Data Structure & Algorithms

Sunbeam Infotech

Hash Table – Searching

Hash Table is DS in which data is stored in key-value pair, so that for a given key value can be searched in fastest possible time. Ideal is O(1).

Key-value pair → associative DS

e.g. Mobile → Contacts appln → name (key) ⇒ mobile (value)

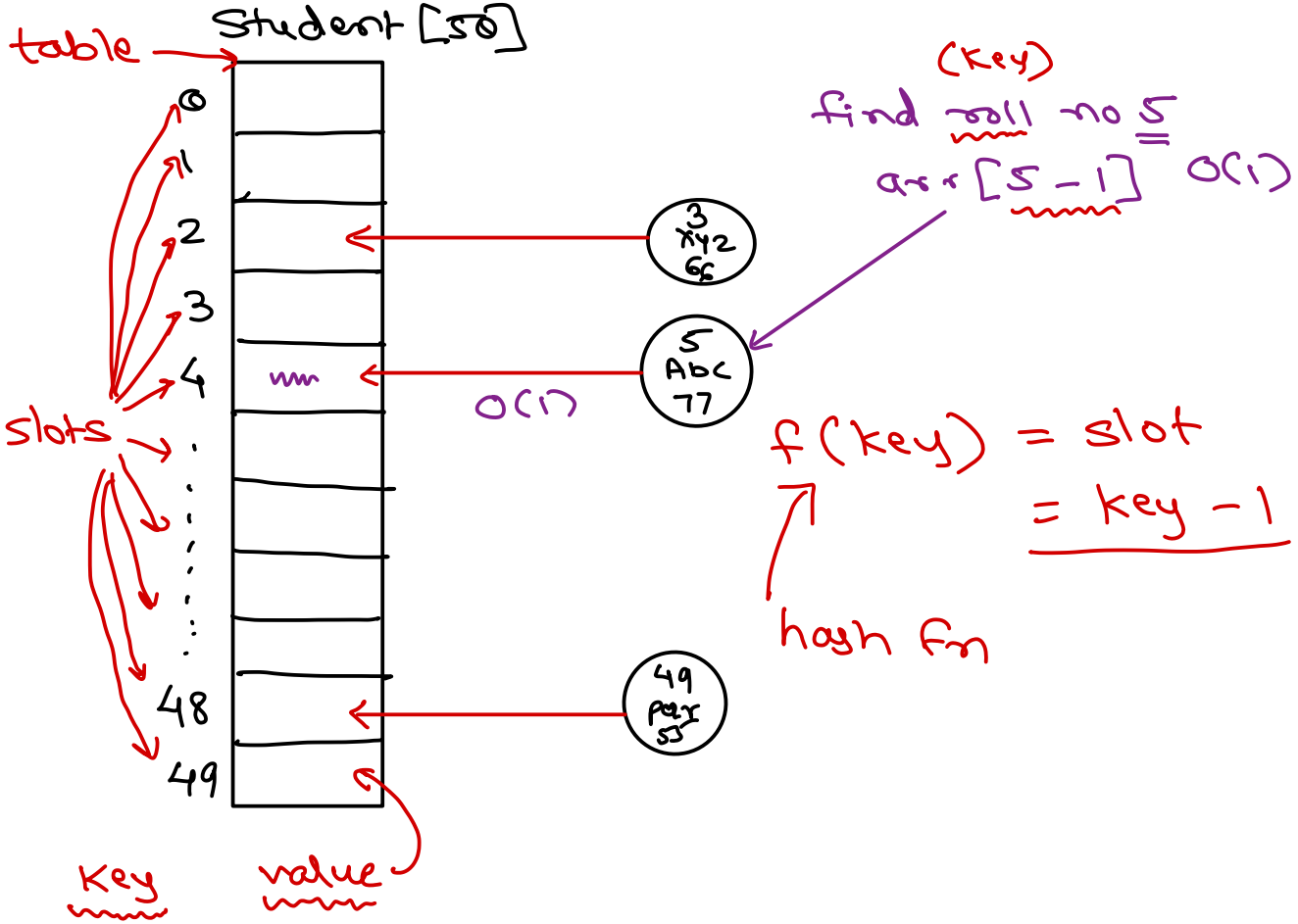
Hash ADT

1. put(key, value)
2. value = get(key)

