Sreelekshmi Prathapan
MCA S1
Roll No: 102

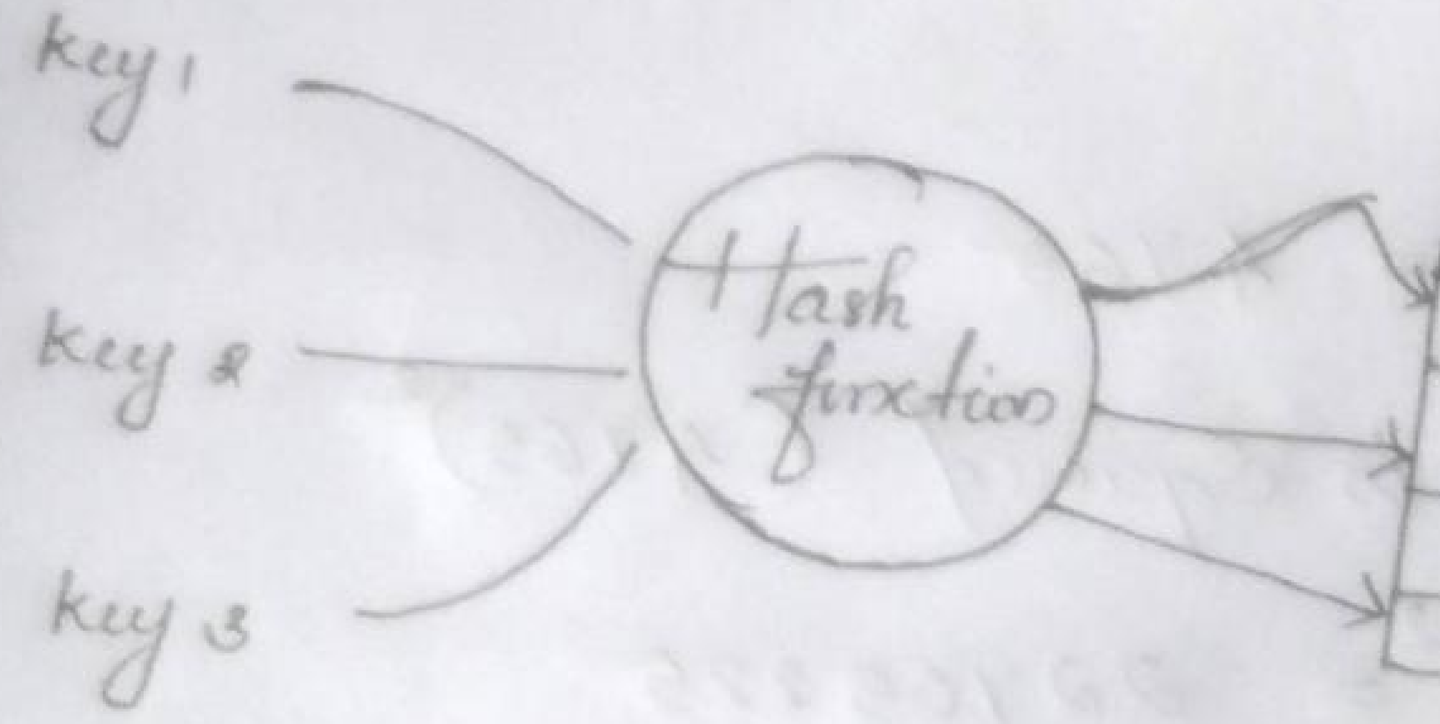
Advanced Data Structure

Part B.

12(a) Hashing

Hashing is a technique to convert a range of key values into a range of indexes of an array.

- Hash table is a data structure which stores data in an associative manner.
- In a hash table data is stored in an array format.
- Each data value has its own unique index value.



index	Value
0	value1
1	value2
2	value3
3	value4

There are some hash function methods are there, They are

1. Folding Method
2. Mid Square Method
3. Truncation Method

Folding Method

* The key 'k' is divided into parts, each of those parts have same length
* The parts are then added together

Eg: if $k = 468\ 117\ 134$

divide they key into equal parts

$P_1 = 468 \quad P_2 = 117, \quad P_3 = 134$

add $P_1 + P_2 + P_3 \Rightarrow$ address.

* Mid Square Method

• A key value is selected and it is squared and then the mid value is assigned as the address.

Example. $k = 4765$

Then the k is squared $(4765)^2$

$(4765)^2 = 22\underbrace{705}225$

705 is taken as address.

