Assignment # 05

Data Structures and Algorithms

Submitted to:

Mam. Afifa Hameed

Submitted by:

Ansir Ali
L1F23BSSE0388

Section:

P4

Assignment # 05

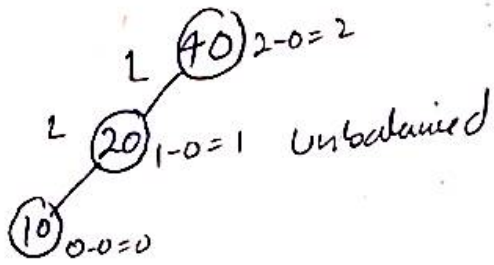
Scenario 1:
Student ID Management System:-

(a)

Insert they given students IDs into an initially empty AVL Tree:

40, 20, 10, 25, 30, 22, 50, 60

step 1:

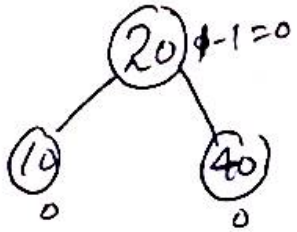


Unbalanced

BF
Balance factor
height of left subtree − height of right subtree

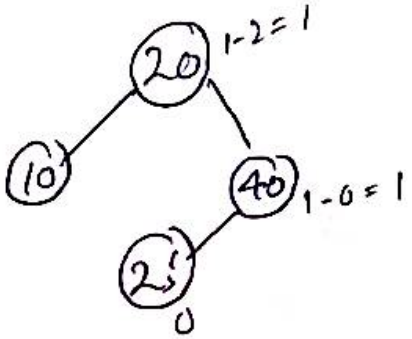
[-1, 0, 1]

⇒ Apply Left Left Rotation to balance this



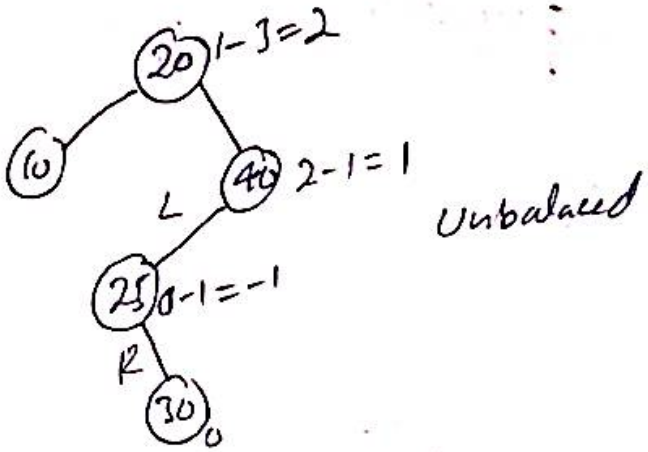
balanced.

step 2: insert 25

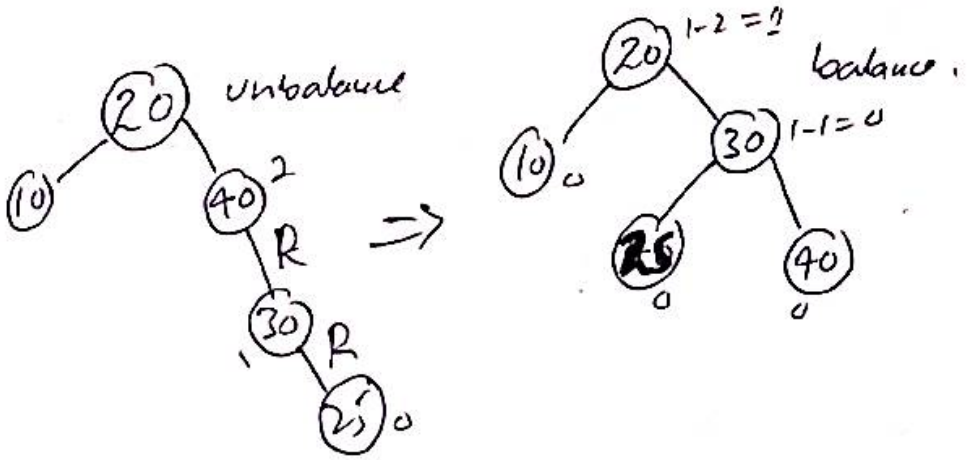


balanced.

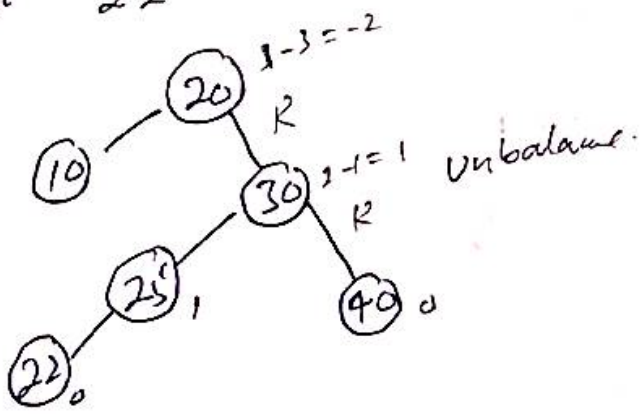
Step 3: insert 30



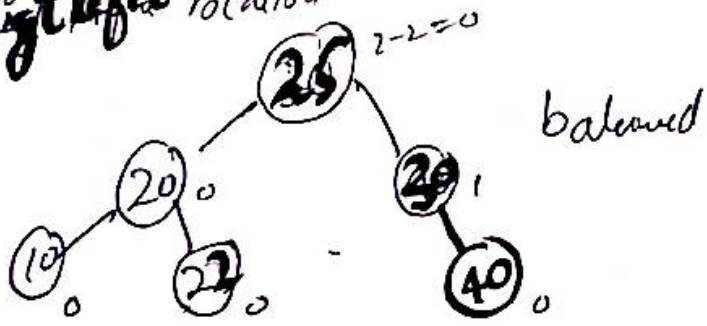
⇒ Apply left Right rotation to balance this



