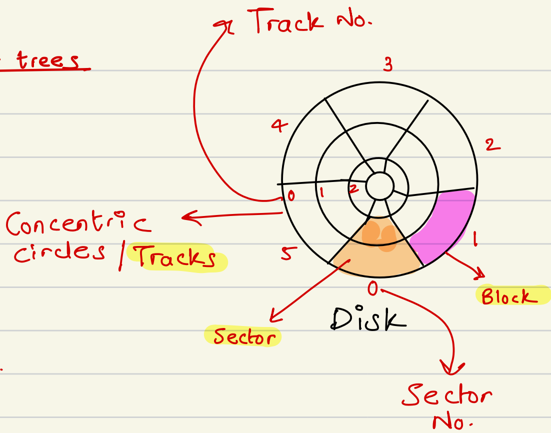
(From Scratch)

B-trees and B+ trees



* Disk Structure :

- Block addr : Track No + Sector No.
- Block size : 4 KB
- Disk head rotates and points
- to blocks .
- Disk head rotates on a spindle.

* How is data stored on disk?

Ans. : Consider a database shown alongside :

eid	name	dept.	sect.	addr.
0	A	. . .	. . .	. . .
1	B	. . .	. . .	. . .
2	C	. . .	. . .	. . .
3	D	. . .	. . .	. . .

Each row is k/a record .

Size of each record : (in bytes)

eid = 10

name = 50

dept = 10

section = 8

addr = 50

128 bytes

Block size = 4 KB = 4096 bytes

No. of records / Block = $\frac{4096}{128} = \boxed{32}$ *

Let us say, we have 1024 records .

So we need $\frac{1024}{32} = \boxed{32 \text{ blocks}}$ to store data.

* So, we create an index which stores eid and pointer to that record on the disk.

eid	pointer
1	200H
2	201H
3	202H
4	203H
5	.
.	.

* The informaⁿ about this index is also stored on the disk.

Let us say pointer takes 6 bytes of storage.

So 1 entry in the index takes (eid + pointer)

$= 10 + 6 = \boxed{16 \text{ bytes}}$

records/block: $\frac{4096}{16} = 256$

And we have 1024 records, so $16 \times 1024 = \boxed{16,384 \text{ bytes}}$

$\Rightarrow$ it needs $\frac{16384}{4096} = \boxed{4 \text{ blocks}}$ on disk.

Now, lets say we want to access record with eid = 512

Case 1: Neglect index

If we didn't have index, we would need to access,

N^{th} record $\leftarrow$ $\frac{512}{32} = 16$ blocks

records/block $\leftarrow$ 32

Case 2: Using index

If we use index,

for accessing 512 from index table,

we need $\frac{512}{256} = 2$ blocks, and once you have the pointer to the record, you can direclty access it i.e. 1 block.

index records/block $\leftarrow$ 256

So, you need a total of $2 + 1 = \underline{3 \text{ blocks}}$

$\therefore$ Indexing is used for fast retrieval of data.

If dataset increases, the index table size increases, so again storage on disk increases.

Soln:

Create an index of index which stores block address of index table.

In our case,

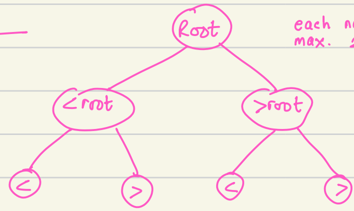
256 index table records are stored on each block.

So, we can create an index:

eid	Block addr
1	.....
257	....
513	....
⋮	....

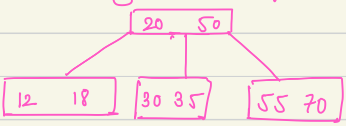
* M- Way search tree :

① BST



each node can have max. 2 children

② extending this concept,



Here : No. of keys = 2

Max. no. of children = 3

So, 3 - way search tree.

m = 3

③ 4 way search tree

means : 3 keys per node

4 children per node (max.)

Node Structure :

lc_1	K_1	rc_1	K_2	rc_2	K_3	rc_3

pointer to left child of K_1

pointer to right child of K_1 or left child of K_2

pointer to right child of K_2 or left child of K_3

Pointer to right child of K_3

④ For implementing index, we can use m-way ST, by just adding a record pointer.

lc_1	K_1	rp_1	c_1	K_2	rp_2	rc_2	K_3	rp_3	rc_3

record pointers

Problem with m-way search tree :

There is no rule to create new m-way search trees.

So, B - TREES !

⑤ B - Trees

+ for every node, fill half values.
- If degree = m
- No. of children = ceil (m/2)

eg. degree = 10

No. of children = 5 for every node.

when this cond^n is satisfied by every node (except root), ONLY THEN can we think of adding elements to half filled nodes!

+ For ROOT : Root can have min. 2 children
+ All left nodes at same level.
+ Bottom up insertion.

Degree : (m)
Max no. of children a node can have

Each node can have (m-1) keys per node

⑥ eg : m = degree = 4

Keys = 10, 20, 40, 50, 60, 70, 80

Step 1 : | 10 |

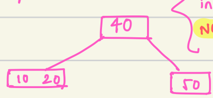
Step 2 : | 10 20 |

Step 3 : | 10 20 40 |

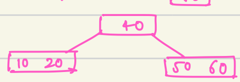
Make the largest element parent/root and add next element into right (rc).

NOTE : elements are sorted!

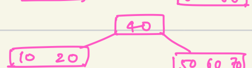
Step 4 : Split the node



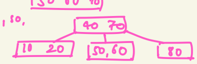
Step 5 :



Step 6 :

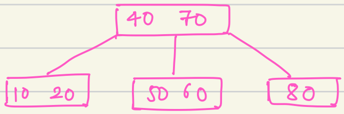


Step 7 : No. space for adding, so,

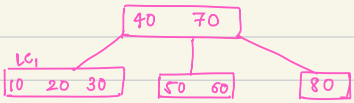


⑦ Adding more elements :

10, 20, 40, 50, 60, 70, 80, 30, 35, 5, 15



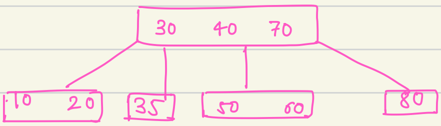
Step 8 :



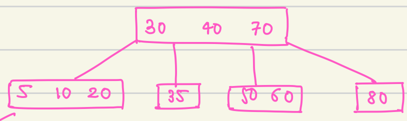
Step 9: Adding 35. It cannot be added to lc1

We will have to add it to rc to lc1

But root has one vacancy, so.

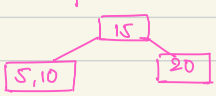


Step 10: Insert 5



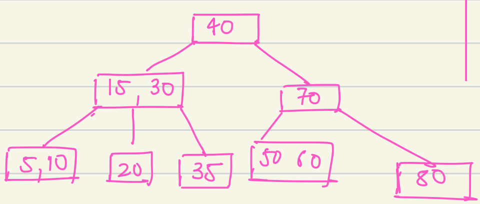
Step 11 : Insert 15. But no space, so split the node

5, 10, 15, 20



+ But again, there is no place to insert element in root.

+ So, split the root : 15, 30, 40, 70



⑧ Applying it with Index concept.

Each element in a node (key) will have a child pointer/s and a record pointer.

⑨ B+ tree

Similar to B-trees EXCEPT

only leaf nodes will have record pointers

In order to achieve this, we create a copy of parent key in the right child (usually) by changing the condⁿ for right key to ≥ parent key.