* Truncation Method

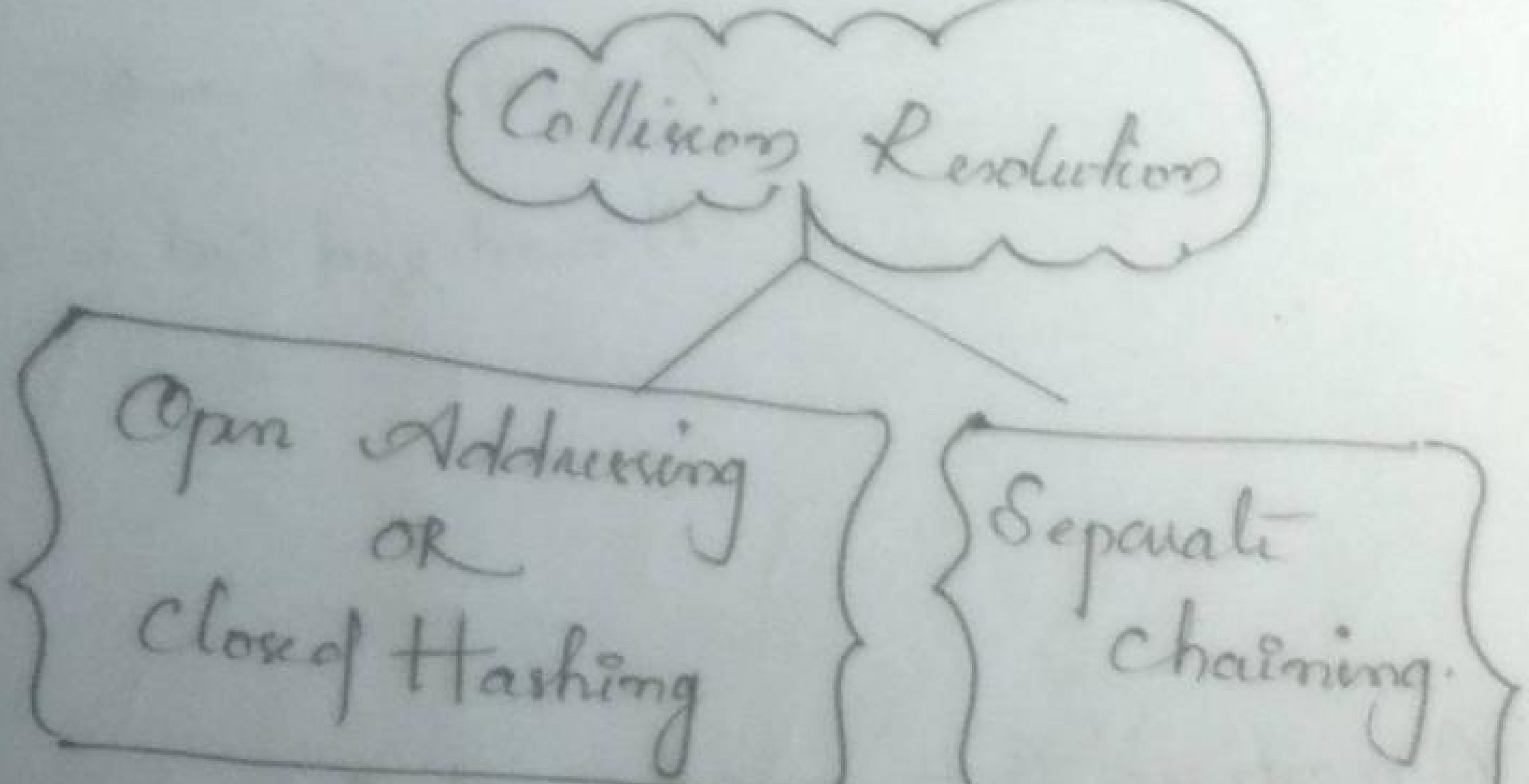
• Ignore a part of the key and use the remaining part directly as the index.

Collision And Collision Resolution

- **Collision**

Collision or clash is a Situation that occurs when two distinct pieces of datas have the Same hash value.
The impact of Collisions depends on the application. Collisions are unavoidable whenever members of a Very large Set are mapped to a hash value.

- Hash functions can map different data to Same hash value then in Order to reduce the Collision we use Some techniques.



1) **Open Addressing OR closed Hashing**

In open Addressing the key value is mapped to particular position in the hash table. if the position is already occupied Then the value is inserted in Some other empty location.

The technique depends on Space usage and can be done with,

* Linear probing
* Quadratic probing
* Double Hashing.

* **Linear Probing**

Linear probing is the technique used in Open addressing in order to avoid Collision.

- If the address given by a hash function is 'a' and it is already occupied, then we try to insert to the next location. ie;

$$a \rightarrow a+1 \text{ (if 'a+1' also occupied)}$$

$$\text{then } a+1 \rightarrow a+2.$$

Consider

11, 12, 13, 14, 15, 16, 10, 8, 9, 20, 21.

table size = 11

h (key) = key % table size.

h (11) = 11 % 11 = 0.

h (12) = 12 % 11 = 1

Then insert 11 in '0'

insert 12 in '1'

The formula $H(k, i) = h((k)+i) \text{ Mod } T \text{ Size}$.

* **Quadratic Probing**

The Quadratic probing is the technique used for avoiding Collision. In linear probing the colliding values or keys are stored to the next or nearest point.

But in Quadratic probing if the position is already occupied then the position value is squared.

- In quadratic probing the problem is solved by storing the colliding keys away from the initial collision point.

formula $H(k, i) = (h(k) + i^2)$ Mod T size.

* Double Hashing

In double Hashing the probing intervals is fixed

The double hashing technique uses one hash value as an index into the table and then repeatedly steps forward an interval until the desired value is located or reached.

formula $H(k, i) = h((k) + i(h')(k)$ mod T size.

2) Separate chaining.

Separate chaining is the procedure or technique used for avoiding the collision.

This technique creates a linked list to the slot for which collision occurs.

- The new key is inserted to the linked list
- These linked list appears like chains.

11)

B. Amortized Analysis.

- Amortized Analysis is a method for analysing a given algorithms Complexity.
- In amortized analysis the time required to perform a Sequence of Operation is averaged ~~all~~ over all the operations performed.
- It looks the worst case run time per operation rather than per algorithm.
- Amortized Analysis guarenties the average performance of each operation in worst case.