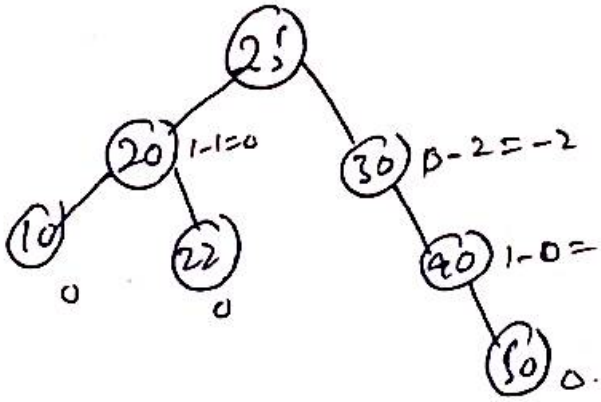
Step 4: insert 22



⇒ Apply Right left rotation

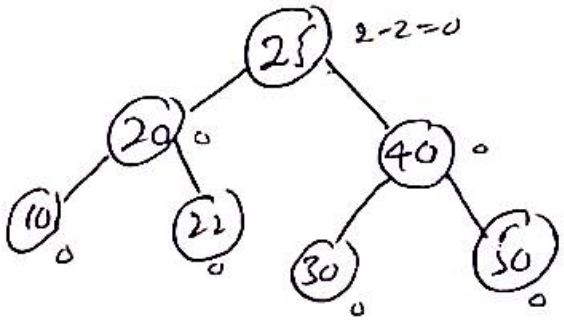


Step 5: insert 50



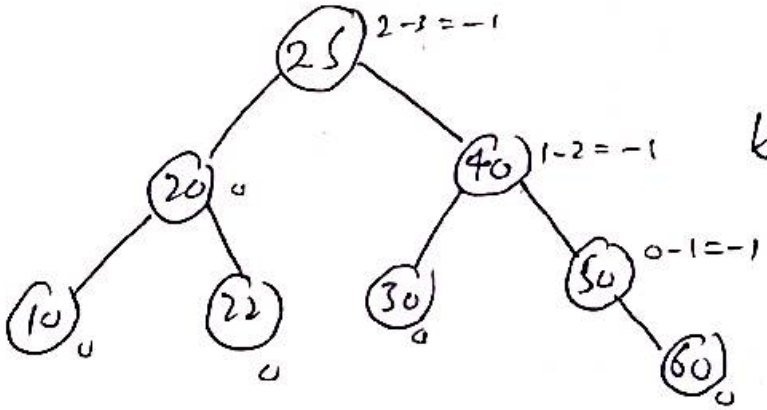
Unbalanced.

=> Apply Right Right rotation



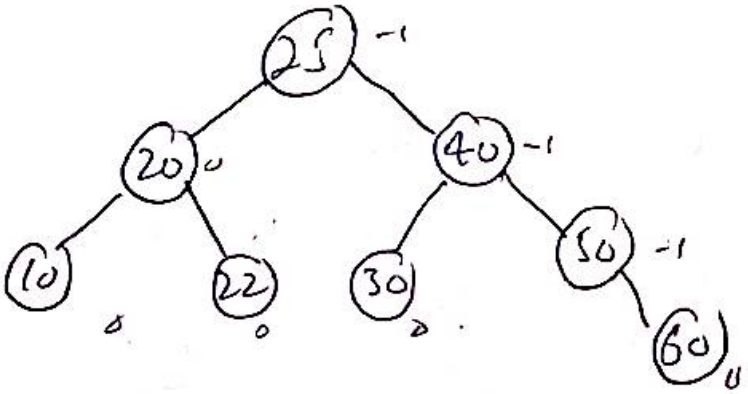
balanced.

Step 6: insert 60



balanced.

Final balanced tree:

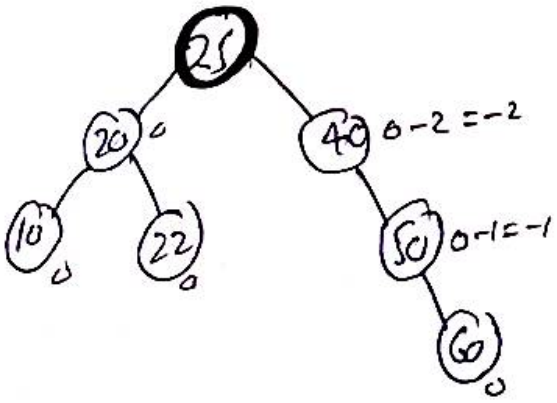


(b)

Now delete the following student IDs from your tree:

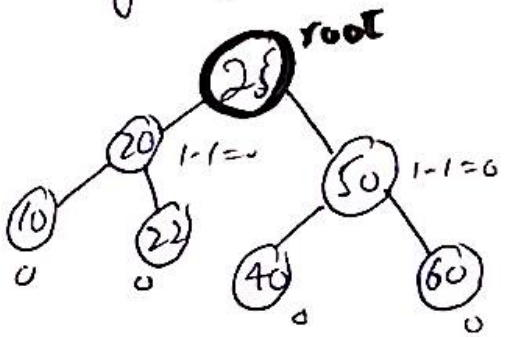
30, 40

Step 1: Delete 30,



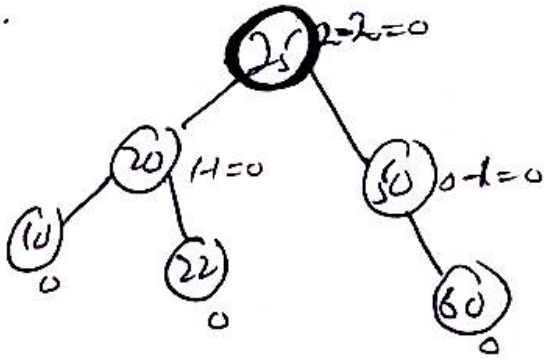
⇒ 30 is child node/ leaf node, so it remove directly:

After deletion Tree is unbalanced, so we balance it by Right Right rotation



Step 2: Delete **40**

Node 40 is a leaf node, so delete it directly.



So, it is balance after deletion

(C)

Perform search operations on the AVL tree you obtained after Part B.

25, 40, 60

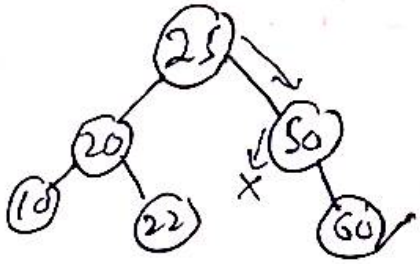
Step 1:

Search for ID = 25

⇒ 25 is a root node, so it found in AVL Tree

Path:

Start at 25



Step 2:

Search for ID = 40

⇒ 40 > 25, so go right side

⇒ 40 < 50, so go left of 50

Not found

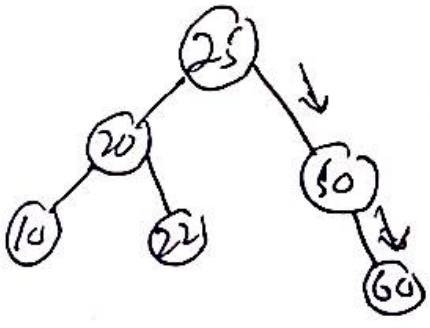
Step 3:

Serach for ID = 60

⇒ Start from 25, 60 > 25, go right

⇒ 60 > 50, go right, it matched

Found.



