ADITYA ENGINEERING COLLEGE (A)

DBMS

INDEXING

By

Dr. S Rama Sree
Professor in CSE & Dean(Academics)
Aditya Engineering College(A)
Surampalem.

CONTENTS

- Indexing Techniques
- File Organization and Indexing
  - Cluster Indexes
  - Primary and Secondary Indexes
- Index data Structures
  - Hash Based Indexing
  - Tree base Indexing
    - B Tree
    - ISAM
    - B+ Trees

Storage and Indexing

- Indexing is defined as a data structure technique which allows you to quickly retrieve records from a database file.
- Ex:- I want to search about primary key topic.

Then open textbook and backside of textbook index will be there and search for primary key.

Primary key is found with different page numbers.

Primary key --- 20,45

20 called as search key value

Structure of index

Search Key	Data Reference

- Search key contains copy of primary key or candidate key of a table.
- Data Reference contains a set of pointers for holding the address of the disk block where that specific key value stored.

Index Classification

1. Clustered and Unclustered

- If order of data records is the same as, or `close to', order of data entries, then it is called clustered index. Otherwise it is unclustered.

2. Primary Index and Secondary Index

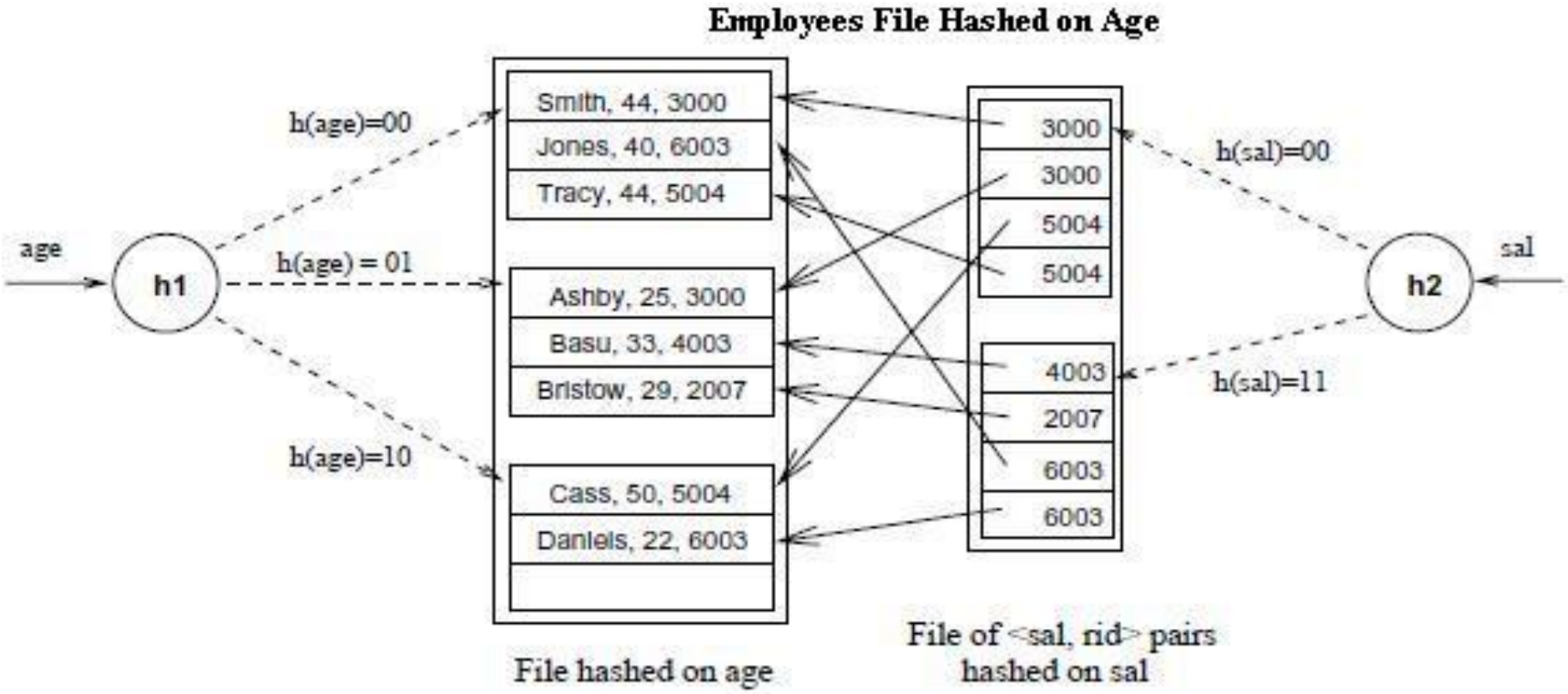
- An index on a set of fields that includes the primary key is called primary index and other indexes are called secondary indexes.
- A primary index is guaranteed not to contain duplicates, but an index on other fields can contain duplicates. A secondary index contains duplicates.

Index Data Structures

Hash Based Indexing

- We can organize records using a technique called hashing to quickly find records when given a search key value.
- Ex:- we can retrieve employee record based on the name field
- The records in a file are grouped in buckets, where a bucket consists of a primary page and zero or more overflow pages linked in a chain.
- Hashing function h: h(r) = bucket in which (data entry for) record r belongs. h looks at the search key fields of r.

Hash Based Indexing



- Hash Table is a data structure in which keys are mapped to array positions by a hash function.
- A value stored in the Hash Table can be searched in O(1) time using a hash function to generate an address from the key (by producing the index of the array where the value is stored).
- Hash Function, h is simply a mathematical formula which when applied to the key, produces an integer which can be used as an index for the key in the hash table. The main aim of a hash function is that elements should be relatively randomly and uniformly distributed.

- **DIFFERENT HASH FUNCTIONS**
- Assuming numeric keys are being used, the hash functions mostly used are

1) Division method
2) Multiplication method
3) Mid-square method
4) Folding method
5) Digit Analysis

- Division Method

- Division method is the most simple method of hashing an integer *x*. The method divides *x* by *M* and then use the remainder thus obtained. In this case, the hash function can be given as
- h(x) = x mod M
- The division method is quite good for just about any value of *M* and since it requires only a single division operation, the method works very fast. However, extra care should be taken to select a suitable value for *M*.
- Example: Calculate hash values of keys 1234 and 5462.

  Setting m = 97, hash values can be calculated as

  h(1234) = 1234 % 97 = 70

  h(5642) = 5642 % 97 = 16

- **Multiplication Method**
- The steps involved in the multiplication method can be given as below:
- **Step 1:** Choose a constant *A* such that *0 < A < 1.*
- **Step 2:** Multiply the key *k* by *A*
- **Step 3:** Extract the fractional part of *kA*
- **Step 4:** Multiply the result of Step 3 by *m* and take the floor.
- Hence, the hash function can be given as, **h (x) = ⌊ m (k A mod 1)**

where, *kA* mod 1 gives the fractional part of *kA* and m is the total number of indices in the hash table

- The greatest advantage of the multiplication method is that it works practically with any value of *A*. Although the algorithm works better with some values than the others but the optimal choice depends on the characteristics of the data being hashed. Knuth has suggested that the best choice of A is (sqrt5 - 1) /2 = 0.6180339887

Example: Given a hash table of size 1000, map the key 12345 to an appropriate location in the hash table

We will use A = 0.618033, m = 1000 and k = 12345

h(12345) = ⌊ 1000 (12345 X 0.618033 mod 1) ⌋

= ⌊ 1000 (7629.617385 mod 1) ⌋ = ⌊ 1000 (0.617385) ⌋ = 617.385 = 617

Mid Square Method

Mid square method is a good hash function which works in two steps.

Step 1: Square the value of the key. That is, find k^2

Step 2: Extract the middle *r* bits of the result obtained in Step 1.

The algorithm works well because most or all bits of the key value contribute to the result. This is because all the digits in the original key value contribute to produce the middle two digits of the squared value. Therefore, the result is not dominated by the distribution of the bottom digit or the top digit of the original key value.

In the mid square method, the same *r* bits must be chosen from all the keys. Therefore, the hash function can be given as,

h (k) = s

where, *s* is obtained by selecting r bits from k^2

Example: Calculate the hash value for keys 1234 and 5642 using the mid square method. The hash table has 100 memory locations.

Note the hash table has 100 memory locations whose indices vary from 0-99. this means, only two digits are needed to map the key to a location in the hash table, so *r = 2.*

When k = 1234, k2 = 1522756, h (k) = 27

When k = 5642, k2 = 31832164, h (k) = 21

Observe that 3rd and 4th digits starting from the right are chosen.

- Folding Method
- The identifier x is partitioned into several parts, all but the last being of the same length.
- All partitions are added together to obtain the hash address for x.

Shift folding

Different partitions are added together to get h(x).

Folding at the boundaries

Identifier is folded at the partition boundaries, and digits falling into the same position are added together to obtain h(x). This is similar to reversing every other partition and then adding.

The shift folding method works in two steps.

Step 1: Divide the key value into a number of parts. That is divide k into parts, *k1, k2, …, kn,* where each part has the same number of digits except the last part which may have lesser digits than the other parts.

Step 2: Add the individual parts. That is obtain the sum of *k1 + k2 + .. + kn*. Hash value is produced by ignoring the last carry, if any.

Example: Given a hash table of 100 locations, calculate the hash value using shift folding method for keys- 5678, 321 and 34567.

Here, since there are 100 memory locations to address, we will break the key into parts where each part (except the last) will contain two digits.

Therefore,

Key	5678	321	34567
Parts	56 and 78	32 and 1	34, 56 and 7
Sum	134	33	97
Hash Value	34 (ignore the last carry)	33	97

Digit Analysis

- This method is useful when a static file where all the identifiers in the table are known in advance.
- Each identifier x is interpreted as a number using some radix r.
- The digits of each identifier are examined.
- Digits having most skewed distributions are deleted.
- Enough digits are deleted so that the number of remaining digits is small enough to give an address in the range of the hash table.