The Common techniques used in amortized analysis are of,

1. Aggrigate Method
2. Accounting Method
3. Potential Method

1. Aggrigate Method

Aggrigate method is one of the popular method used for amortized analysis. Here,

- we determine an Upper bound T(n) on the total Sequence of 'n' operations.

Then the cost of each will be,

$$\text{Cost} = \frac{T(n)}{n}$$

Example: Stack with new operation multipop (s, k);

If we consider PUSH and pop to be elementary Operation, then MULTIpop takes O(n) in the worst case.

⇒ Stack Operation

Two fundamental Stack Operation, each takes O(1) time

PUSH(s, k) → Pushes object k on 's' Stack

pop (s, k) → Popes out object k from Stack 's'

⇒ Push and pop operations runs in O(1) time.

Then the time taken for completing 'n' push and 'n' pop takes the running time of O(n).

The Code for MULTIpop

```
Multipop (s, k)

While not Stack-empty (s) and k > 0
        pop (s)
        k = k-1
```

Example.

Stak (s) →

10
19
20
30
46
21

Multipop (s, 3) →

30
46
21

Potential Method

Similar to accounting method instead of using cost and credit method we potential energy.

potential energy is associated with the Data Structure as whole, not with individual operation.

(push, pop, multipop)

- Amortized cost of potential method.

i^{th} operation defined by

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

$c_i \rightarrow$ Cost

$D_i - \phi(D_{i-1}) \rightarrow$ change in potential due to operation.

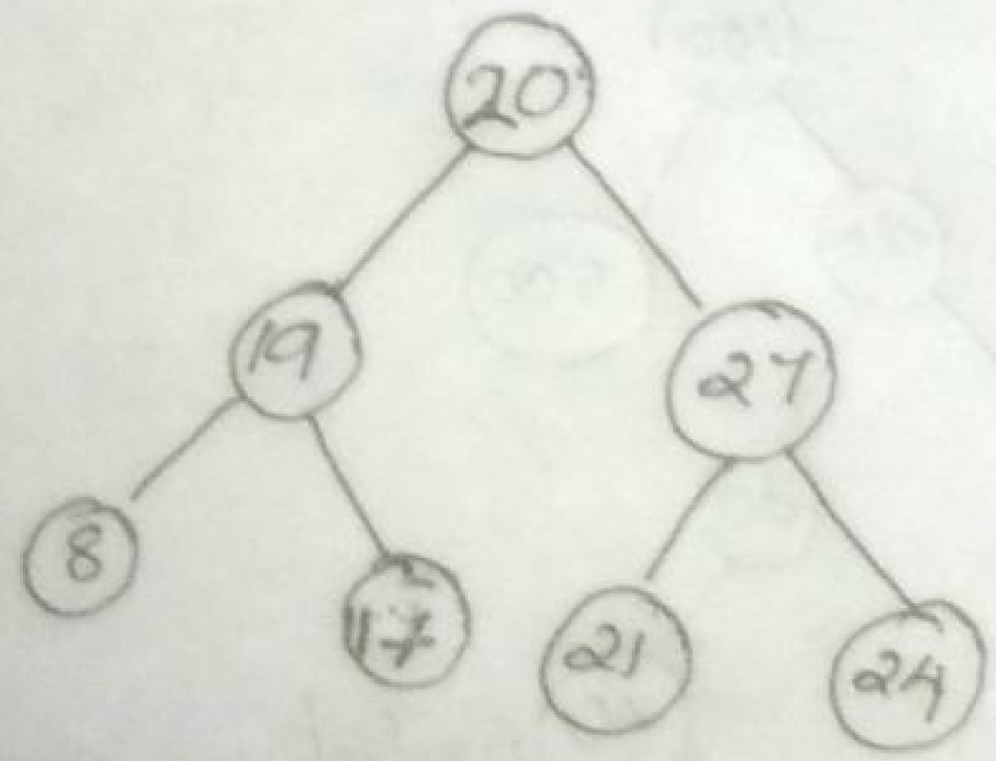
change in potential $= \phi(D_i) - \phi(D_{i-1})$.

(b) Insertion In Binary Search Tree.

Binary Search tree (BST) is a tree which has the following properties

- The value of they key of the left Subtree is less than the value of root
- The value of the key of the right Subtree is greater than or equal to the value of the root.

Example.



Left Sub tree (K) < root (R)
Right Subtree (K) > root (R).

The Basic Operations of Binary Search tree are.