Scenario 2:

Store unique Book IDs in Library System:

(a)

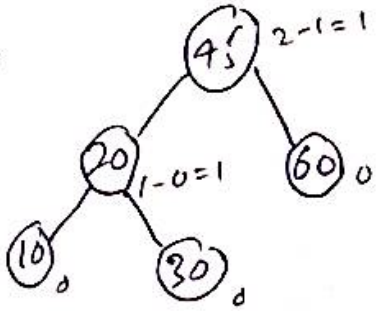
Insert the following Book IDs into an empty AVL Tree:

45, 20, 60, 10, 30, 25, 35, 50, 70, 65, 80, 55

Step 1:

Take root node 45:

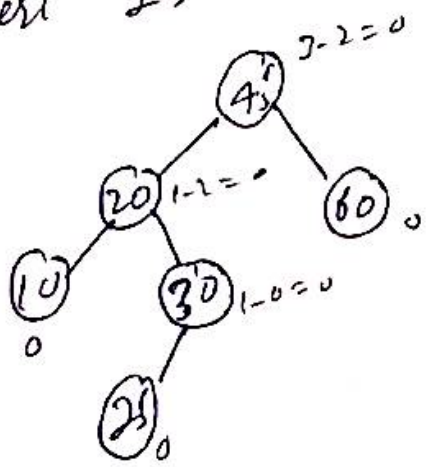
Insert 45, 20, 60, 10, 30



balanced.

Step 2:

Insert 25

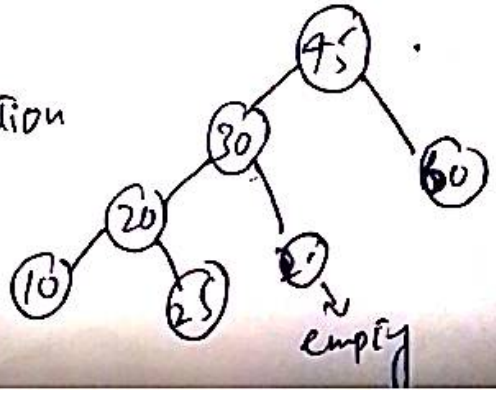


Unbalanced.

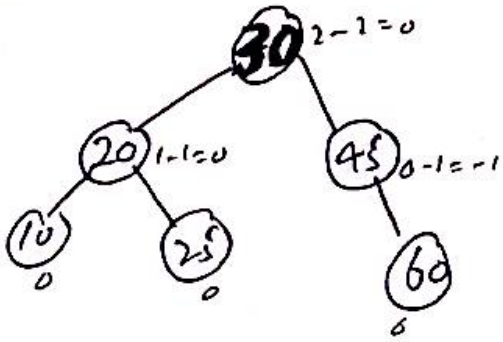
⇒ Apply Left Right rotation

To balance this

- First by rotate



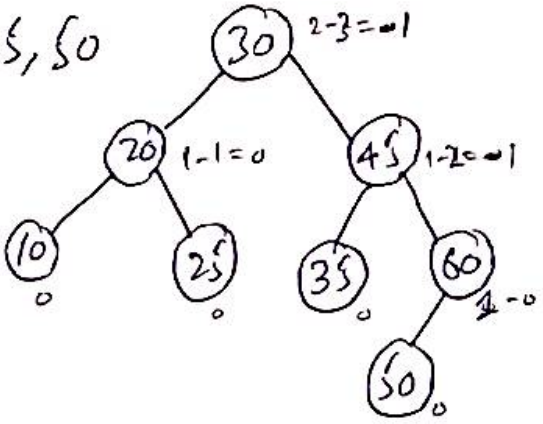
1 • Right rotate



So, it is balance, go next insertion.

Step 3:

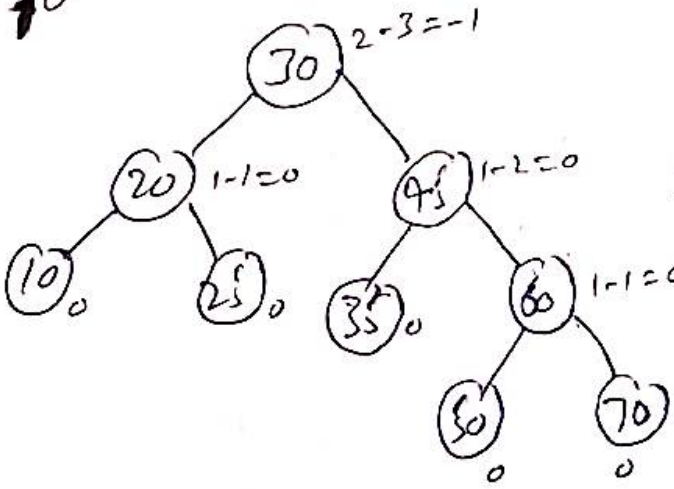
insert 35, 50



balanced.

Step 4:

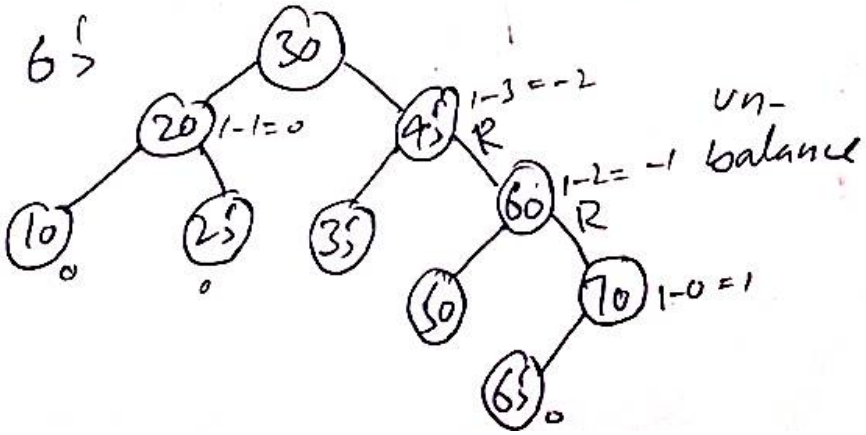
insert 70



balanced

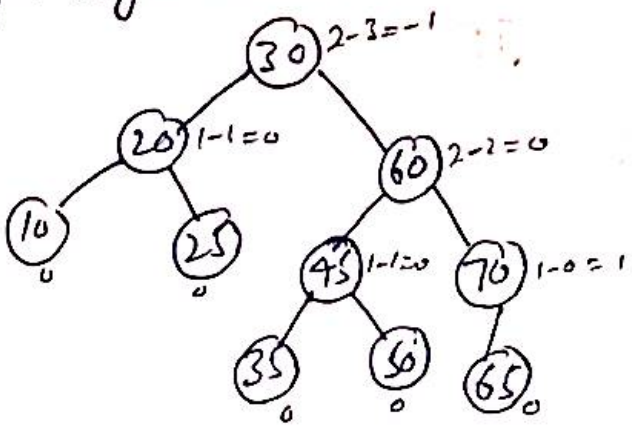
Step 5:

insert 65



un-balance

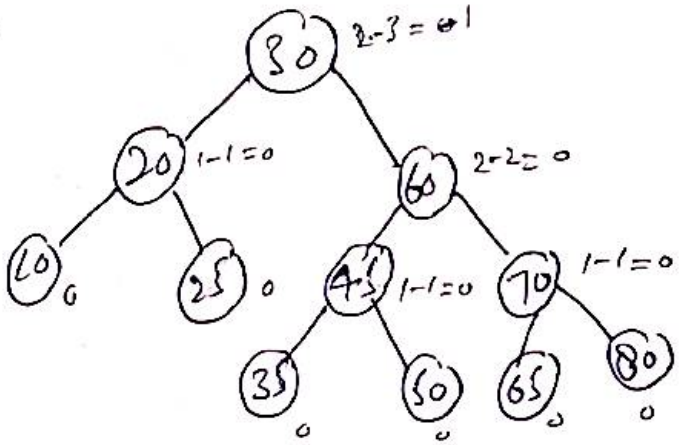
⇒ Apply Right Right rotation..



balanced.