Index Data Structures

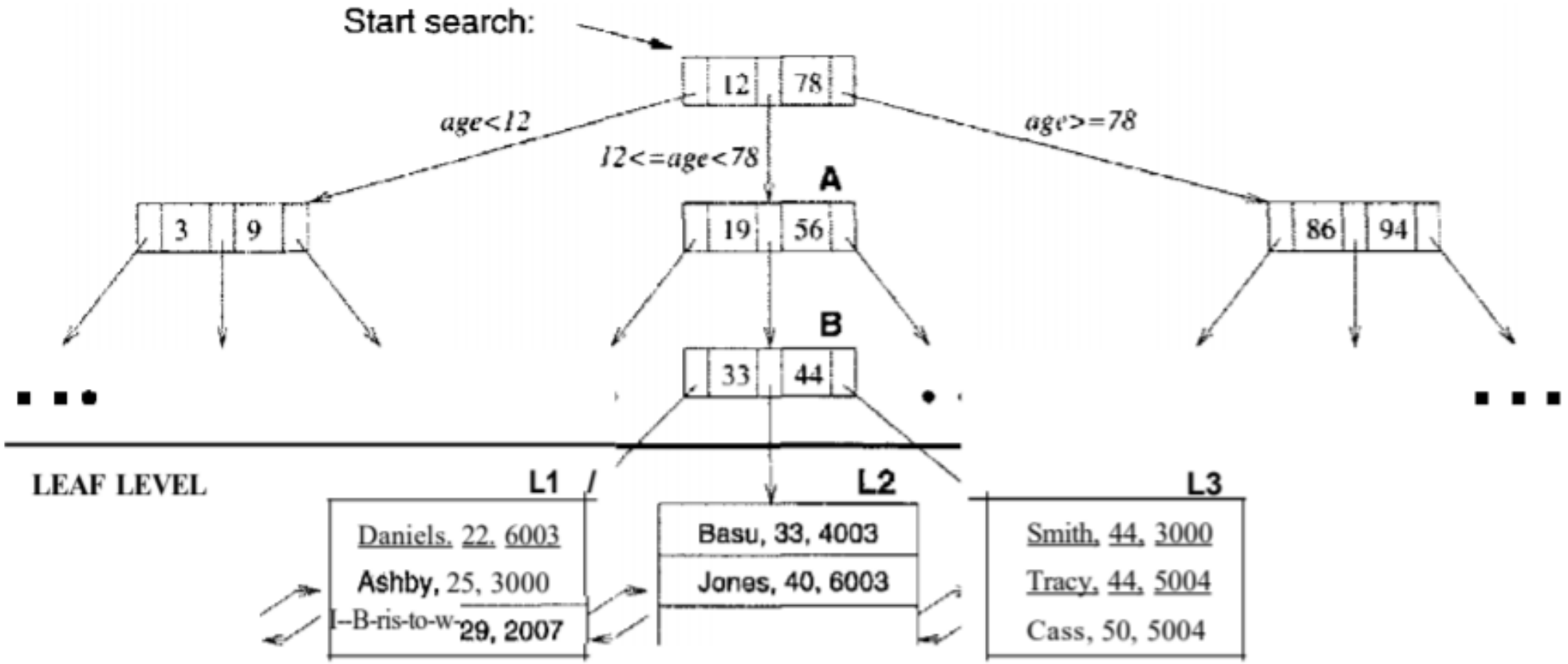
Tree Based Indexing

The data entries are arranged in sorted order by search key value & hierarchical search data structure is maintained that directs searches to the correct page of data entries.

Ex:- Employee records are organized in a tree-structured index with search key age. Each node in this diagram(A,B,L1,L2) is a physical page & retrieving a node involves a disk I/O.

The lowest level of the tree called leaf level contains the data entries.

Tree Based Indexing



B-Tree

B-tree is a data structure that store data in its node in sorted order.

B-tree stores data such that each node contains keys in ascending order. Each of these keys has two references to another two child nodes.

The left side child node keys are less than the current keys and the right side child node keys are more than the current keys.

B-Tree

- A B-tree is self balancing BST and a specialized m- way tree that is widely used for disk access. It is developed by Rudolf Bayer and Ed McCreight in 1970.
- A B tree of order m can have maximum m-1 keys and m pointers to its sub-trees. A B-tree may contain a large number of key values and pointers to sub-trees. Storing a large number of keys in a single node keeps the height of the tree relatively small.
- A B-tree is designed to store sorted data and allows search, insert, and delete operations to be performed in logarithmic amortized time.
- A B-tree of order *m* (the maximum number of children that each node can have) is a tree with all the properties of an m-way search tree and in addition has the following properties:

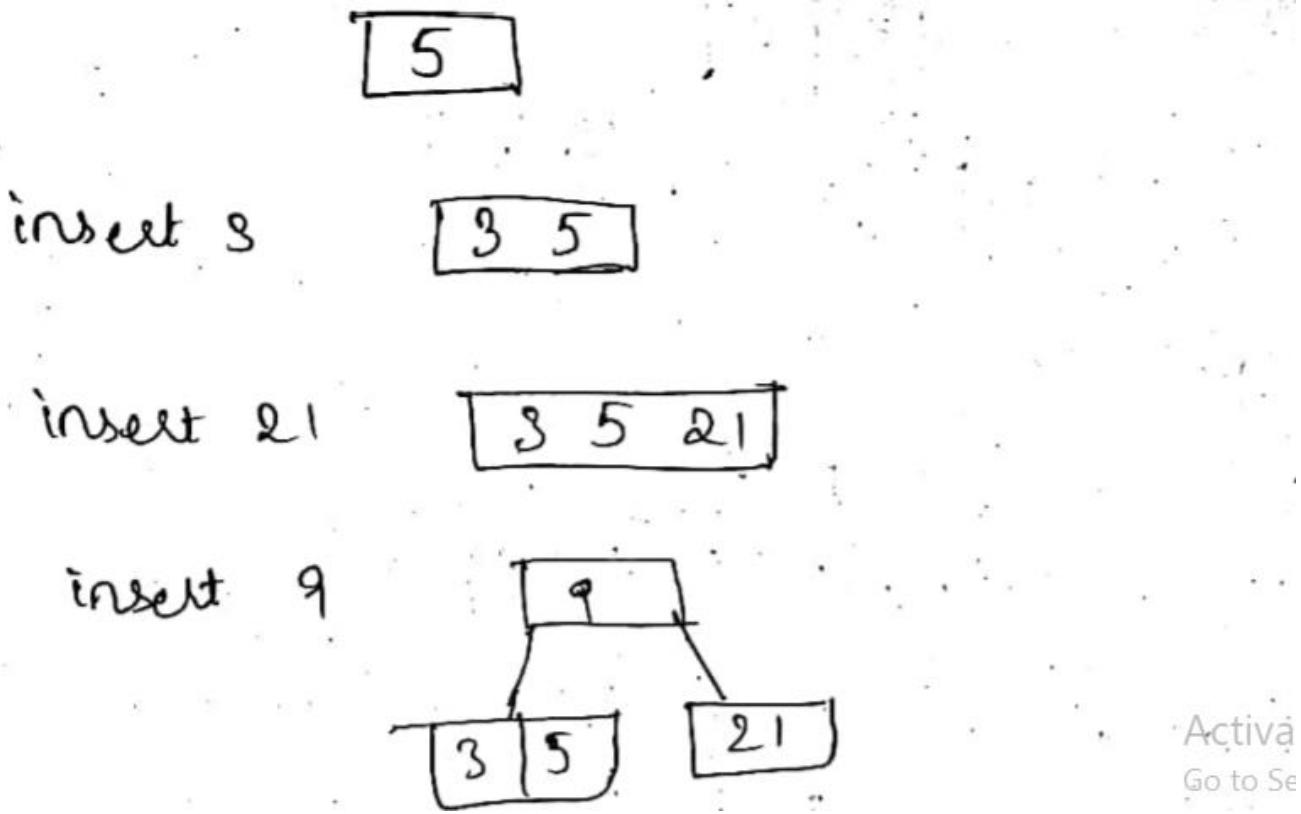
B-Tree

- Every node in the B-tree has at most (maximum) *m* children.
- Every node in the B-tree except the root node and leaf nodes have at least (minimum) *m/*2 children. This condition helps to keep the tree bushy so that the path from the root node to the leaf is very short even in a tree that stores a lot of data.
- The root node has at least two children if it is not a terminal (leaf) node.
- All leaf nodes are at the same level.
- An internal node in the B tree can have n number of children, where 0 ≤n ≤ m. It is not necessary that every node has the same number of children, but the only restriction is that the node should have at least m/2 children.

B Tree Insertion

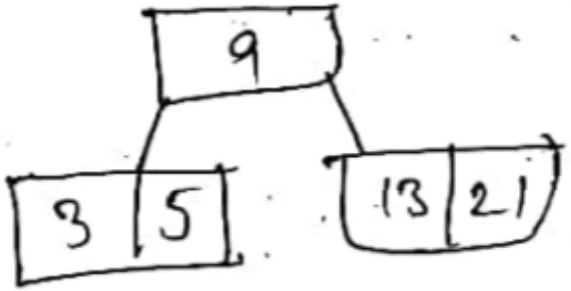
Construction of B Tree for the elements 5,3,21,9,13,1,2,7,10,12,25