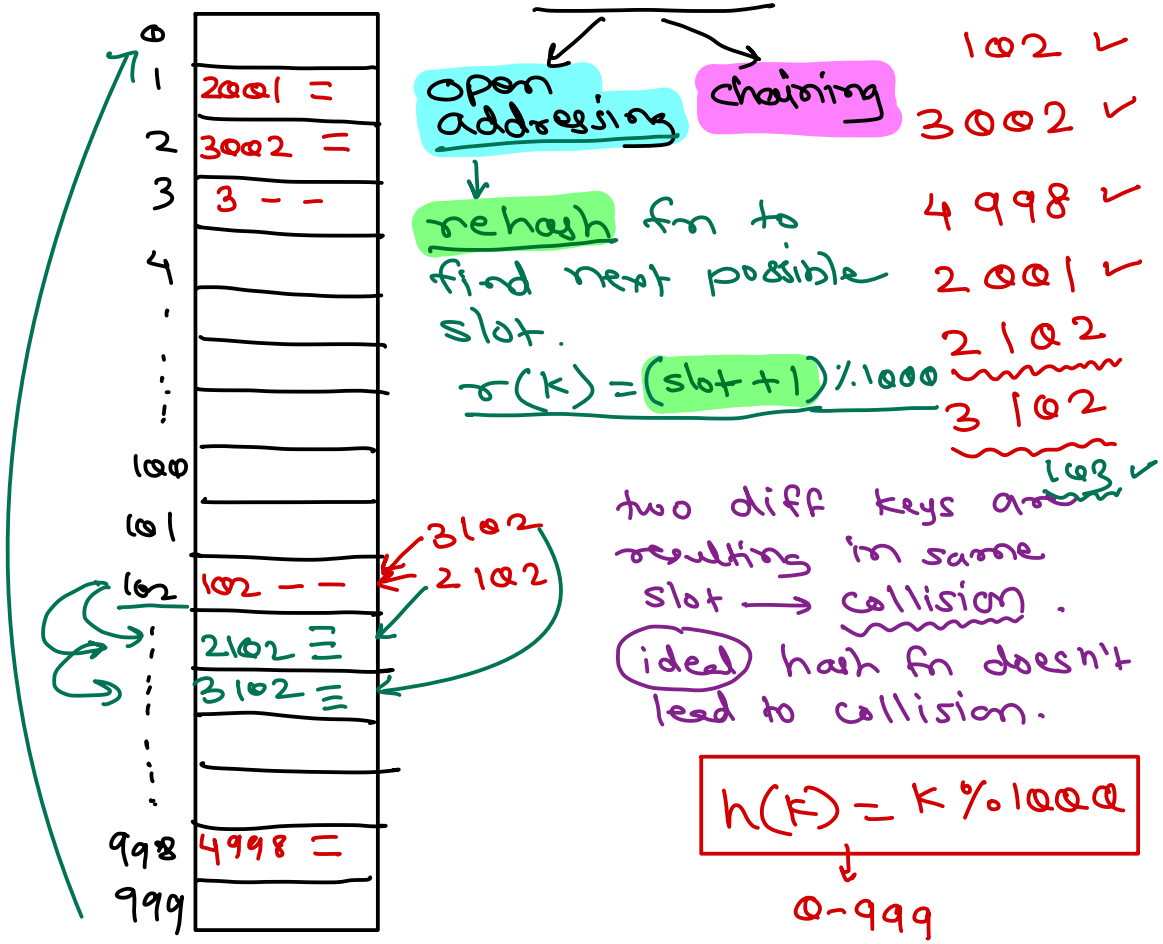
Searching

1. Linear → O(n)
2. Binary → O(log n)
3. ~~Fibbonacci~~
4. Hashing → O(1) ideal

A class have 50 students with roll 1 to 50.
Store students so that, student can be found in fastest way for given roll.



in an alumni meet max 1000 students will come. invitations sent to 5000 students. Store details so that given prn num, student can be found quickly.



Open Addressing

① if collision occur, call rehash fn.

