Database Management Systems

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Lecture Session-14

B+ Tree Indexing

innovate achieve lead

Content

- *What is Tree Indexing*
- *B+ tree*
- *Inserting and deleting keys into B+ Trees*
- *B Tree*
- *Constructing a B+ tree*
- *Designing a B+ Tree node structure*

Tree Indexing

innovate achieve lead

Adopting Tree structure for implementing indexes

A *tree* consists of *nodes* & *leaves*. The number of arcs from a node in the tree to root is known as *path-length*.

The *height* of non empty tree is equal to max.level of a node in a tree. For empty tree height is zero.

Binary tree

Each node has max two children (left and right). Hence at i^{th} level, no of nodes present are $2^{(i-1)}$ (root is at level 1).

Complete binary tree: All nodes except at last level are present.

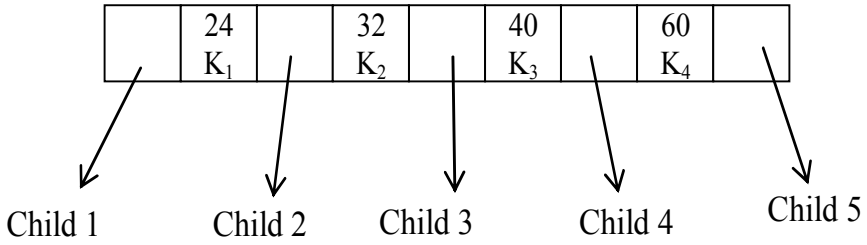
Binary Search Trees: for each node in the tree, all values stored in its left subtree are less than value stored in the node and all values stored in the right subtree are greater than the value in the node.

Multilevel Search Tree of order m

(or)

M-way search tree

- Each node has *m* children and (*m* -1) keys
- Keys in each node are in ascending order



No of children = (*m*) = 5

No of keys = (*m* -1) = (5-1) = 4

B+ Tree Indexing

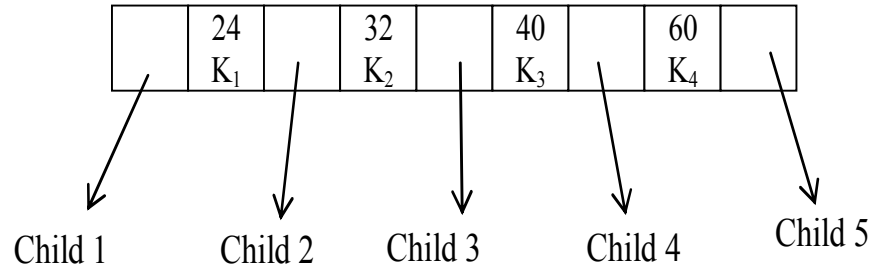
innovate achieve lead

B+ Tree is a multilevel search tree used to implement dynamic multilevel indexing. The primary disadvantage of implementing multilevel indexes is that the performance degrades as the file grows. It can be remedied by reorganization, but frequent reorganization is not advisable. B+ tree is best suited for multilevel indexing of files, because it is dynamic.

B+ Tree of Order p

It is a balanced tree, (all leaves are at same level).

Each internal node is of the form-



B+ Trees

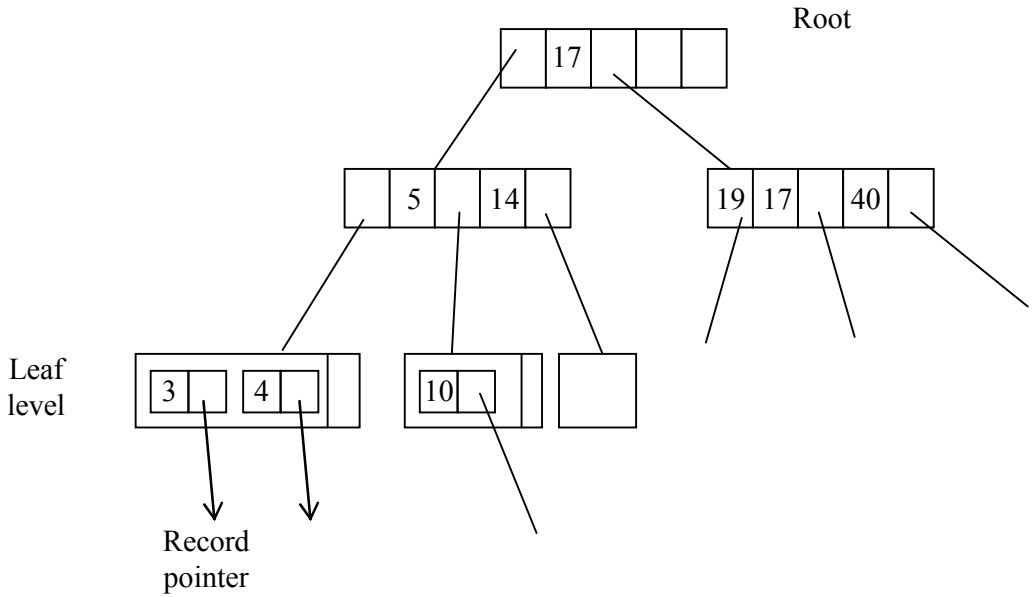
For a B+ tree of order p

- With in each internal node $K_1 < K_2$
- P_1, P_2… are tree pointers
- K_1, K_2, K_3,… are key values which are in ascending order from left to right
- Each internal node has at most *p* (order) pointers.
- Each internal node except the root node has at least *ceil (p/2)* tree pointer to next level. Root has at least 2 pointers.