Order 4

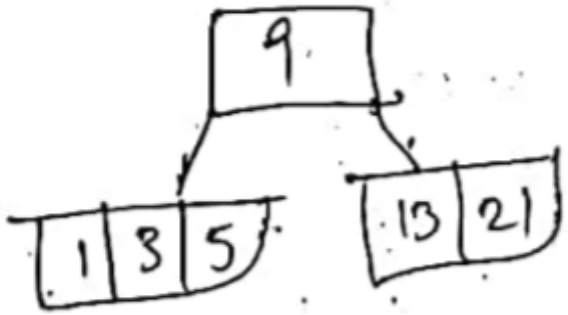


B Tree Insertion

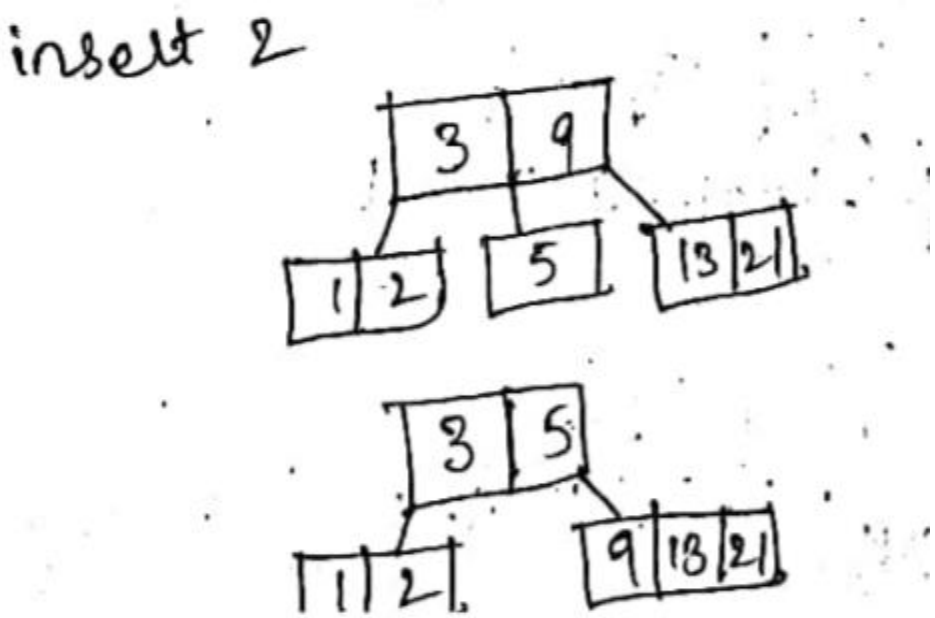
insert 13



insert 1

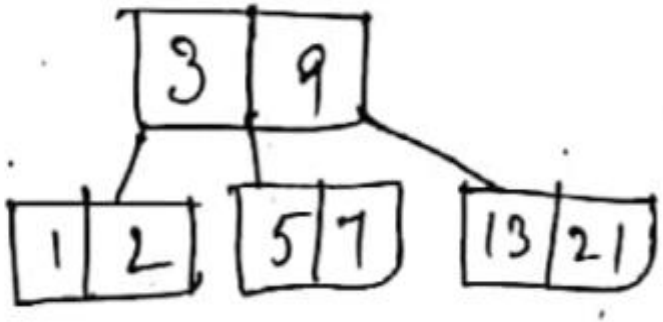


B Tree Insertion

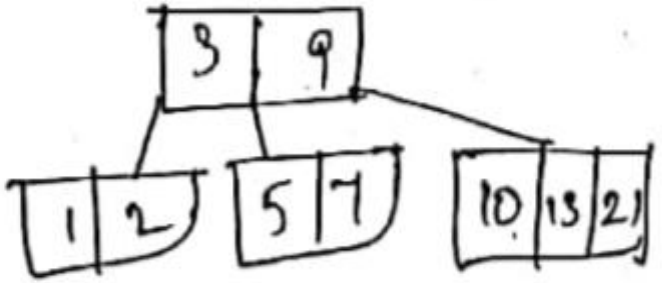


B Tree Insertion

insert 7

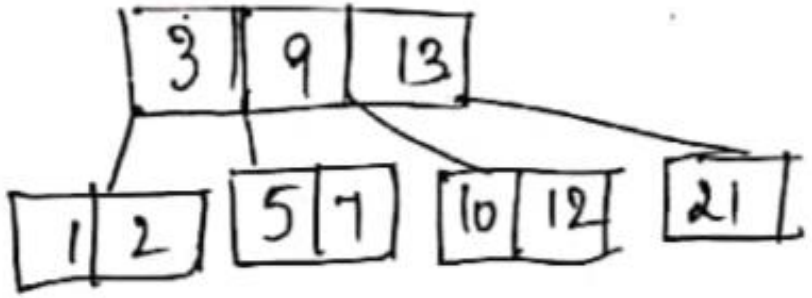


insert 10

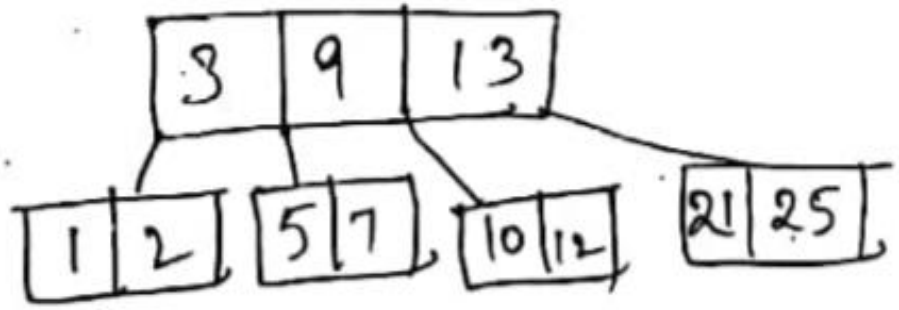


B Tree Insertion

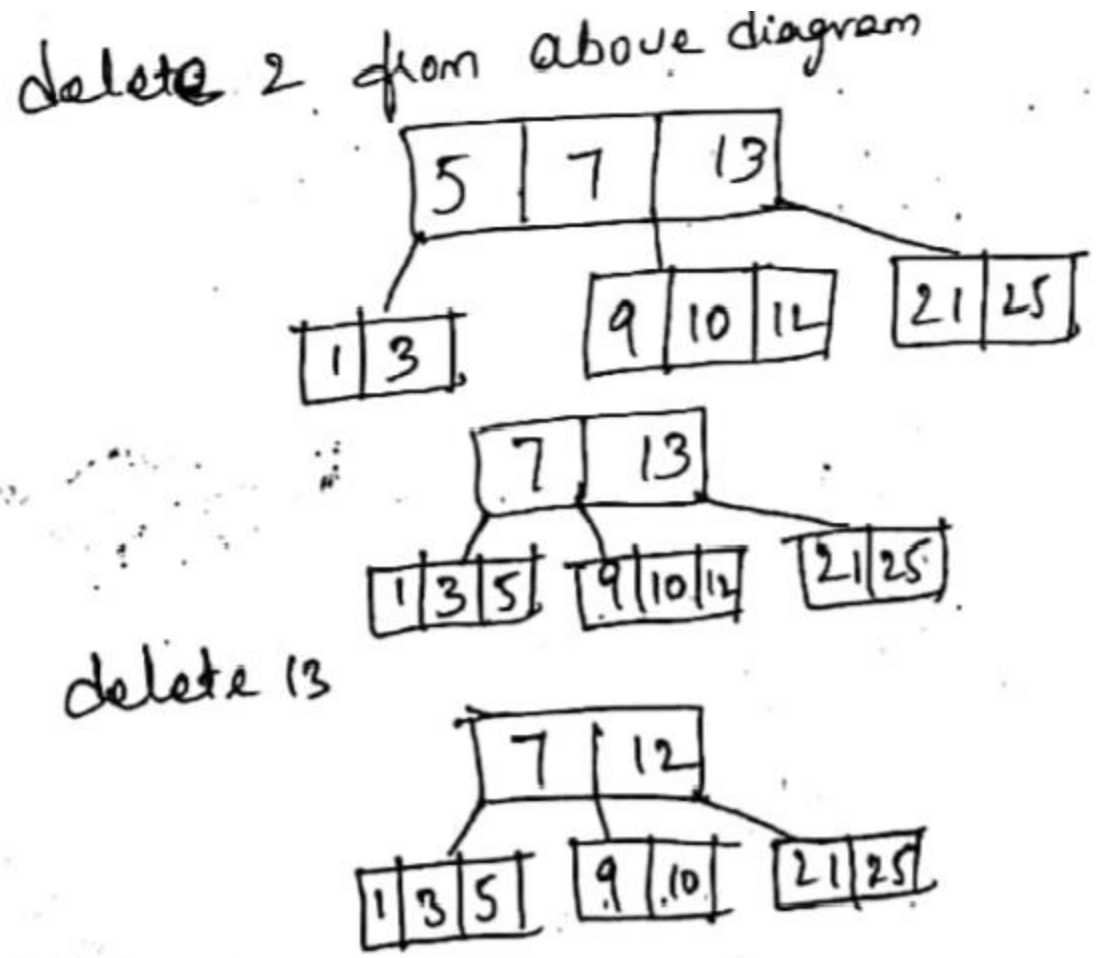
insert 12



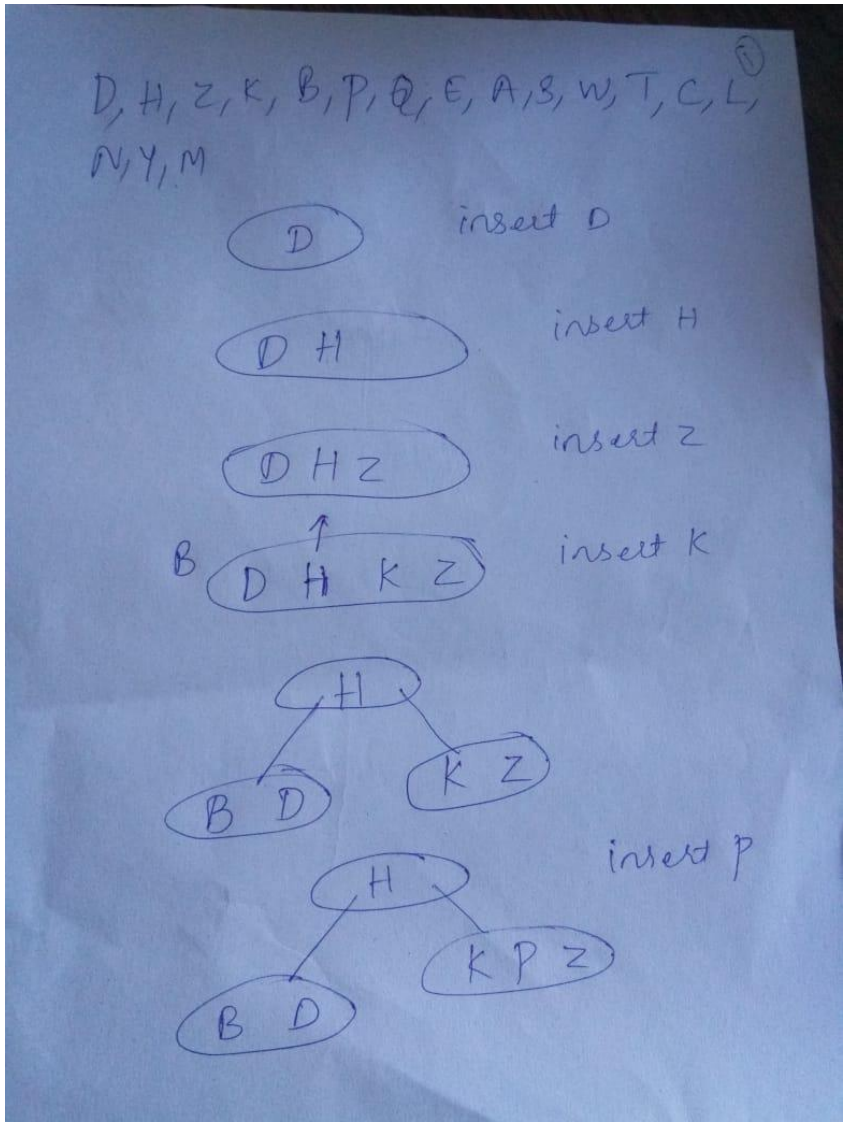
insert 25



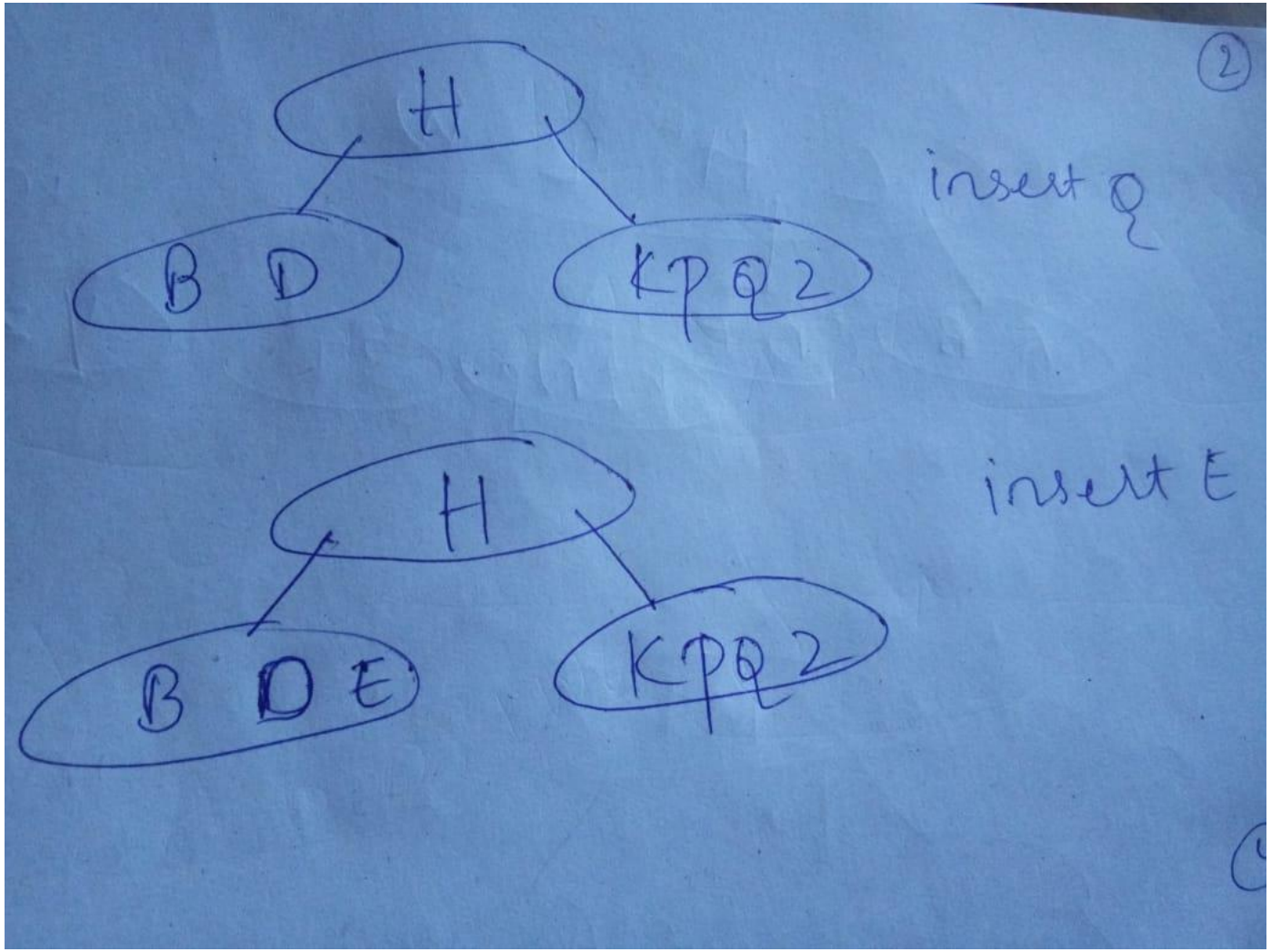
