redundant (Duplicate) search key present.
4.) Leaf Nodes not linked together.

B+ Tree :-

1.) Data is stored only in leaf Nodes.
2.) Searching is faster, deletion easy. (directly from leaf Node).
3.) Redundant keys may present.
4.) Linked together like linked list.

We use B & B+ Tree, to put Index Record. These trees actually contain Index Record.

Index Record → key
Data pointer (Record pointer).



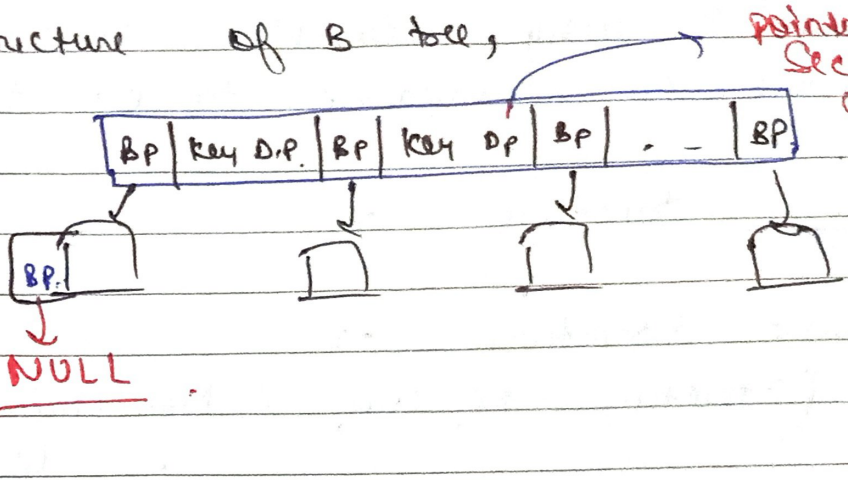
In B tree, the structure of every node is same. (Either root, internal or leaf).

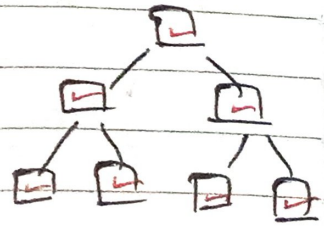
leaf Node, has no children. So, what's the role of B.P.. Then, they points to NULL.

B.P - Block pointer.

Date
Page No. 51

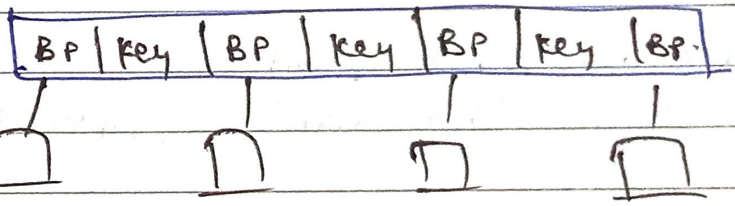
Structure of B tree,





Structure of B+ tree :→

⇒ Non leaf structure :→ or Internal Node →.



↳ There is no Data pointer (D.P.) in Internal Node structure / Non-leaf Node structure.

R.P / D.P ~~X~~

Hence,

We have more space in internal node. Hence, we can create more children & put more keys.

So, that's why,

B+ tree → Breathwise longer.

B tree → Depthwise longer.

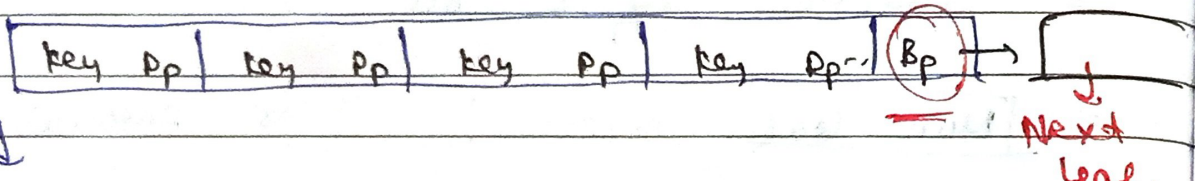
↳ (for less no. of children breathwise as compared to the B+ tree).

So, depth more.

#. ∞, that's why searching is slower in B tree as compared to B^+ tree.