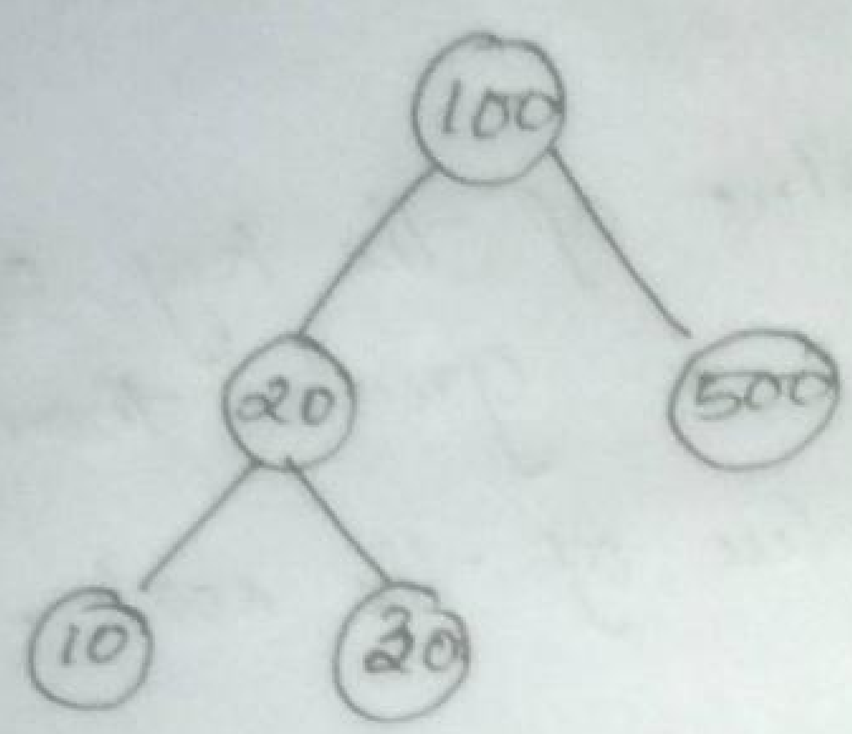
(a) Search — Search element in a tree
(b) Insert — Insert element to a tree.
(c) Delete — Delete element from a tree

⇒ Insert Operation.

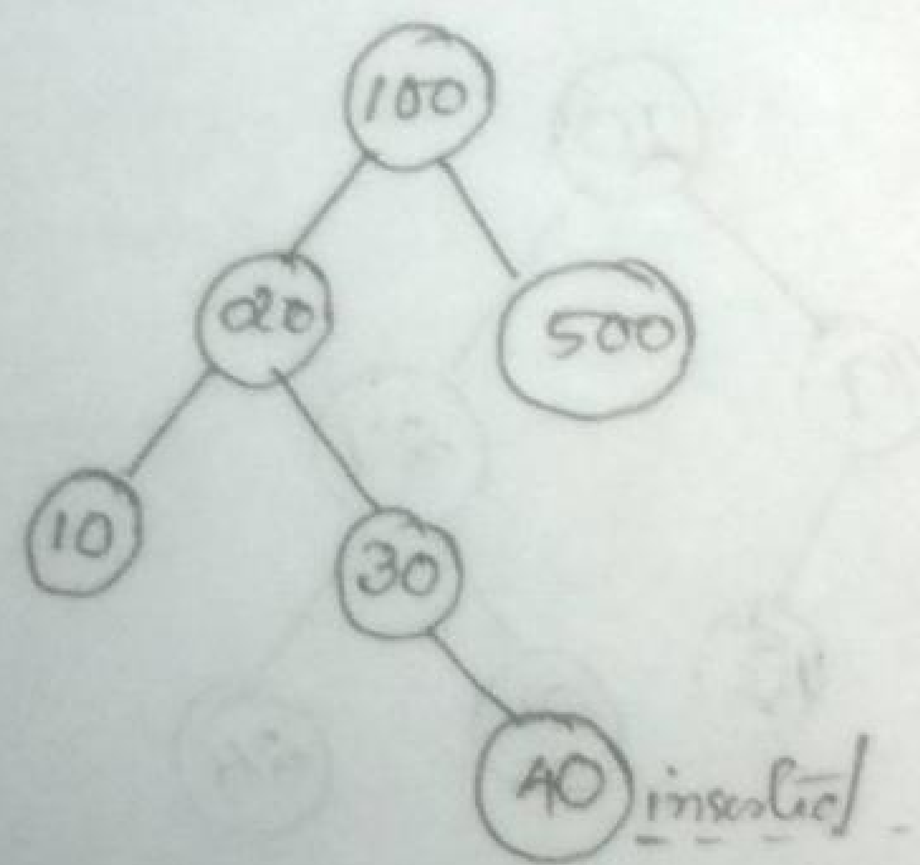
The procedure for inserting an element to the binary Search tree Should bother about the following

- Locate its proper position / location.
- Search from the root node.
  - ↳ k < root → Left
  - ↳ k > root → Right

Consider the given example tree.



Insert 40 into the tree.



Algorithm for BST insertion

```
If node == NULL
return createNode (d).
if (d < node → d)
node → left = Insert (node → left, d);
else if (d > node → d)
node → right = Insert (node → right, d);
return node;
```