B Tree Deletion



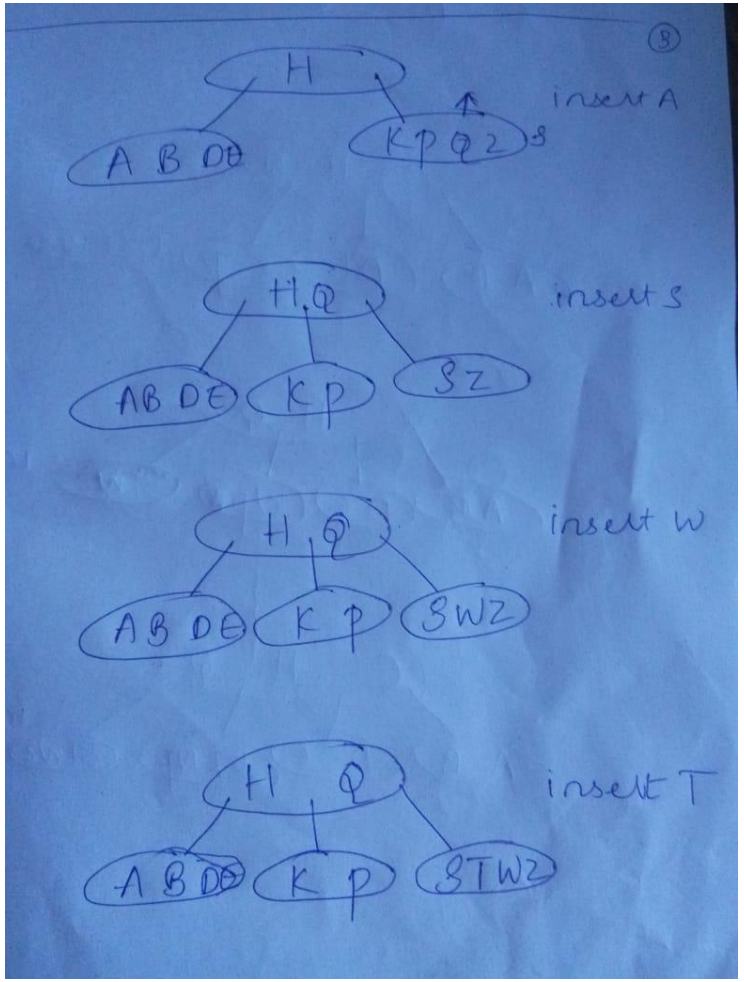
B Tree Insertion



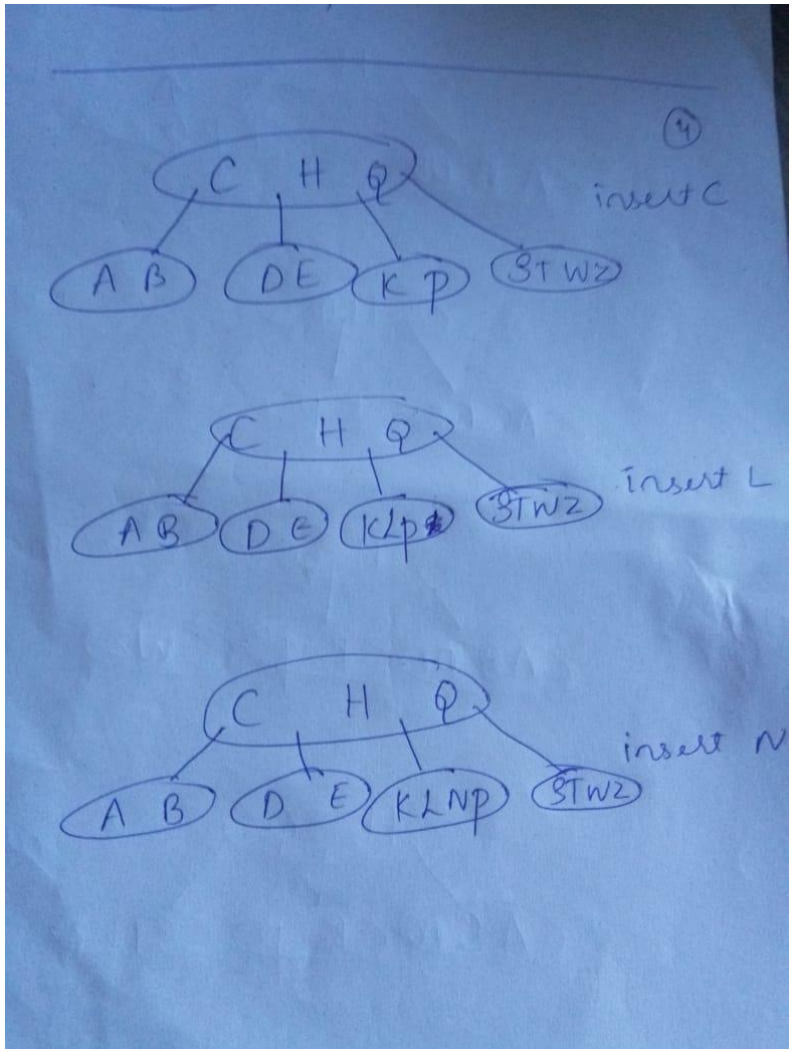
B Tree Insertion



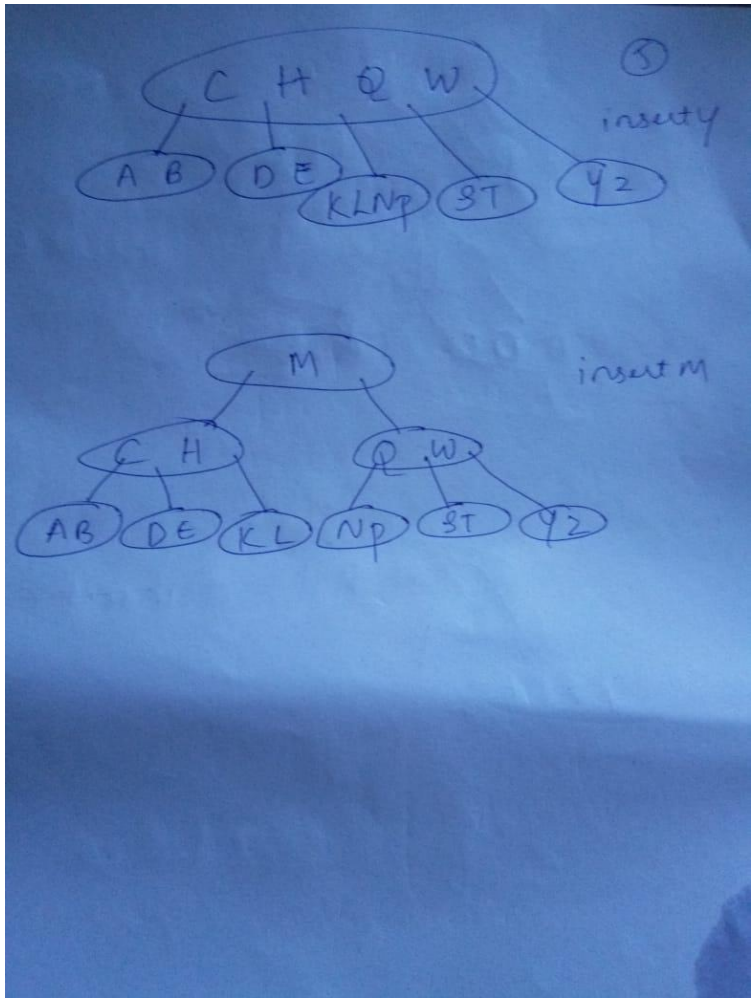
B Tree Insertion



B Tree Insertion

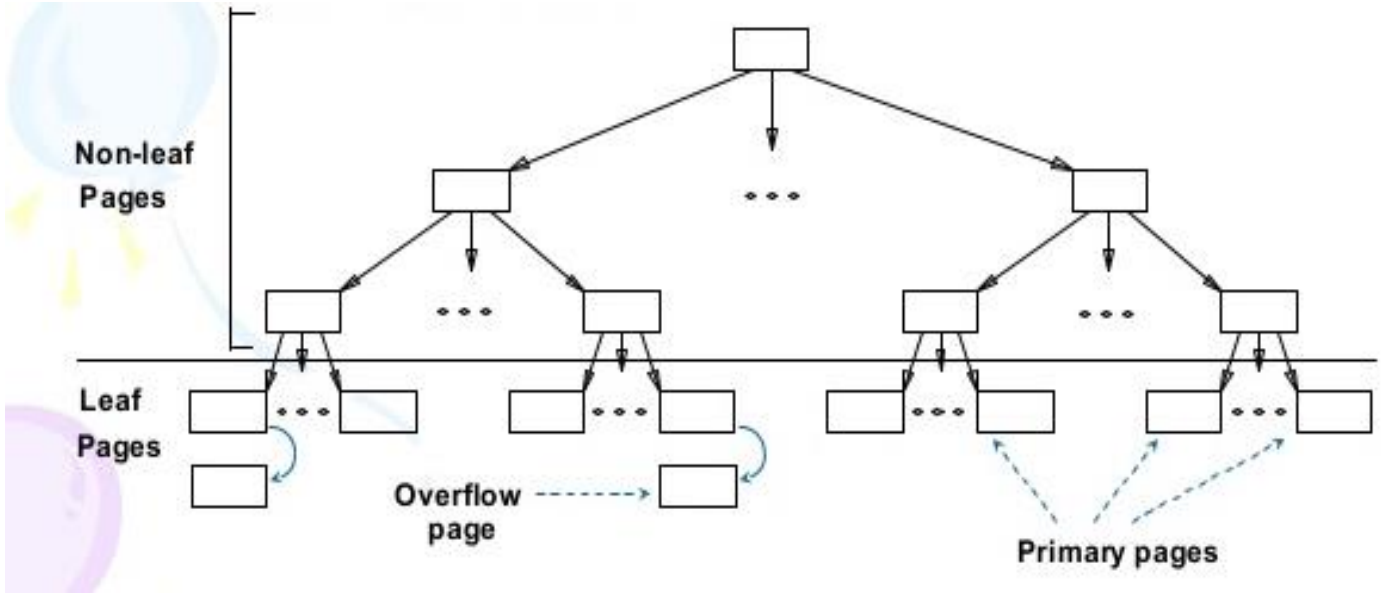


B Tree Insertion

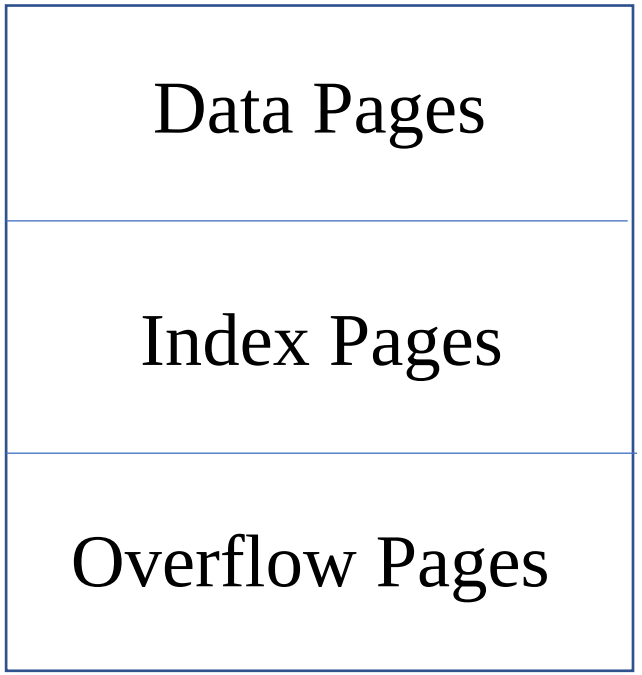


ISAM-Indexed Sequential Access Method

ISAM is a static index structure. The data entries of ISAM are in the leaf pages of the tree and additional overflow pages chained to some leaf page. Each Tree node is a disk page