Step 6:

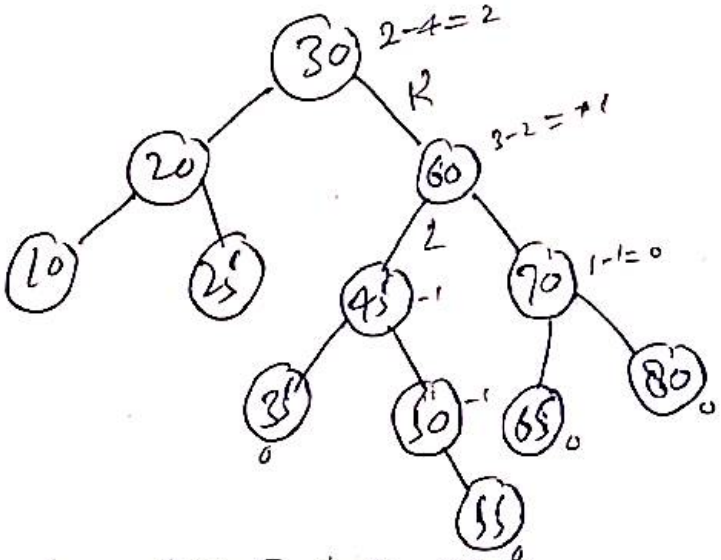
insert 80



balanced.

Step 7:

insert 55

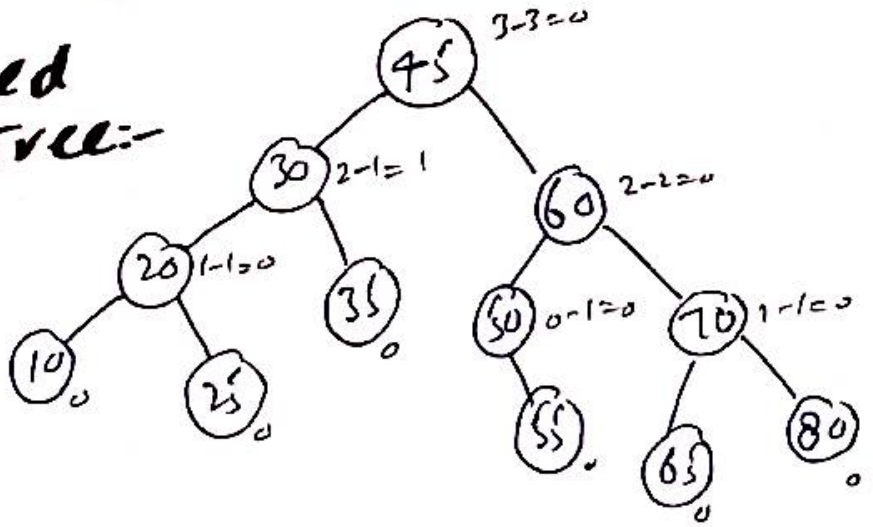


Unbalanced

⇒ Apply Right left rotation

to make the AVL balanced.

9. **Final balanced AVL Tree:-**



(b)

How delete the following Book IDs from the AV tree.

10, 30, 45

Step 1:

Delete 10.

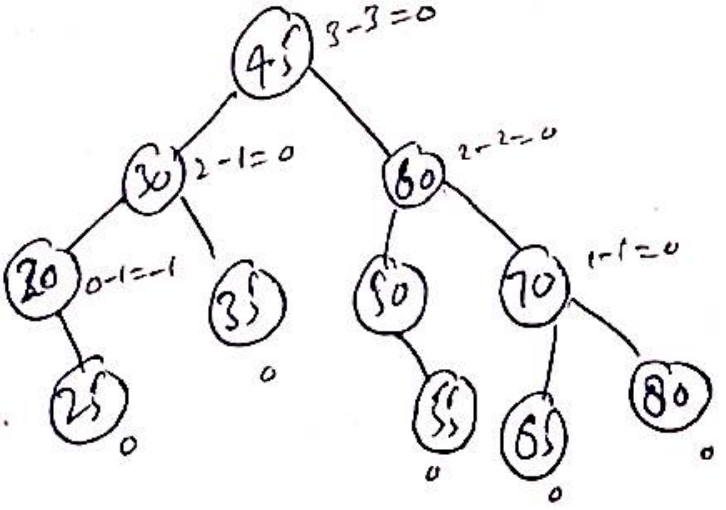
⇒ 10 - is leaf, so delete directly.

10 < 45, go left

30 > 10, go, left

20 > 10, go left then

10, found.



∴ AVL Tree is also balanced.

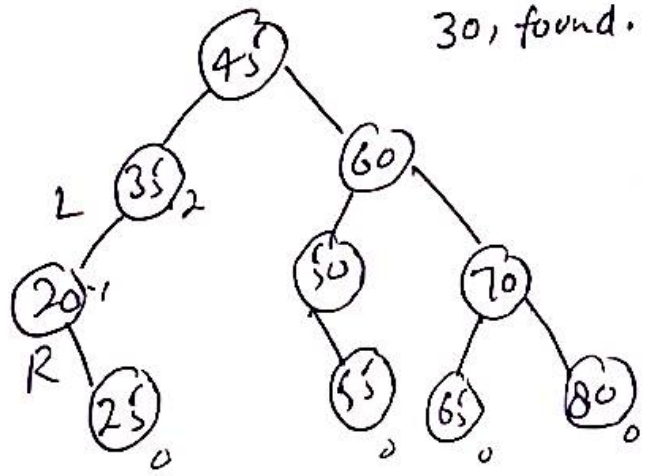
Step 2:

Delete 30,

Replace 30 with 35 and delete 30 (a leaf).

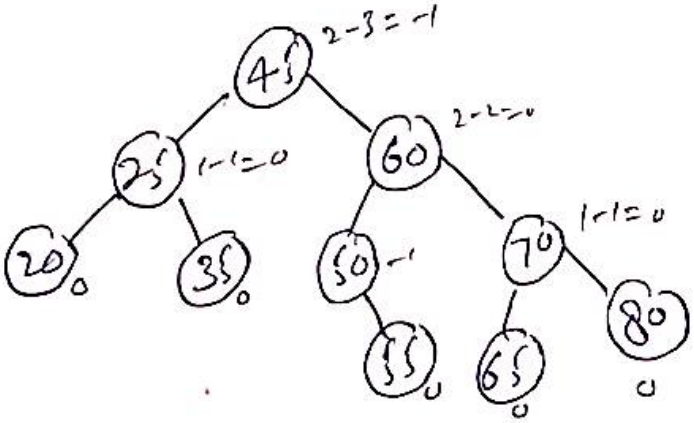
→ 30 24 5, go left

30, found.