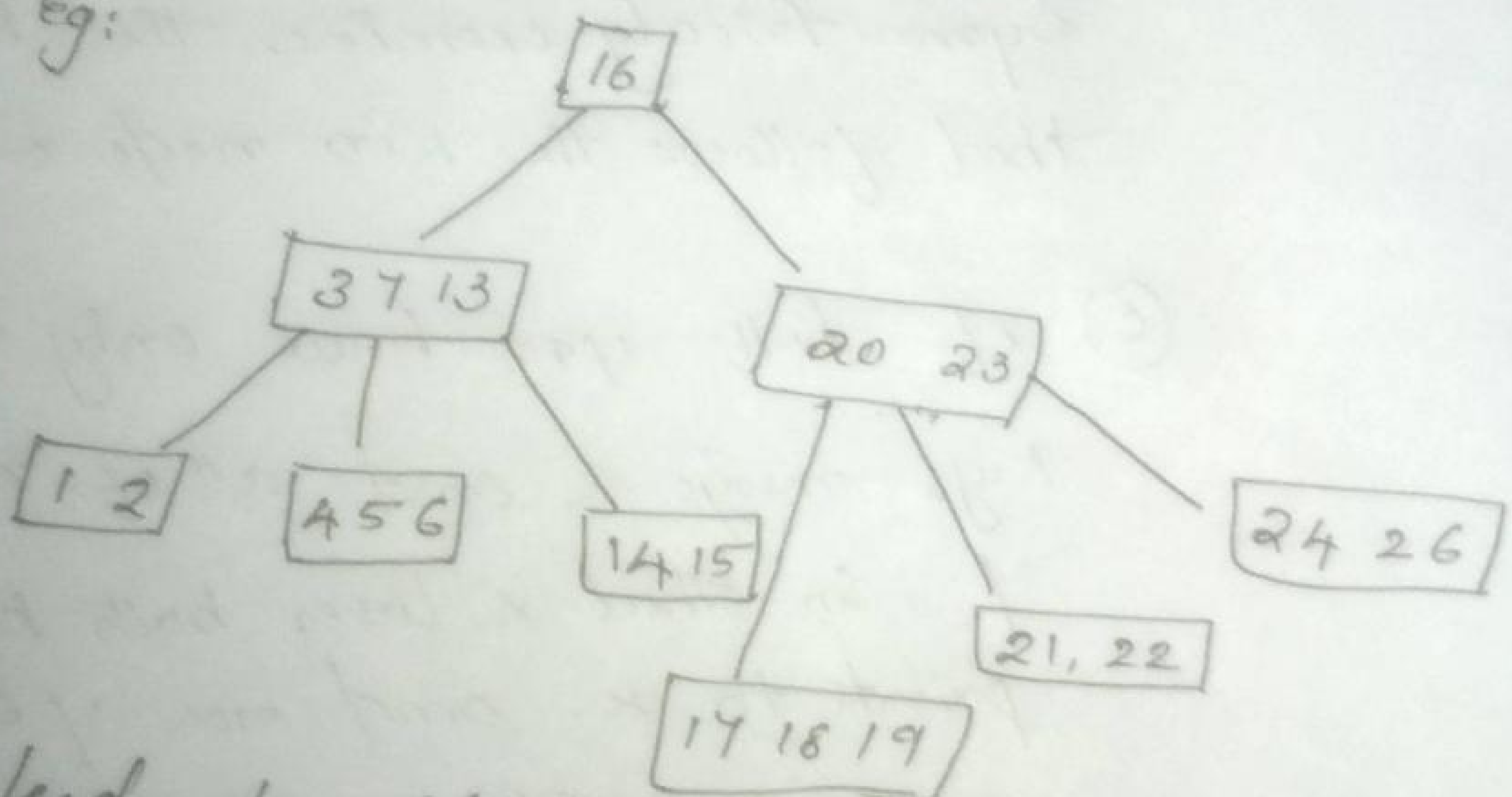
14 (a) Deletion Operation in B-Tree.

For deletion in B-tree we wish to remove from a leaf. There are three possible case for deletion in B-tree.

Case -1 : if the key is already in leaf node.

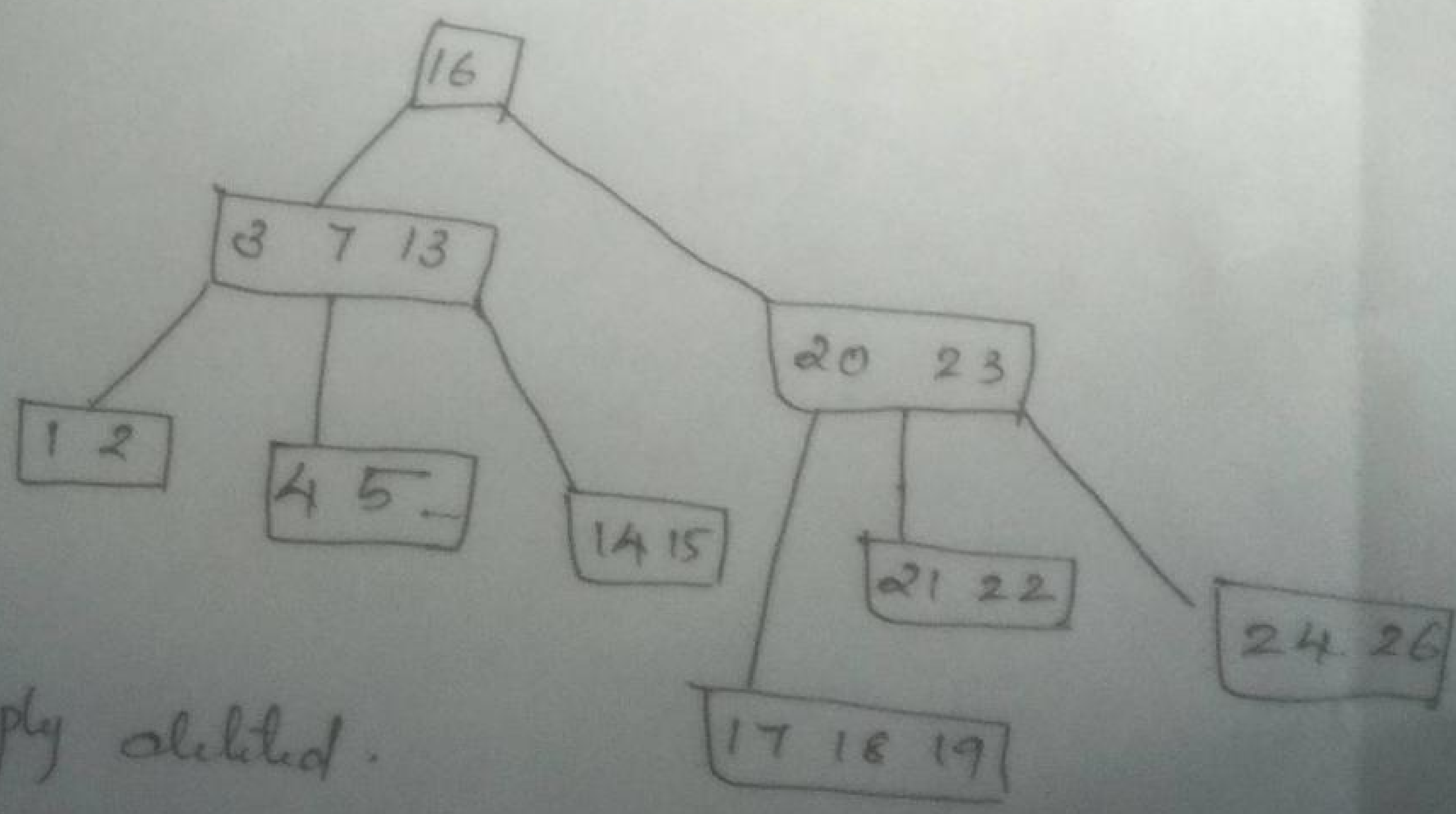
Removing the key from leaf node doesn't cause that leaf node. Then we can simply delet that key.