ISAM-Indexed Sequential Access Method



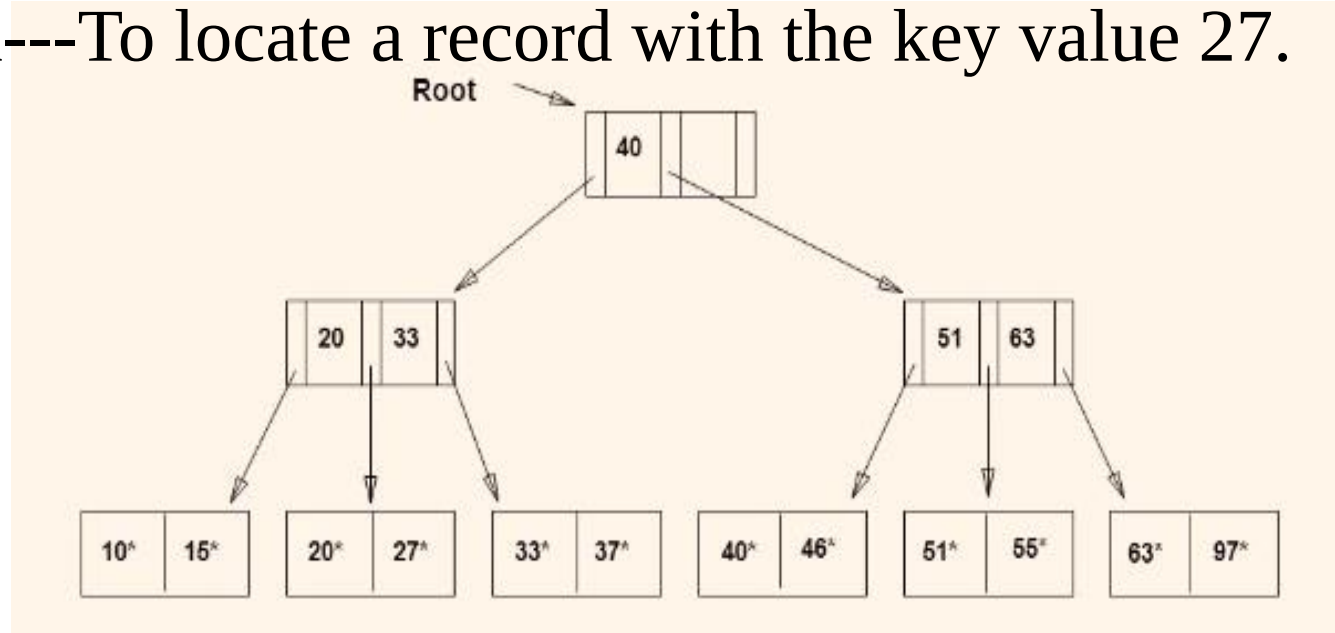
Page Allocation in ISAM

ISAM-Indexed Sequential Access Method

Only leaf nodes are changed, the non-leaf nodes remain same.

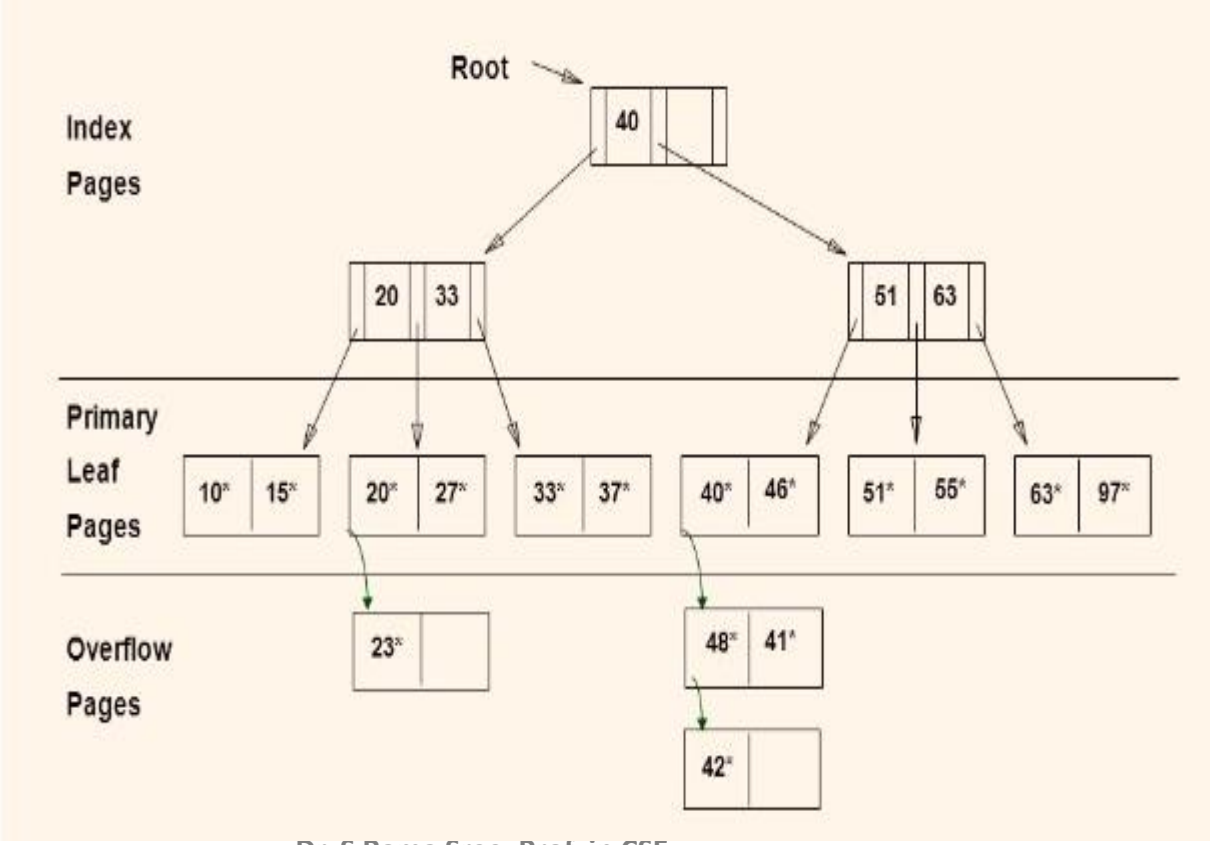
Operations:-

1) Insertion
2) Deletion
3) Search---To locate a record with the key value 27.



ISAM-Indexed Sequential Access Method

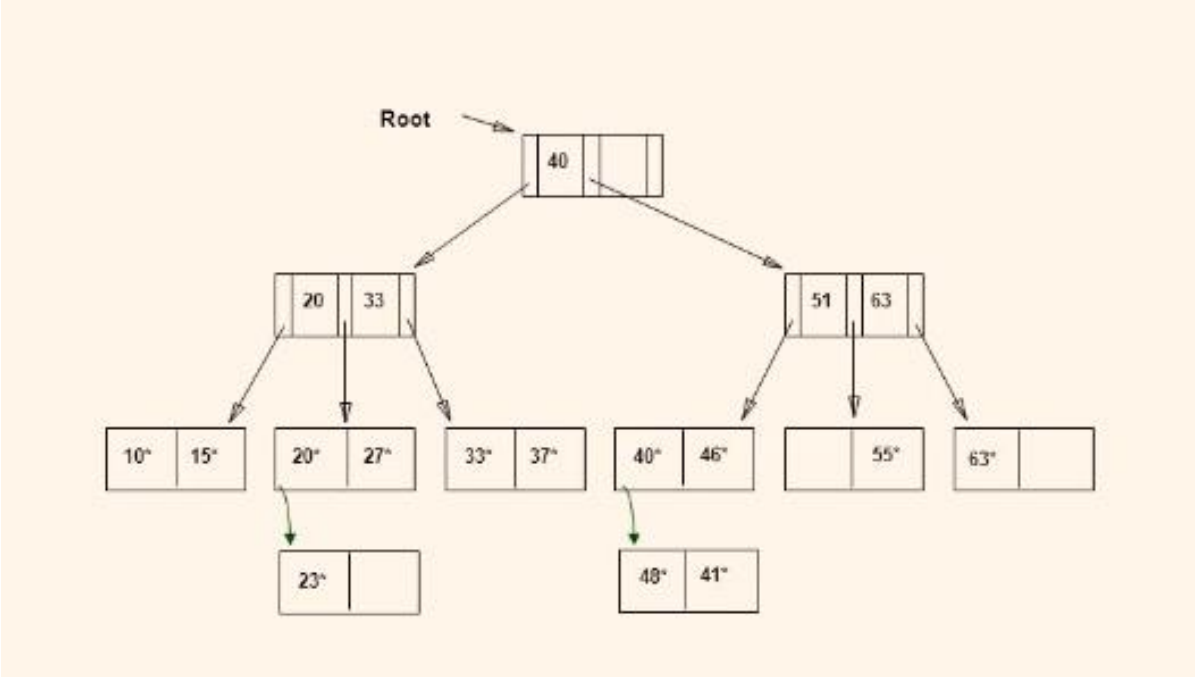
To insert a record with key value 23. 23* in the overflow page.
To insert 48,41,42 leads to overflow chain of 2 pages.



ISAM-Indexed Sequential Access Method

If we delete a Overflow page & overflow page becomes empty.
If we delete a primary page that makes primary page empty and it serves as a place holder for future insertions.