After deletion of 30 AVL Tree become unbalance.

⇒ Apply left right rotation to make it balance

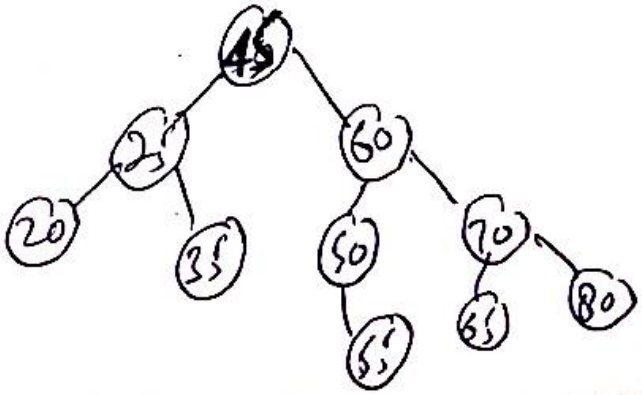


balanced.

Step 3:

Delete 45,

Use in-order predecessor = 35 replaced with 45 and delete 45

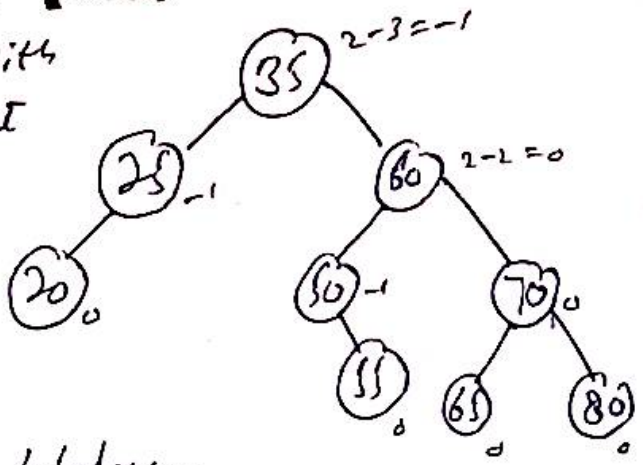


Now, we replace.

11

Final Tree:

4$, replaced with predecessor 3$ right most node:-



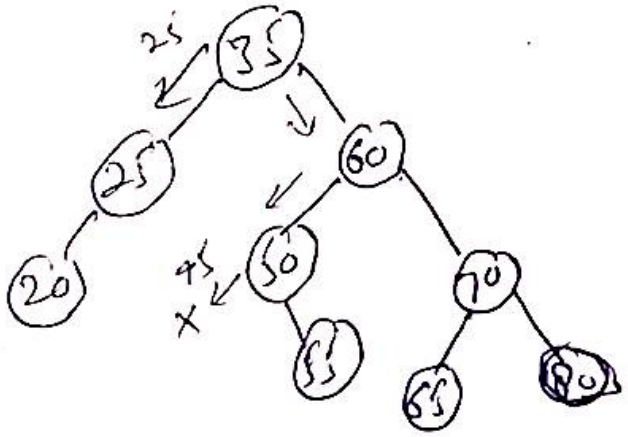
Tree is balance.

(C)

Now, search the following Book IDs from AVL tree.

2$, 4$, 5$, 100

Step 1: Search 2$.



start from root 3$

⇒ 2$ < 3$ go left then

2$ matches so, found

2$ node.

Step 2: Search 4$,

4$ > 3$, go right then

60 > 4$, go left then

50 > 4$, go left

no matches, not found.

Step 3:

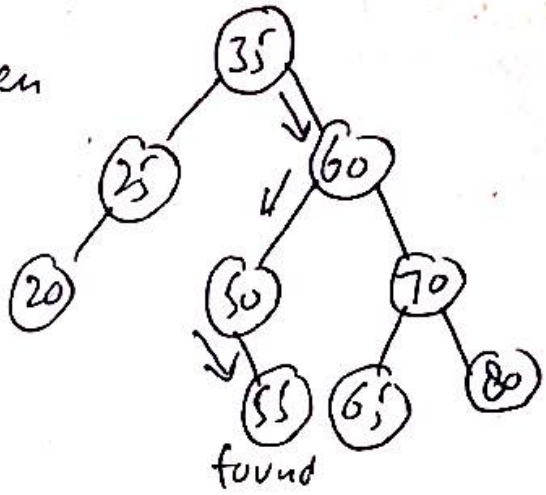
Search 55

55 > 35, so go right then

55 < 60, go left then

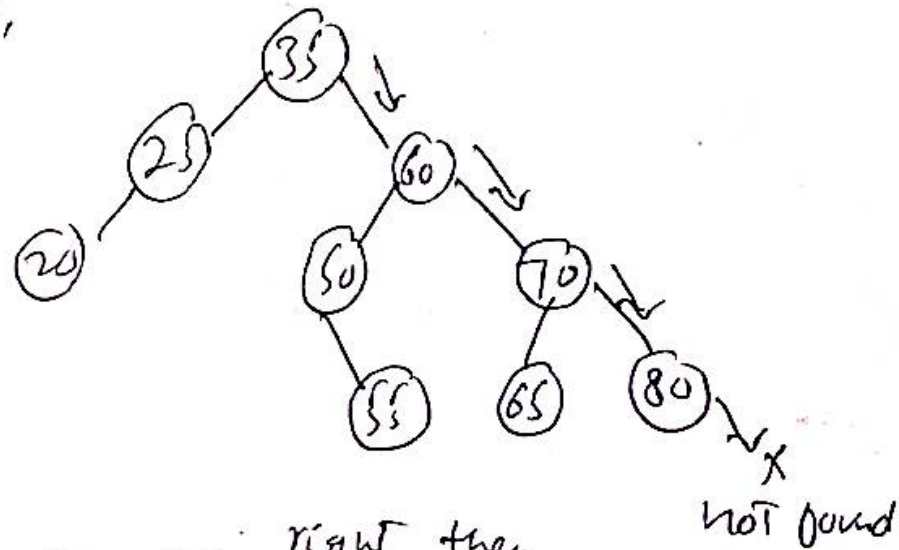
55 > 50, go right

it matches found.



Step 4:

Search 100,



100 > 35, so go right then

100 > 60, go right then

100 > 70, go right then

100 > 80, go right

no matches, NOT found.