#. B^+ tree Structure →

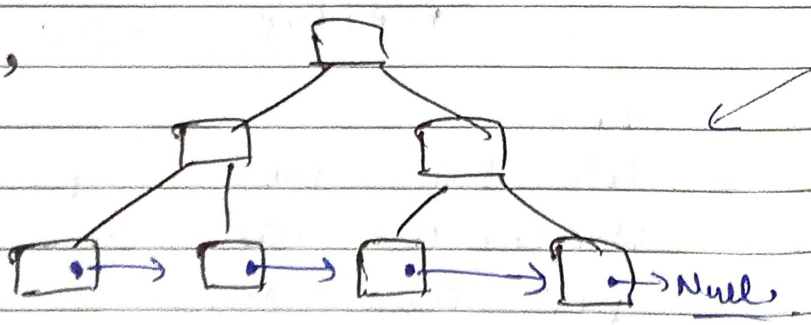
Leaf Node Structure →
(Structure of leaf & Non leaf Nodes are different.)



⇒ (There are no Block pointers in leaf Node Except 1 in last which points to Next leaf.)

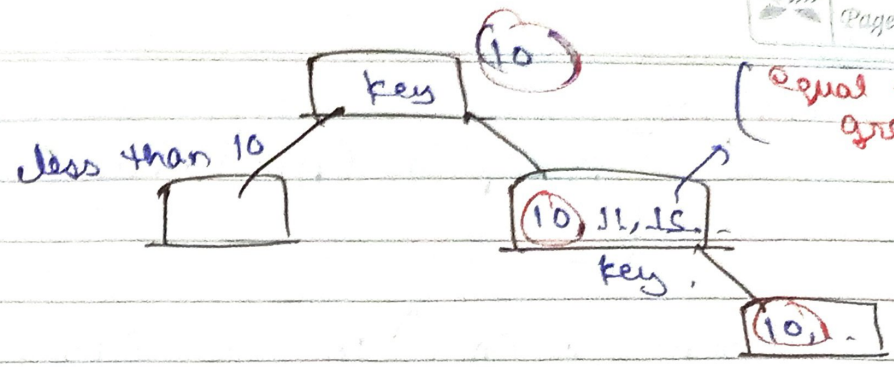
Both, B & B^+ are balanced, i.e, all leaf Nodes are at same level.

In B^+ structure,



⇒ In this, less value to key is in left node & the greater value to key are in Right node. (But here, in Right side, we also have to put the key value with its greater values).

⇒ i.e,



⇒ Reason :→ bcz, we only search in leaf Node, bcz only leaf Node has all keys with Data pointers (D.P.) are present.
So,
(We also have to reach the D.P. of the key)
so, that's why we also carry the key value upto leaf Node.

→ Searching is that's why faster in B+ tree, bcz have to search only in leaf Node.

&
bcz of this, Redundant keys are also present in B+ tree.
Leaf Nodes are also linked together.

---

102. Qus on Order of B+ Tree :→

Q!→ Consider a B+ Tree with key size = 10 bytes, block size = 512 bytes, data pointer = 8 bytes & block pointer = 5 bytes. What is the order of leaf & non leaf node ?

Sol!→ Non leaf :→

BP	key	BP	key	BP	key	BP

Leaf Node :→

key DP	key DP	key DP	B.P. →

Date / /
Page No.

⇒ **Non-leaf :-**

$$n \times BP + (n-1) \times key \leq \text{Block size}$$

→ $5 \times n + (n+1) 10 \leq 512$

$5n + 10n + 10 \leq 512$

$15n \leq 522$

$n \leq \frac{522}{15}$ 34.8

$n \leq 34.8$

$\therefore$ $n = 34$ ✓

Order = 34

(Max BP.) or (Max children possible).

⇒ **Leaf :-**

Let x pairs

$$x(key + PP) + BP \leq \text{Block size}$$

$x(10+8) + 5 \leq 512$

$18x \leq 507$

$x \leq \frac{507}{18}$ 28.18

$x \leq 28.18$

$\therefore$ $x = 28$ ✓ Order = 28 ✓

Note :- Order of a leaf node in B+ tree is the no. of (key, D.P.) pairs.

H.D.D. - Hard Disk Drive.