Delete 42



B+ Tree

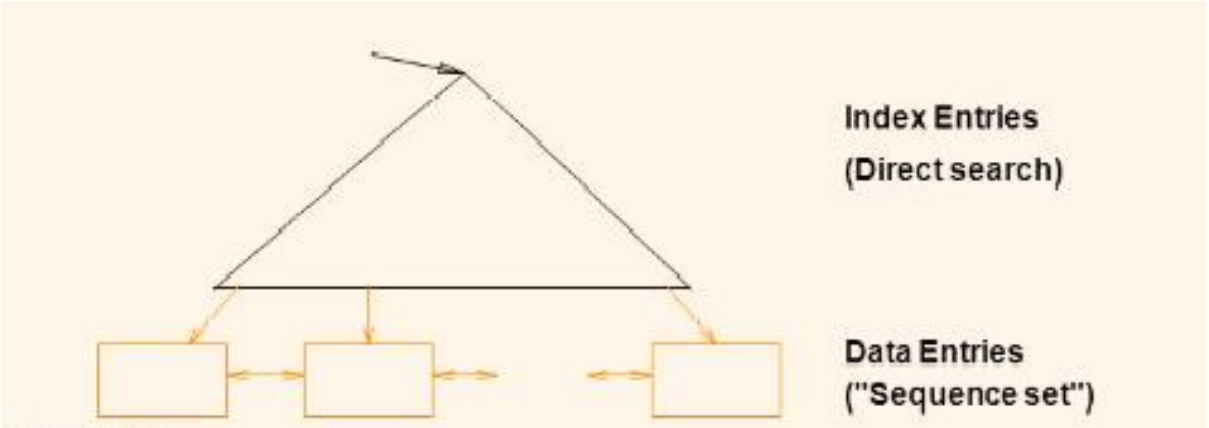
ISAM Suffers from the problem that long overflow chains can develop as the file grows.

B+ Tree is a dynamic structure means the tree grows and shrinks dynamically.

Balanced tree in which the internal nodes direct the search and leaf nodes contain the data entries. Non leaf node is a <key-value,node-pointer> pair.

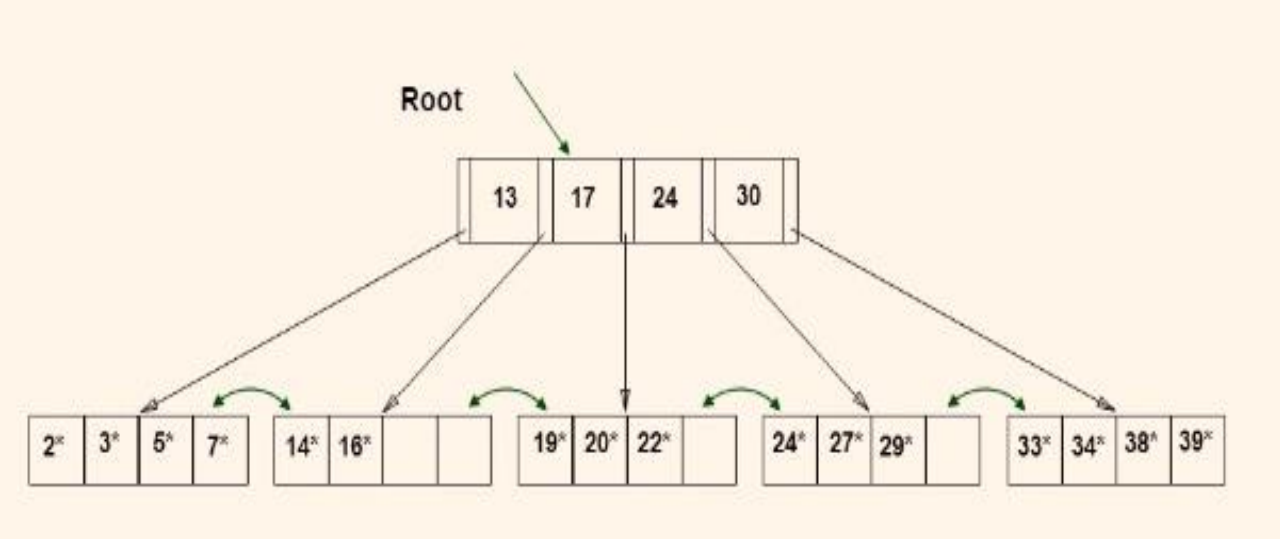
To retrieve all leaf pages efficiently, we have to link them using page pointers(doubly linked list).

Structure of B+ Tree



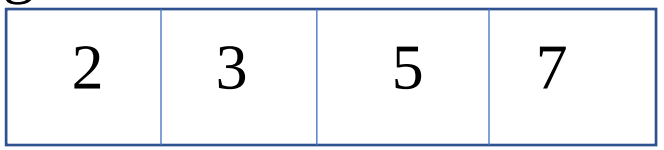
B+ Tree

Search 5* we follow the left most child pointer since 5 <13.
To search for the entries 14* and 15* we follow the second pointer since 13<14<17 and 13<15<17.
To find 24* we follow the fourth child pointer since 24<=24<30.

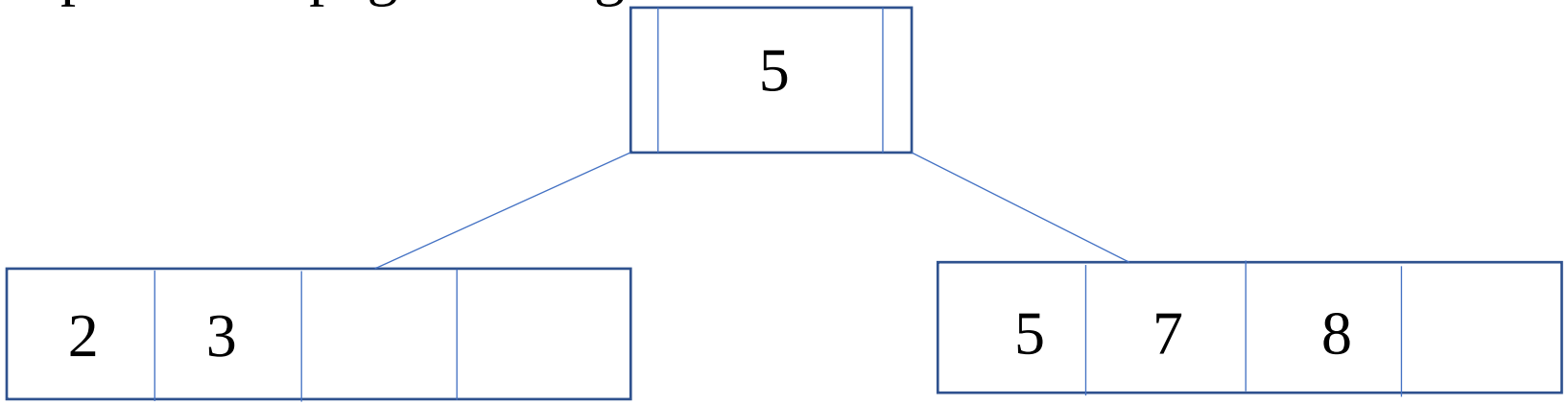


B+ Tree Insertion

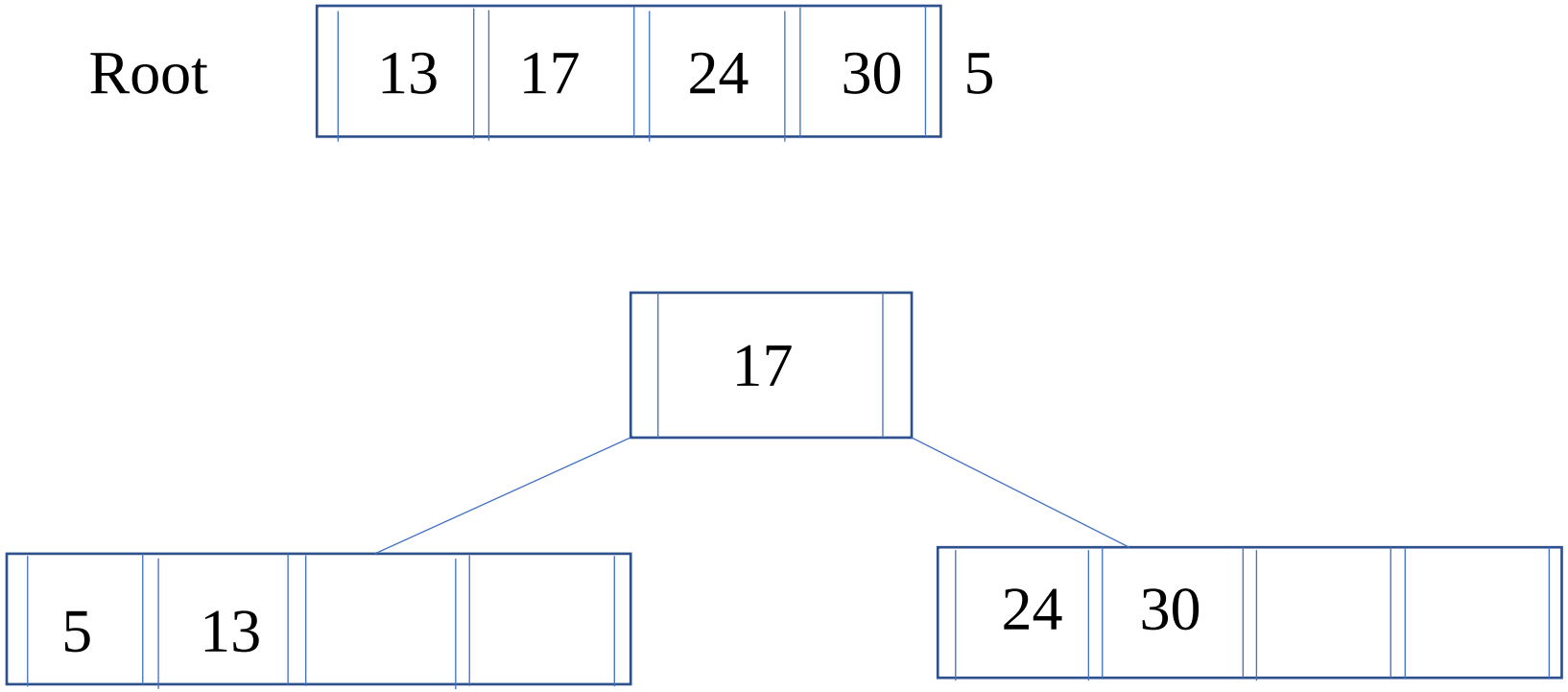
We insert 8* in the left most leaf which is already full. Insertion causes a split of the leaf page. Order=5