Scenario 03:

Building a smart parking System and assign slot of Each car:-

Table size = 10 ⇒ means 10 slots available

h (license-plate) = license-plate % 10

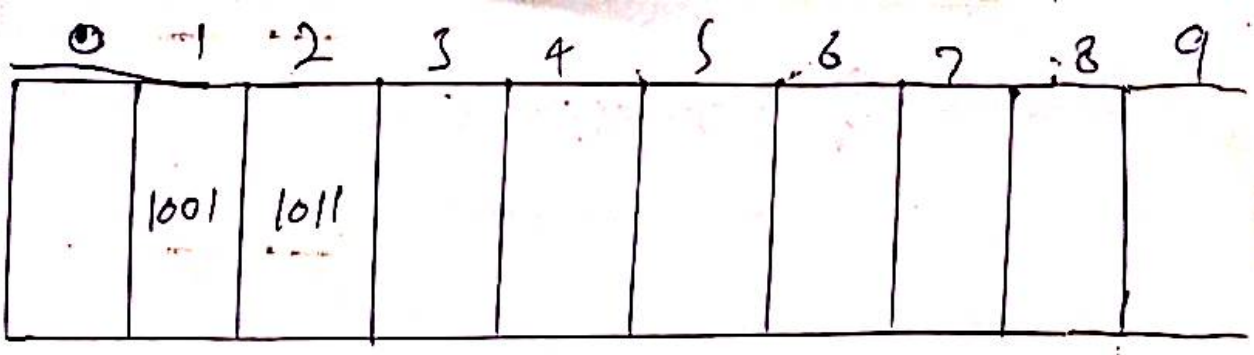
(a)

You decided to first implement linear Probing to resolve collisions.

Insertion in hash table using linear Probing ⇒ 1001, 1011, 1021, 1002, 1003

1. Show the hash table after insertion. 1001 % 10 = 1

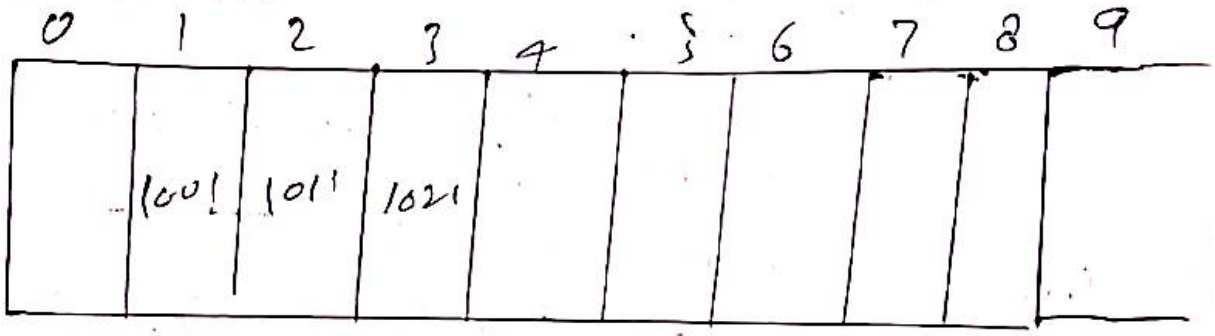
0	1	2	3	4	5	6	7	8	9
	1001								



1011 % 10 = 1

collision occurs so we use

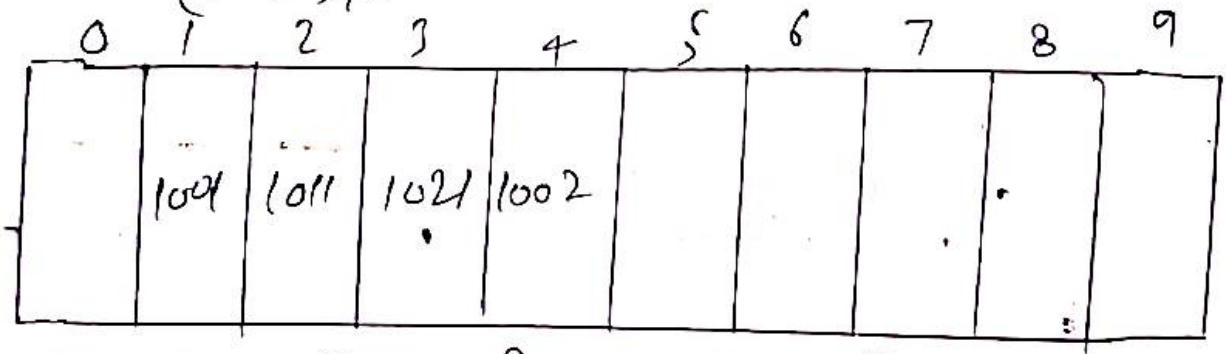
Linear probing. (1 + 1) % 10 = 2, no collision put in table



1021 % 10 = 1 , collision at 1

(1+1) % 10 = 2 , collision at 2

(1+2) % 10 = 3 , no collision, put in table



1002 % 10 = 2 , collision at 2.

(2+1) % 10 = 3, collision at 3

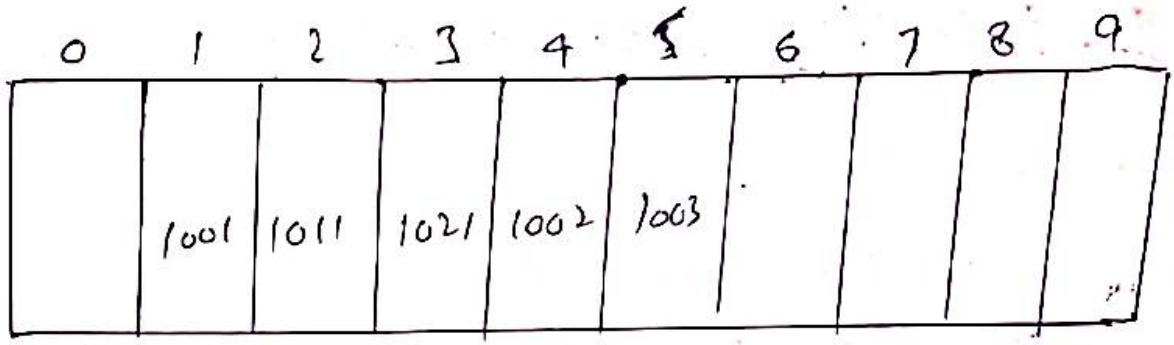
(2+2) % 10 = 4, no collision, put in table

15

1003 % 10 = 3 , collision occurs

(3+1) % 10 = 4 ; collision at 4.

(3+2) % 10 = 5 ; no collision, put in table



2. Check collision points:-

- 1011 collision at slot 1
- 1021 collision at sot 1, 2
- 1002 collision at 2, 3
- 1003 collision at 3, 4 slots.

3. what slot does 1003 end up in?

At the slot 5 index, 1003 end up.

4. Search for 1021

slot 1 , not match,

slot 2, not match

slot 3 , match found

Path : 1 → 2 → 3

(b)

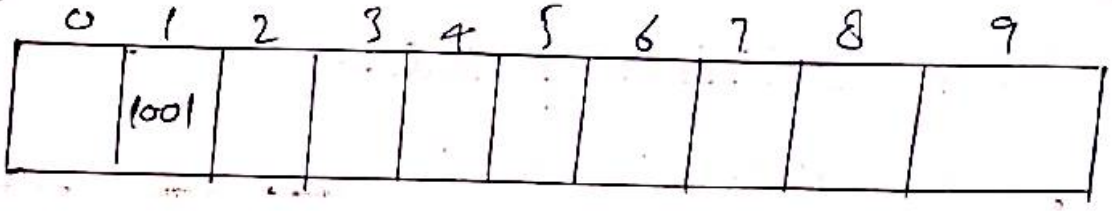
Quadratic Probing :-

Formula: $(h(k) + i^2) \% \text{size of table}$.