Eg:



Need to delete $\boxed{6}$

Then



Simply deleted.

Case - 2 : Deletion from Non-leaf.

if key k is in a node, which is an internal node. Then there are 3. cases are there to consider.

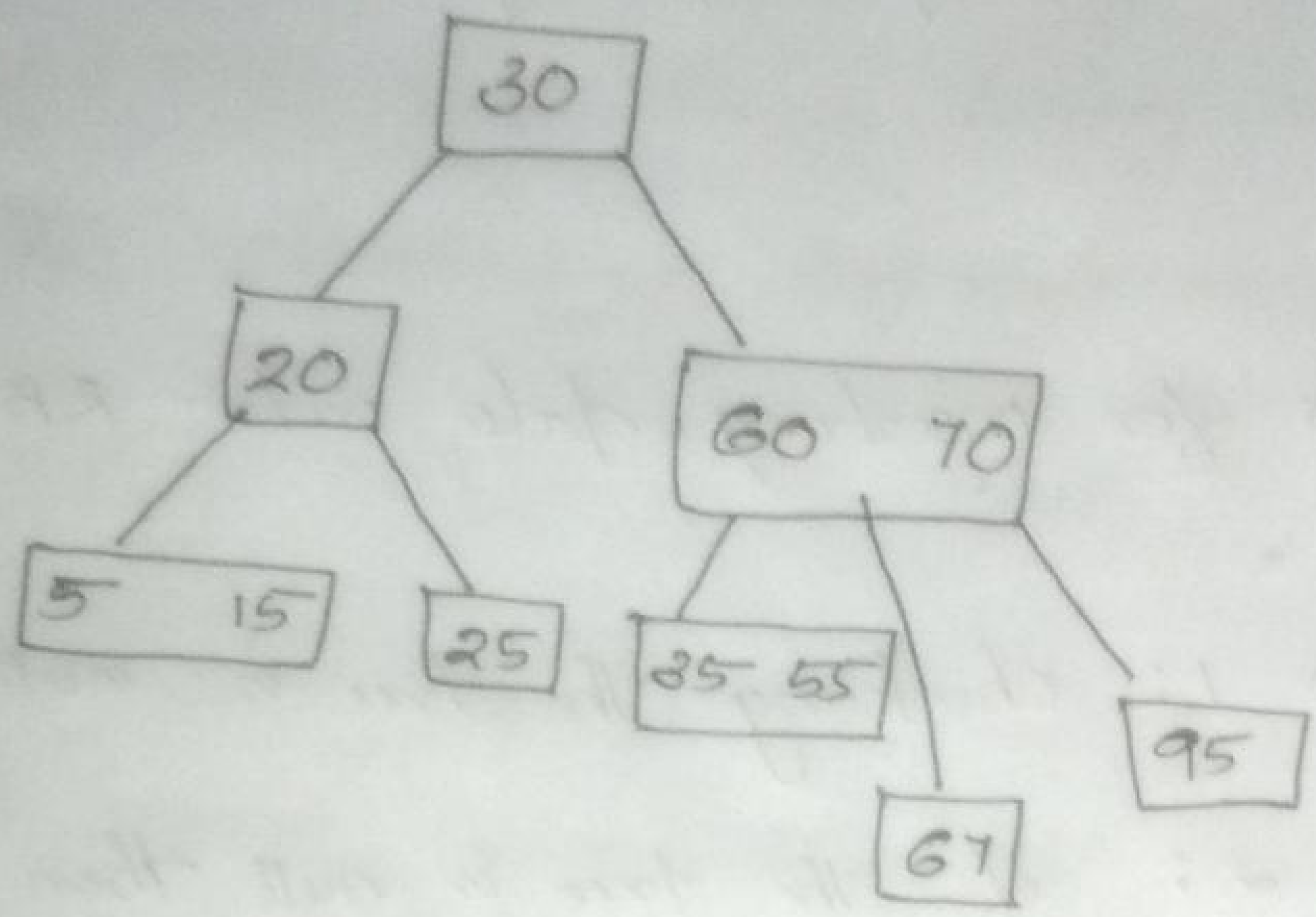
ⓐ if child y that proceeds k in node x at least t keys, then find the predicisser. ko of k in the Subtree

ⓑ if y has fewer than t keys then Symmetrically examine the child z that follows the k in node x.