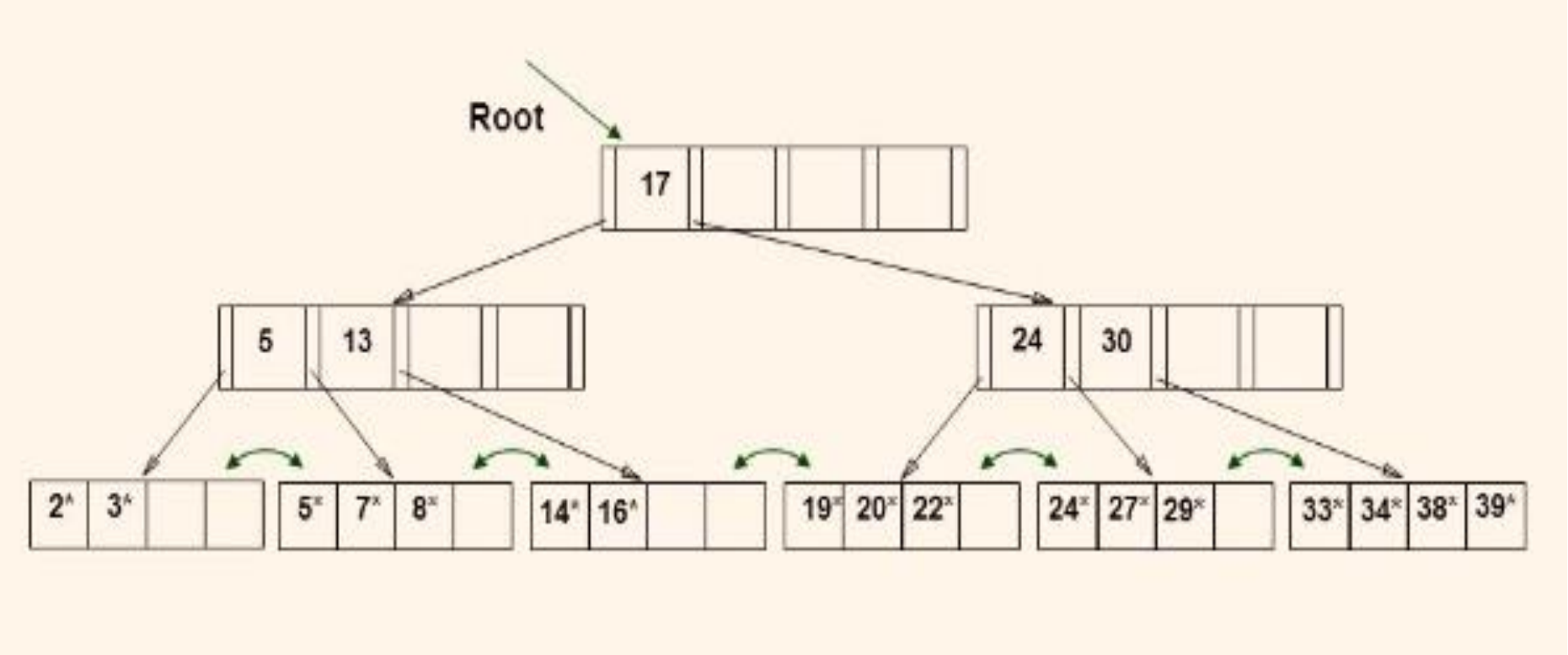
Split index pages during the insertion of 8*



B+ Tree

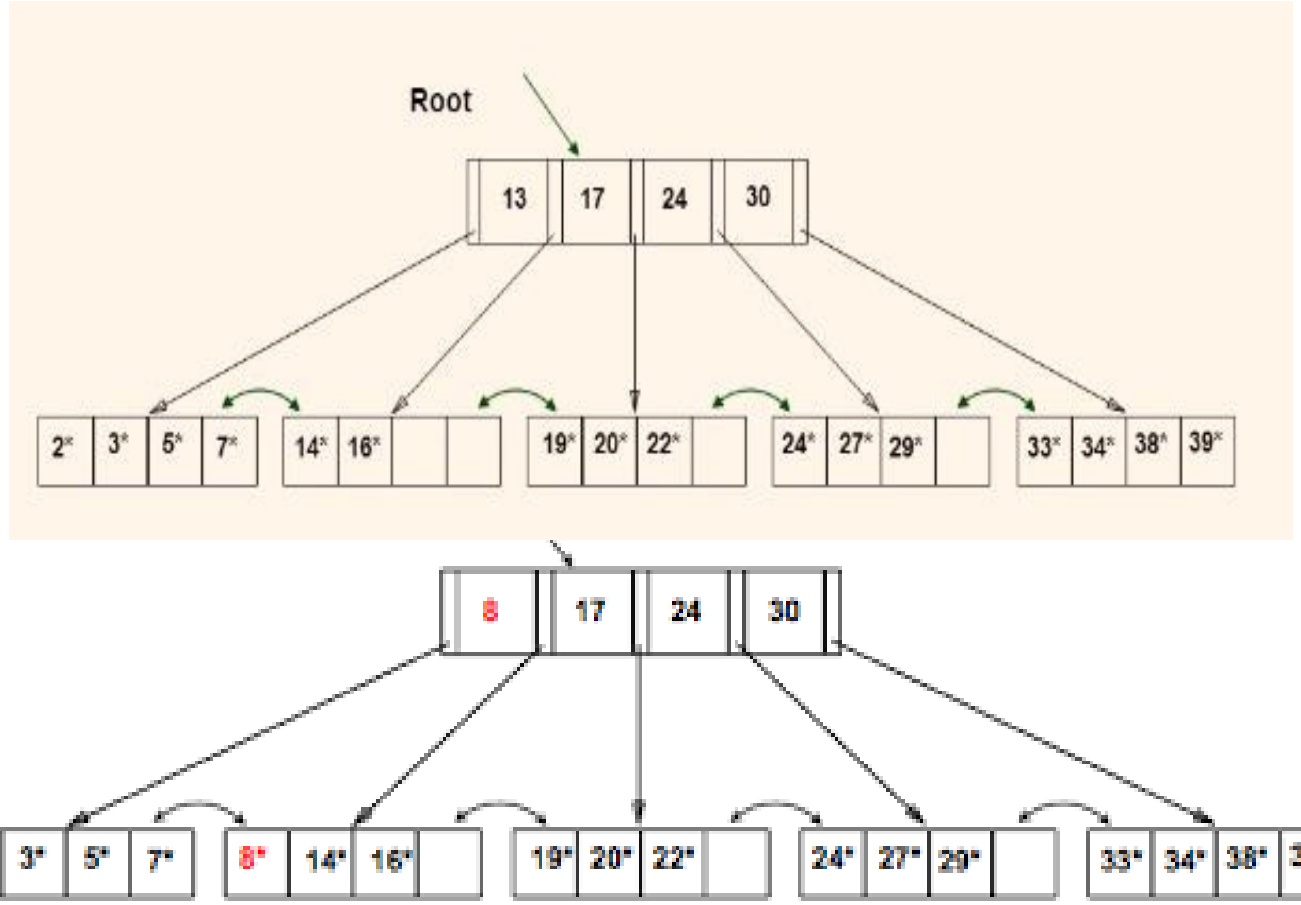


B+ Tree



B+ Tree Redistribution

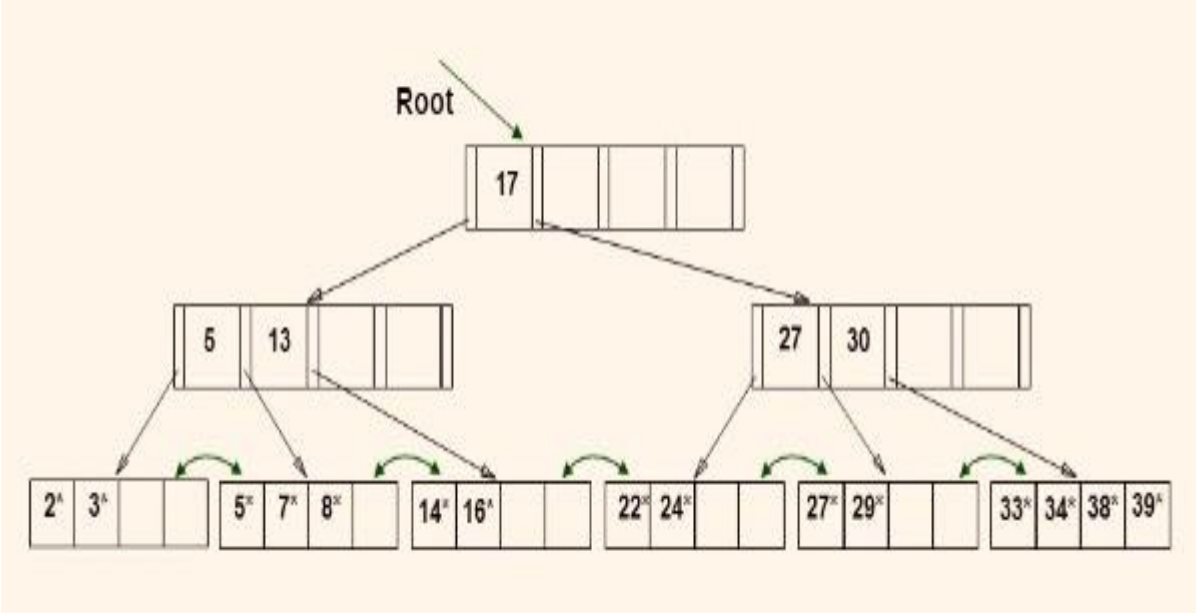
Insert 8* using redistribution



B+ Tree Deletion

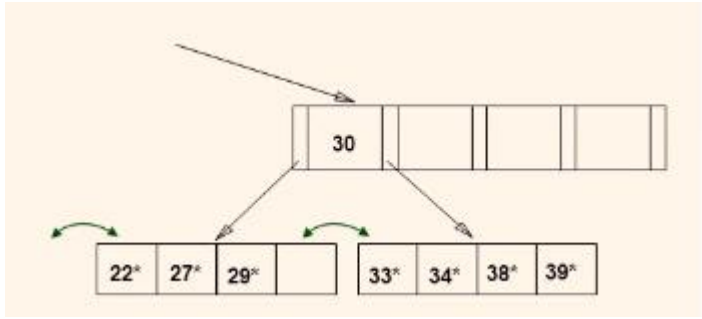
Delete 19, we simply remove it from the leaf page & leaf still contains two entries.

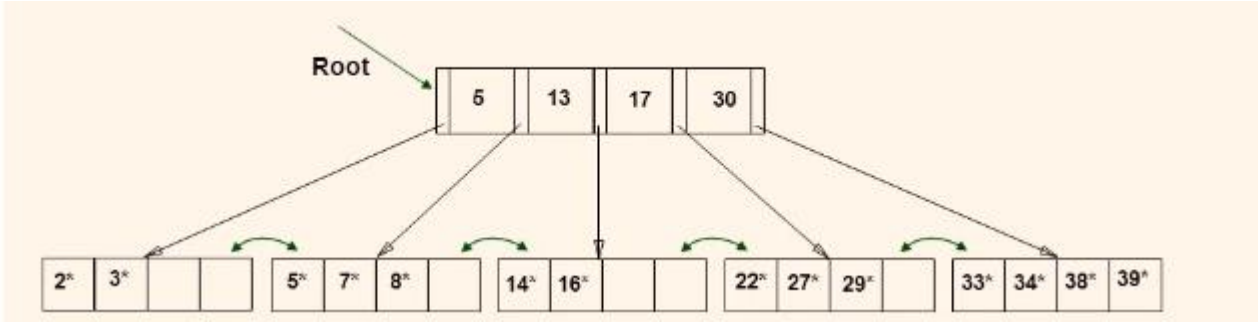
Delete 20, leaf contains only one entry. The sibling of the leaf node has 3 entries & we can have redistribution.