1. insertion step-by-step

- insert **1001**

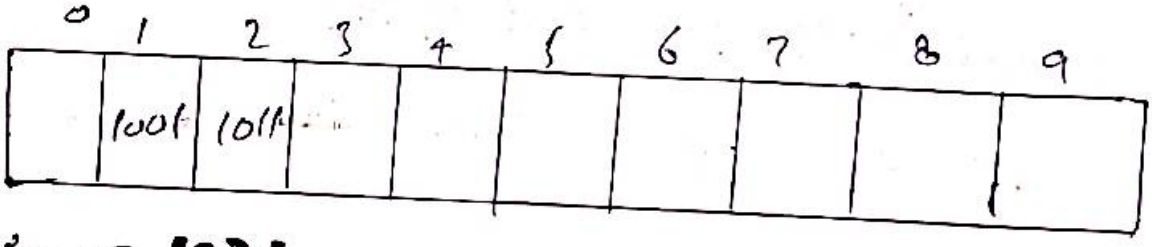
$1001 \% 10 = 1$, no collision.



- insert **1011**

$1011 \% 10 = 1$, collision occur.

$(1 + 1^2) \% 10 = 2$, no collision.

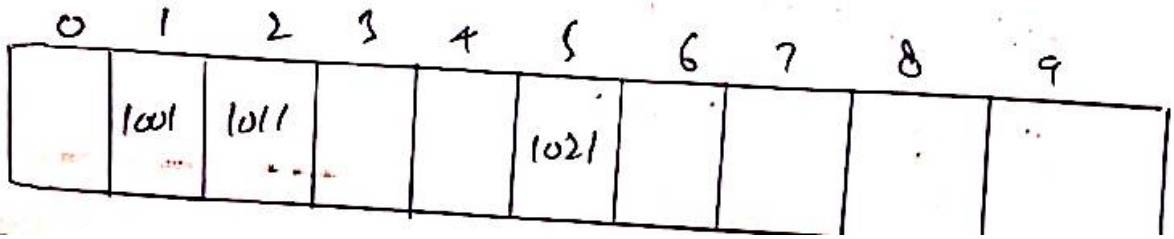


- insert **1021**

$1021 \% 10 = 1$, collision occur

$(1 + 1^2) \% 10 = 2$, collision occur at 2.

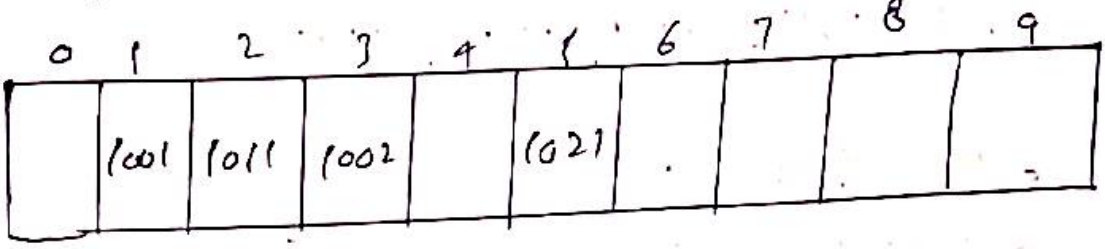
$(1 + (2)^2) \% 10 = 5$, no collision put in table



17 ⊙ Insert 1002

1002 % 10 = 2, collision occurs at 2.

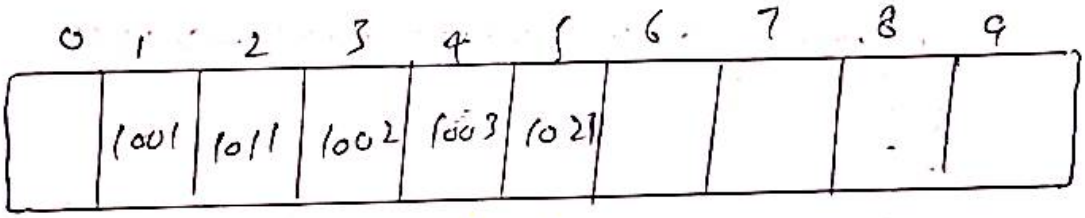
(2 + i) % 10 = 3, no collision. put in table.



⊙ insert 1003

1003 % 10 = 3, collision at 3

(3 + i) % 10 = 4, no collision put in table.



2. final slot for 1003

At the slot 4 index, 1003 end up.

3. probs for 1021

⊙ linear probing, 3 probs 1 → 2 → 3

⊙ Quadratic probing:

1 − 2 − 5, 3 probs as well.

⇒ same number of probes, but Quadratic spread data.

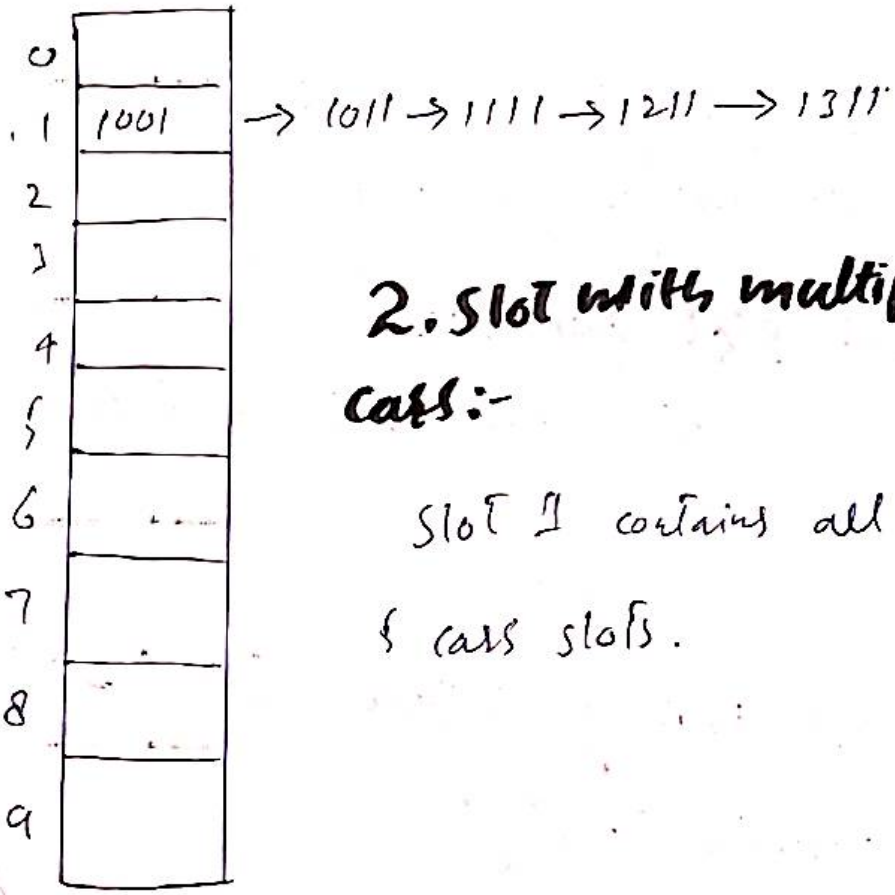
Separate chaining:-

1. insert these license plates:-

1001, 1011, 1111, 1211, 1311

- 1001 % 10 = 1
- 1011 % 10 = 1
- 1111 % 10 = 1
- 1211 % 10 = 1
- 1311 % 10 = 1

All map to slot 1.