ⓒ if both y & z have only t-1 keys merge k and all of z into y, so that x loses both k and points to. x. and now y contains 2t-1 keys. Then free z. and recursively delete k from. y.

⑧ B- tree of order - 3.

30, 20, 35, 95, 15, 60, 55, 25, 5,
67, 70.



⑨ Split Operation in b tree insertion

if a node is already full then a new key insertion will overflow and disturb the B tree property. then there arises a process called splitting.

Splitting node. B- tree

1. find the median of the full node.

2. Create a new leaf node and copy into it all the keys which appear after the median

3. Move up the median at an appropriate position in the parent of this node.

4. An additional child pointer from the parent node to the new node.

5. Add new key at the right location in the child nodes of the median.

(10) Steps for inserting data into RB tree.

Step 1: Checking the tree is null or not

Step 2: if the tree is null then insert the new node as Root with black Colour.

Step 3: if tree is not empty insert new node as leaf node with red colour.

Step 4: if parent of new node is black then operation cant performed.

Step 5: if parent of new node is Red then check colour of parent nodes Sybling of new node.

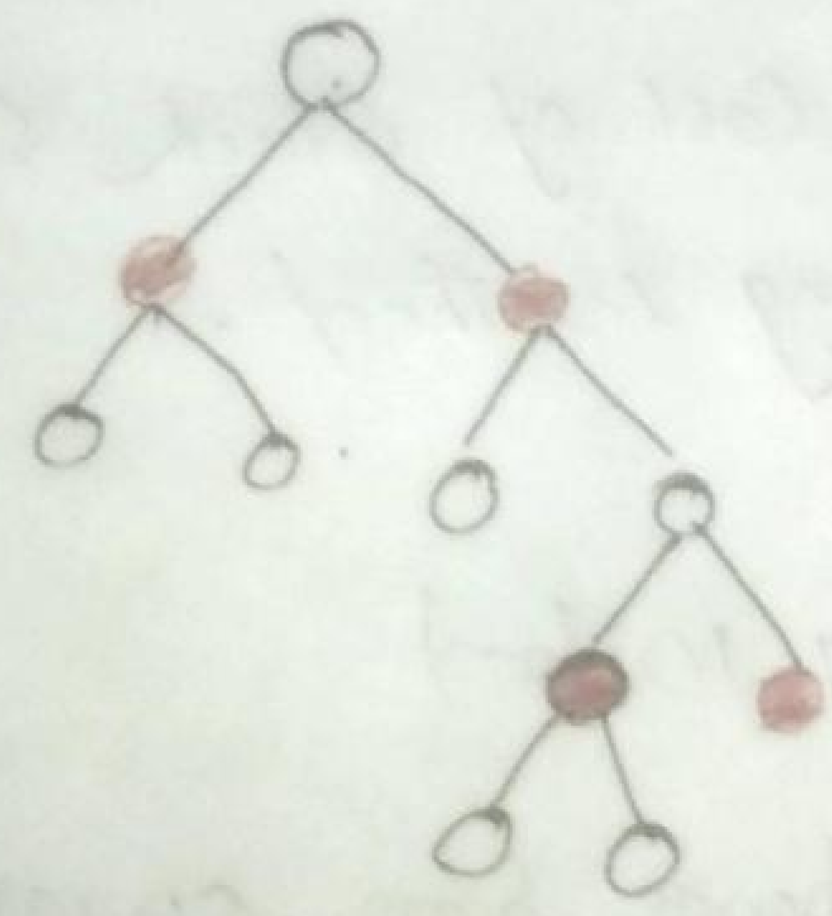
Step 6: if it is Black or Null then make rotation or recolour it.

Step 7: if its Colors is red then perform Recolour Until the tree become Red black tree.

Scanned by TapScanner

④ properties of Red Black tree.

- Root node must be in black colour.
- Every leaf must be coloured black.
- RB tree must be a binary tree.
- The childreen of Red node must be Black.
- Every inserted node must be Red.
- In all path of the tree there should be same number of black coloured nodes.



⑤ Disjoint Set

A disjoint data structure maintains a collection $S = \{S_1, S_2, \dots S_k\}$ of disjoint dynamic set.