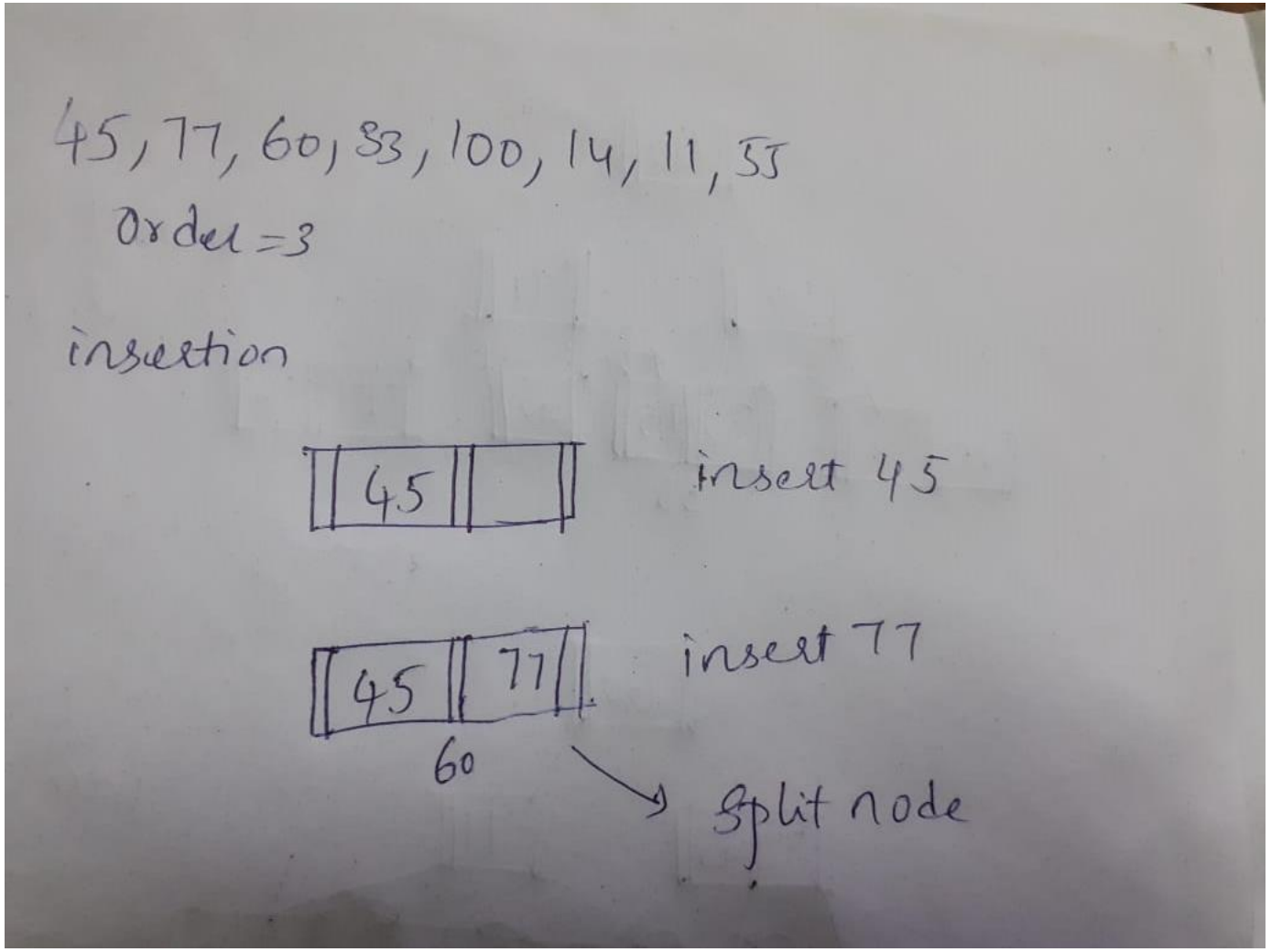
B+ Tree Deletion

Delete 24, redistribution

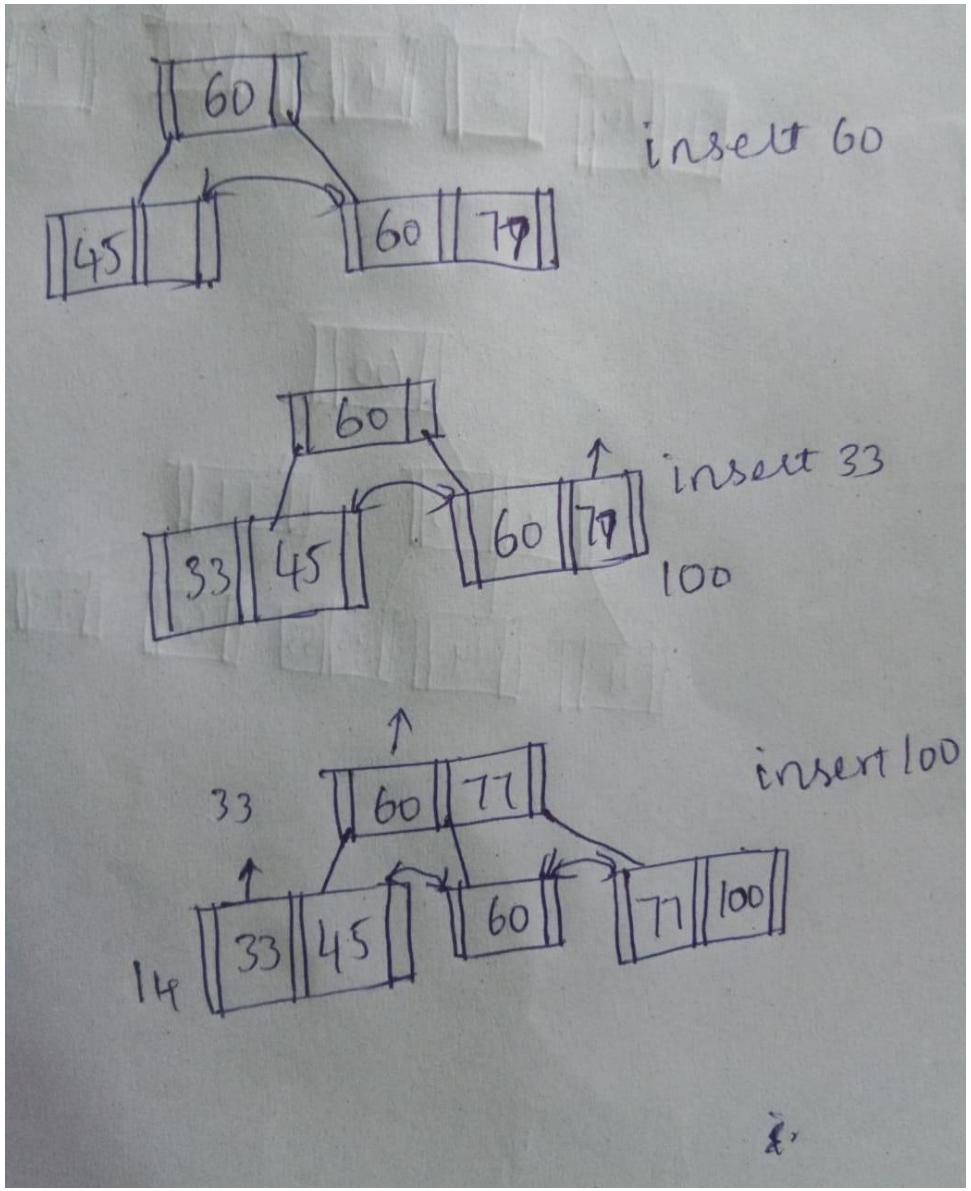




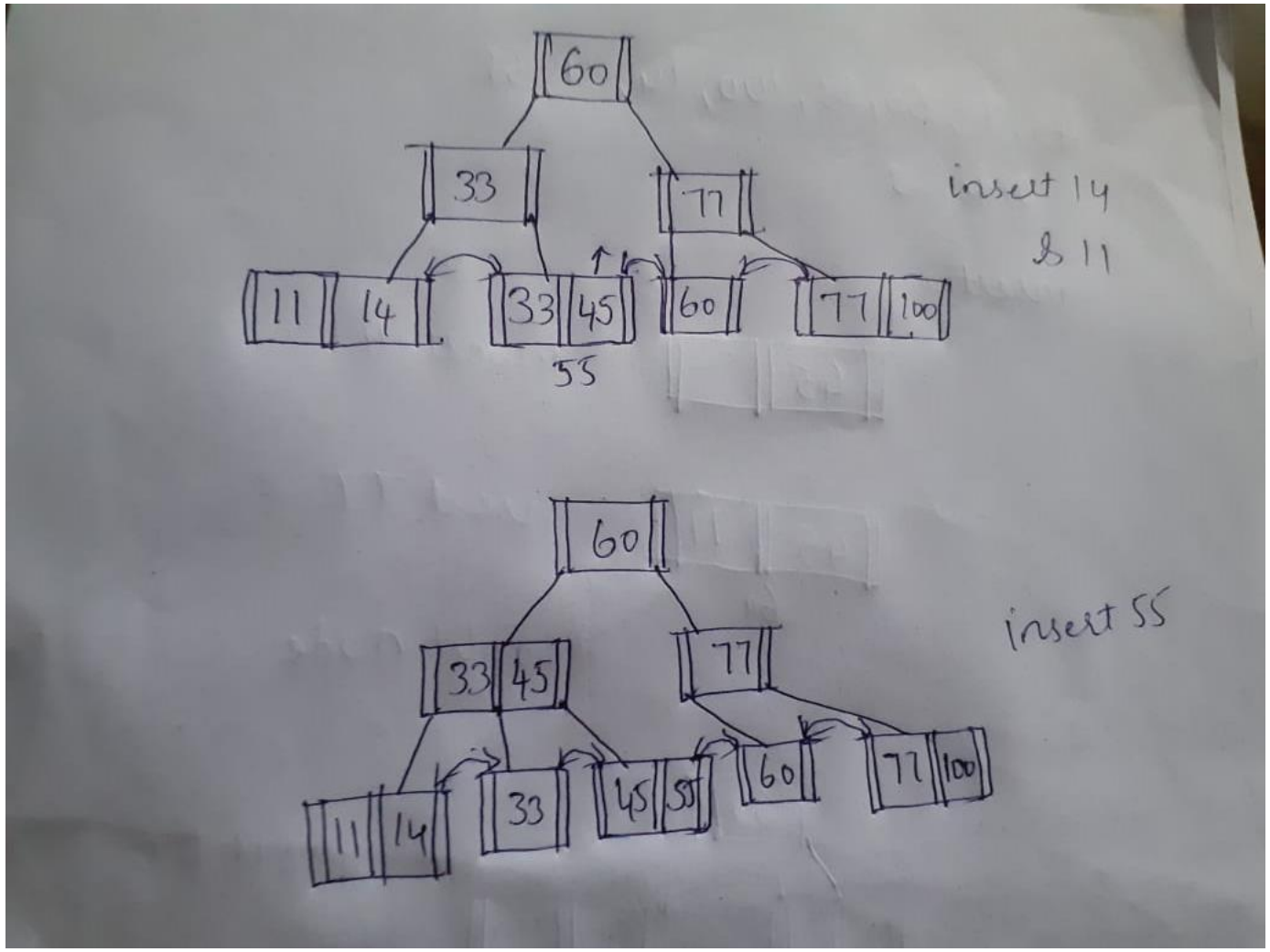
B+ Tree


45, 77, 60, 83, 100, 14, 11, 55
Order = 3
insertion
45
insert 45
45
77
insert 77
60
split node

B+ Tree


60
45
60
79
insert 60
60
33
45
60
79
insert 33
100
33
60
77
insert 100
14
33
45
60
77
100

B+ Tree


60
33
77
11
14
33
45
60
77
100
55
insert 14
& 11
60
33
45
77
11
14
33
45
55
60
77
100
insert 55

References

Text Books:

- Database Management Systems, 3/e, Raghurama Krishnan, Johannes Gehrke, TMH.
- Database System Concepts,5/e, Silberschatz, Korth, TMH.

Reference Books:

- Introduction to Database Systems, 8/e C J Date, PEA.
- Database Management System, 6/e Ramez Elmasri, Shamkant B. Navathe, PEA.
- Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning.

Web Links:

- https://nptel.ac.in/courses/106/105/106105175/
- https://www.geeksforgeeks.org/introduction-to-nosql/
- https://www.youtube.com/watch?v=wkOD6mbXc2M
- https://beginnersbook.com/2015/05/normalization-in-dbms/