2. Slot with multiple cars:-

Slot 1 contains all 5 cars slots.

3. **Remove 1211**

we simply unlink the node with 1211 from the list.

New list at slot 1

1001 → 1011 → 1111 → 1311

4. Search 1311

- Hash = 1
- Traverse list

1001 → 1011 → 1111 → 1311 (found)

Path.

indexes

1 → 2 → 3 → 4

steps in the list

Scenario 04:

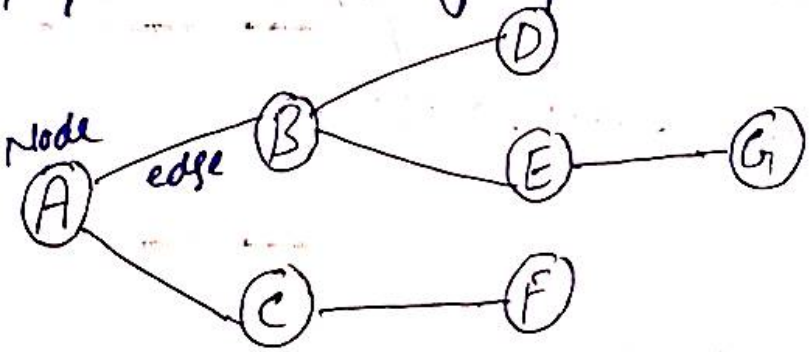
University campus connected by walking paths using graph:

Building (Node) = A, B, C, D, E, F, G

Paths (edges) = A-B, A-C, B-D, B-E, C-F, E-G

(a)

Represent the graph:-



- **Adjacency List:-**

Node	Adjacent node
A :	B, C
B :	A, D, E
C :	A, F
D :	B
E :	B, G
F :	C
G :	E

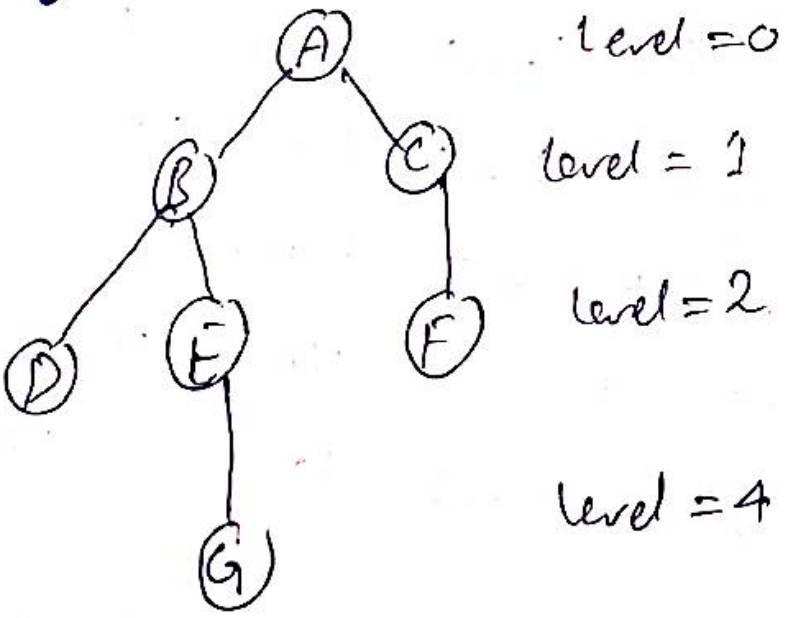
⊙ Adjacent Matrix

⇒ (A,A) means self loop.

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	0	1	1	0	0
C	1	0	0	0	0	1	0
D	0	1	0	0	0	0	1
E	0	1	0	0	0	0	1
F	0	0	1	0	0	0	0
G	0	0	0	0	1	0	0

(b)

Breadth-first Search (BFS)



⇒ BFS uses Queue and visit level by level.

Queue → FIFO (First in First out)

Front	rear
A, B, C, D, E, F, G	

order List = A → B → C → D → E → F → G

Process:

1. Start at A → enqueue A
2. visit A → Enqueue B, C
3. visit B → Enqueue D, E
4. visit C → Enqueue F
5. visit D → no child
6. visit E → Enqueue G
7. visit F → Now child
8. visit G → No new child.

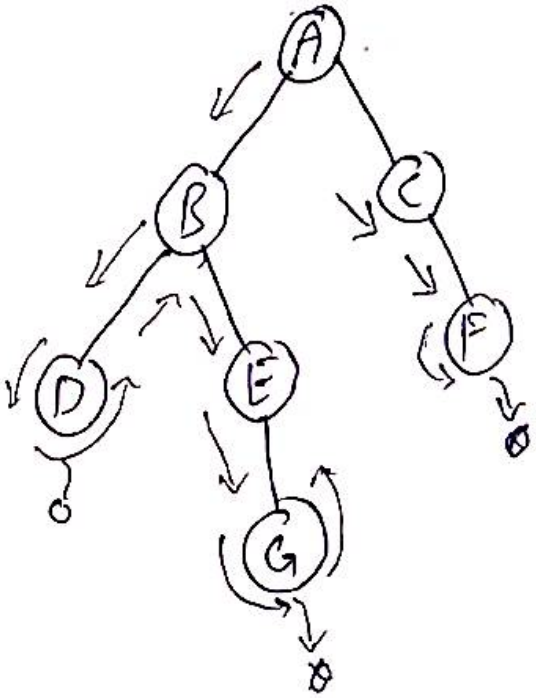
⇒ So, BFS uses Queues as a data structure and Traversal level by level.

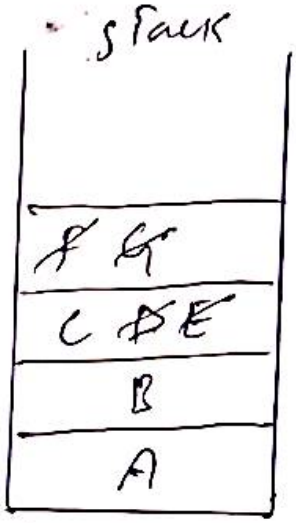
(C)

Depth-First Search (DFS)

DFS uses stack and traverse backtracking.

Stack FILO (First in Last out)





List: D, G, E, B, F, C, A

DFS order

= A → B → D → E → G → C → F

Process:

1. start at A,
2. go to B.
3. go to D
4. D have no leaf node, so, backtrack
5. Go to E
6. Go to G

7. G have no leaf node, back track.

8. B is done → Backtrack to A

9. Go to C

10. Go to F