103. Immediate Database Modification :→
(Log Based Recovery Methods).

⇒ ↳ Immediate means तुरंत, उसी समय।

Ex-1

A = ~~100~~ 200
B = ~~200~~ 400.

Hard Drive.

T_1
R(A)
A = A+100
W(A) −200
R(B)
B = B+200
W(B) −400.
Commit.

Transaction Log.

< T_1, Start >

< T_1, A, 100, 200 > (old, new)

< T_1, B, 200, 400 > (old, new)

< T_1, Commit >

Redo

⇒ जैसे ही Ram के अंदर value (200) हुई, उसी time पे ही Database में भी (A=200) कर दो। बिना Commit हुए।

⇒ i.e, At time, when we write in memory (RAM), At same time also update in the H.D. We don't wait for commit.

⇒ But, when we see in Trans. Log,

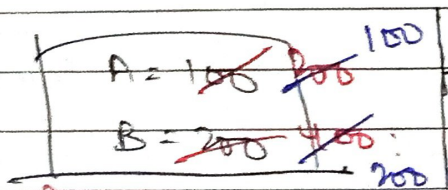
Our Recovery Manager sees the log & check whether T_i has both Start & commit or not. & If yes, then it REDO

→ Redo, means saves the latest value in the Database. i.e.,

→ (Recovery Manager don't see value in the H.D., it only sees in the Trans. log & check (starts & Commit) & then saves in the H.D. database. & if already saved, then Over-write & fixed them.

→ But, If

Ex:-1



T_1
R(A)
A = A + 100
W(A) - 200
R(B)
B = B + 200
W(B) - 400
↓
fail.

Transac^n log.

<T1, start>

<T1, A, 100, 200>
old new

<T1, B, 200, 400>.

UNDO

→ Here, Recovery Manager don't find Commit in Trans. log. So, he UNDO.

→ UNDO, means it saves the old value. But, In database there are updated values now. So, Recovery Manager takes the old values from the Trans. log & saved them in the Database.

In Immediate, we store both old & new value but,
In Deferred, we only store new value.
(i.e. only REDO, not UNDO).

that's why
Immediate Database Modification is known as UNDO - REDO strategy.

Ex:- ⇒ Transac^n log ⇒

< T_1, start >
< T_1, A, 1000, 2000 >
< T_1, B, 5000, 6000 >
< T_1, Commit >
— REDO.

< T_2, start >
< T_2, C, 700, 800 >
old.
— UNDO.

---

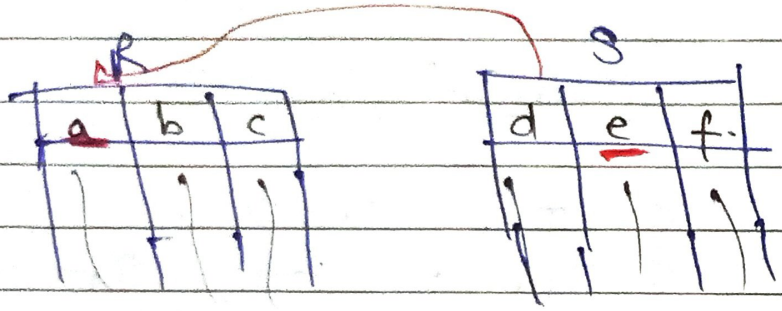
104. Ques^n on DBMS basic Concepts & Data Modelling :-

Q:- Let, R (a, b, c) and S (d, e, f) be 2 Rel^n. 'd' is foreign key of S that refers to primary key of R. Consider 4 operⁿ on 'R' & 'S'

i) insert into R
ii) insert into S
iii) delete from R
iv) delete from S

→ Which of these can violate Referential Integrity?

a) i & ii
b) i & iii
c) ii & iii → Ans.
d) i & iv



⇒ Understand from: → कि Intelligent student कुछ भी delete कर सकता है लेकिन कि इससे बच्चों को भी तो Delete करना पड़ेगा।
(If they don't delete → then, violation.)

2. The view of total database content is —

a) Conceptual view Ans. b) Internal view
c) External view d) physical view.

⇒ This is On 3 schema architecture.

* Internal view & External view,
external view is basically related to outer view.

⇒ User को Total view नहीं होता।
User has partial view (data only he need)