- An internal node with *q* pointers has (*q* -1) field values.
- All record pointers are available at leaf node only.
- Once we get a key value at leaf node, from there accessing next value in sequence is easy because all keys at leaf level are in ascending order.

EX: B+ Tree of order 3 i.e., p = 3

Min. no. pointers in any node = $\left\lceil \frac{3}{2} \right\rceil$



B- Tree

Note

In a B+ tree record pointer for a record with given key can be found only at leaf node.

But if it is in case of B-tree it can happen at intermediate node also.

Hence in B+ tree search, success or failure can be declared only after reaching leaf_level.

Where as in B-tree search can be successful at intermediate level as well.

On failure we reach the leaf level.

Constructing a B+ Tree

Construct a B+ tree with given specifications. The order of the tree, p=3 and p_{leaf} =2. The tree should be such that all the keys in the subtree pointed by a pointer which is preceding the key must be equal to or less than the key value , and all the keys in the subtree pointed by a pointer which is succeeding the key must be greater than the key.

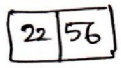
Insert the following keys in same order- 56, 22, 78, 42, 102, 90, 96, 35. Show how the tree will expand after each insertion, and the final tree.

Next, delete 56, 46, 22 in the same order and show the status of the tree after each deletion.

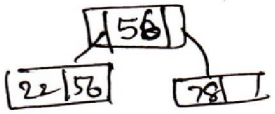
Q5 @ B+ Tree $p = 3$ $p_{leaf} = 2$

Keys - 56, 22, 78, 42, 102, 90, 96, 35

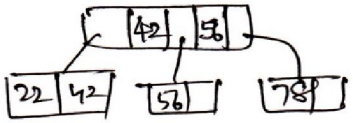
Insert 56, 22



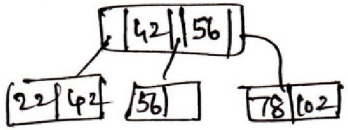
Insert 78



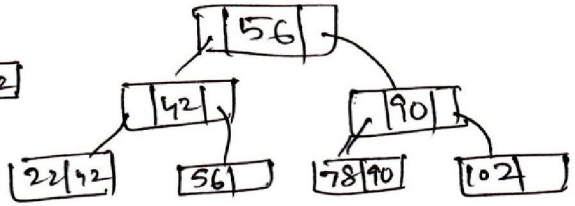
Insert 42



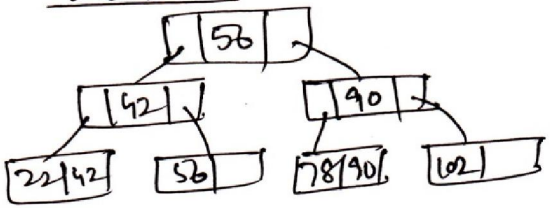
Insert 102



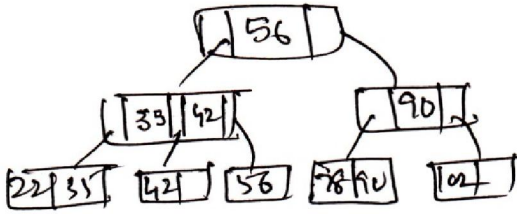
Insert 90



Insert 96

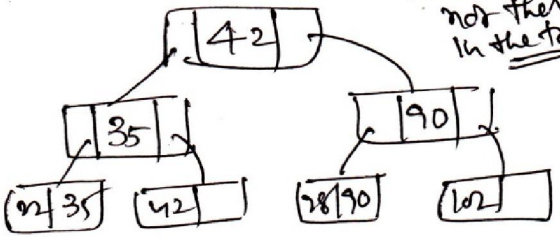


Insert 35



Final tree

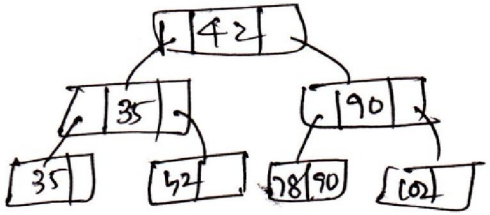
Delete 56



Delete 46

not there in the tree

Delete ~~42~~



final tree

Node design for B+ tree

We need to design a B+ tree indexing for Student relation, on student_id attribute; the key of the relation. The attribute student_id is of 4 bytes length. Other attributes are- student_age(4 bytes), student_name(20 bytes), student_address(40 bytes), student_branch(3 bytes). The Disk block size is 1024 Bytes. If the tree-pointer takes 4 bytes, for the above situation, design the best possible number of pointers per node(internal) of the above B+ tree. Each internal node is a disk block which contains search key values and pointers to subtrees.

Disk block size=1024 Bytes
Size of B+ tree node= size of disk block
Each tree pointer points to disk block and takes 4 Bytes.
Each key (student_id) takes 4 Bytes
In a B+ tree node, No. of pointers = no. keys +1
Assume that no. keys = n
Then no. pointers= n+1
Then min. size for a node= {(no.Keys* size of each key)+
(no.pointers * size of each pointer)} <= 1024
(n*4)+(n+1)*4 <=1024
4n+4n+4 <= 1024
8n+4 <= 1024
8n <= 1024-4= 1020
n <= 127.5 or 127
hence In each internal node, no. keys=127; and no. pointers=128

Summary

- *What is Tree Indexing*
- *B tree and B+ tree concepts*
- *Constructing a B+ tree (Insert/Delete operations)*
- *Designing a B+ Tree node structure*