$S_x = \{3, 4, 5, 6, 7\}$

$S_y = \{1, 2\}$.

disjoint Set Operation

1. MAKESET(x) → Create new set whose only member is pointed by x.

2. UNION(x, y) → Merge two sets. $S_x \cup S_y$.

3. Find Set(x) → Returns a pointer to the representation of unique Set Containing x

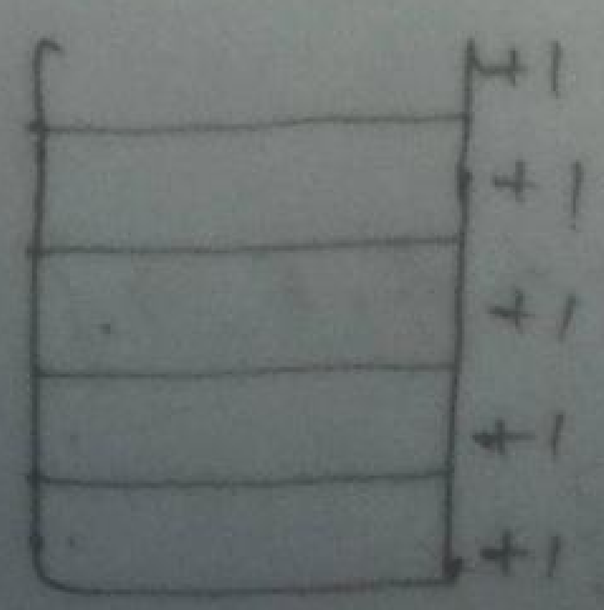
(4) Amortized Cost of Stack Operation Using Accounting Method.

Accounting Method

↳ over charge some operations early and use them as to prepaid charge later.

↳ The amount we charge to an operation is called Amortized Cost

Eg: perform 5 PUSH operation on stack.



Credit = Amortized Cost − Actual Cost

= 10 − 5

= 5

③ Binary Search tree.

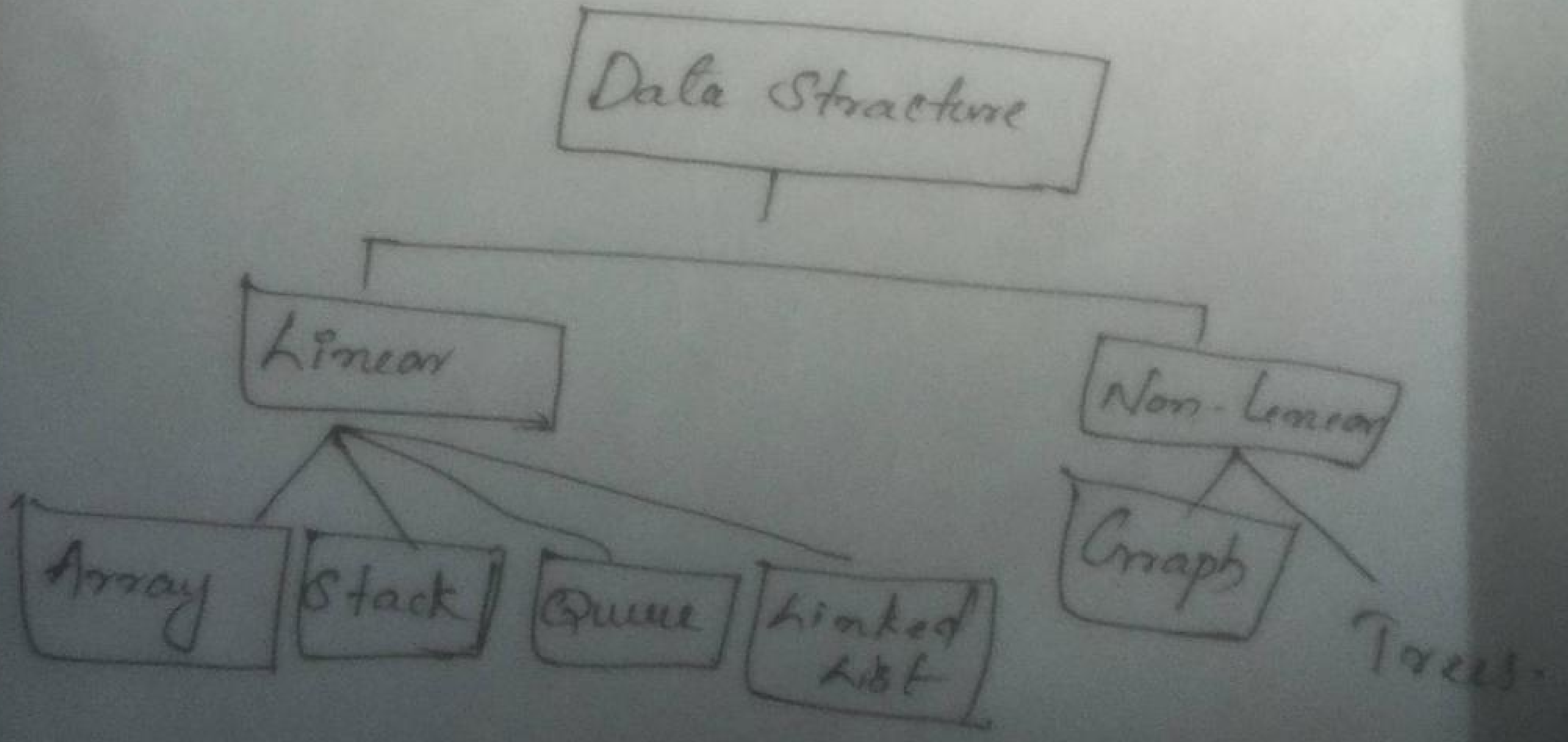
A binary Search tree is a binary tree that may be mpty and if it is not empty then it have the following property.

1. All the keys in the left Subtree of root is less thant root key.

2. All keys in the right Subtree of root is great than root

3. Left and right Subtree are also binary Search tree.

① Linear and Non Linear.



Linear — In this Data Structure the elements are organised in a Sequence such as array.

Non Linear — In this data structure data is Organised without any Sequence
Eg: tree.

Scanned by TapScanner