② rehash fn find next possible slot to store/find the element → probing.

③ rehash fn
- linear – $ax+b$
- quadratic – ax^2+bx+c
- polynomial

④ find process:
- ⓐ hash fn → slot
- ⓑ check if ele found in the slot.
- ⓒ if not call rehash fn.
- ⓓ repeat b & c until ele is found.

⑤ open addr mechanism is internal storage. Data stored within array/table.

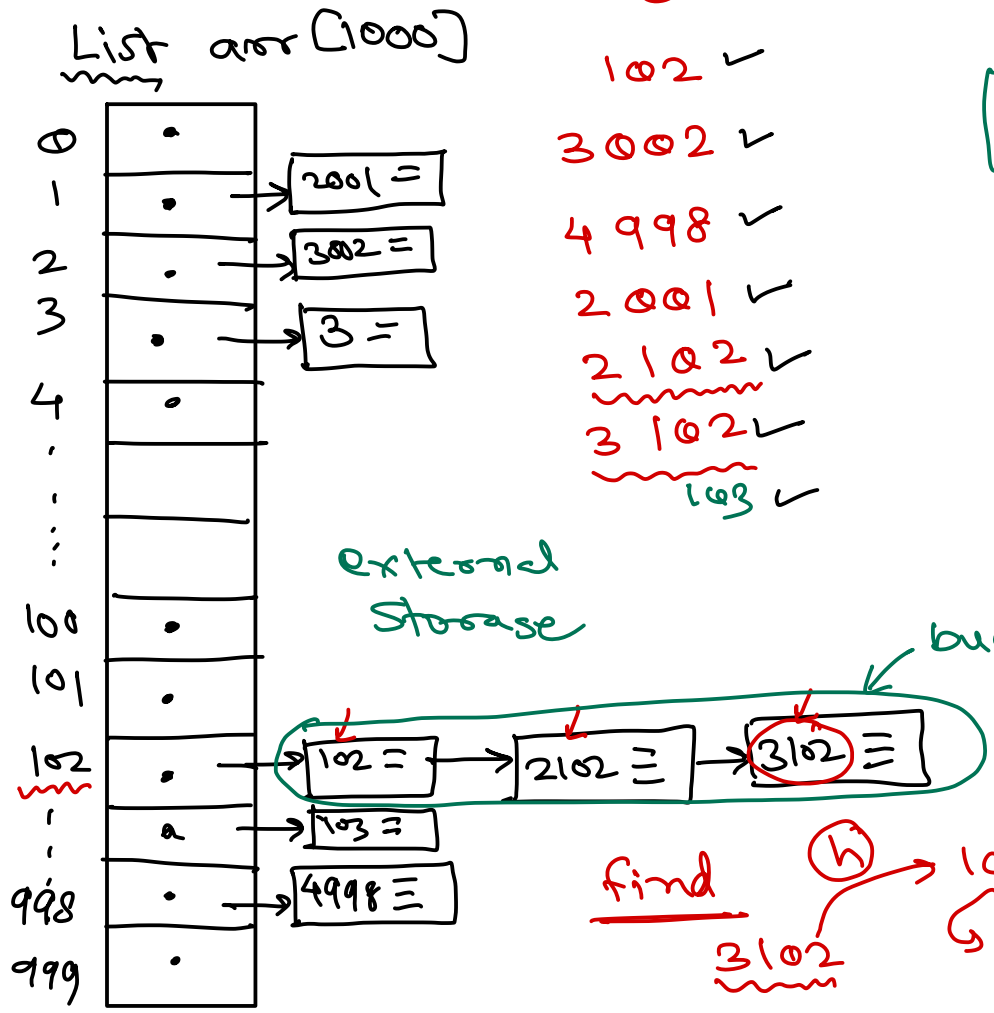
Load factor

$$= \frac{\text{num of kv pairs}}{\text{num of slots}}$$

* num of kv pairs < num of slots
  * load factor < 1
* num of kv pairs = num of slots
  * load factor = 1
* num of kv pairs > num of slots
  * load factor > 1

→ open addressing cannot be used.

chaining



Java collections : HashMap, ...
- chaining method

$$h(k) = 100$$

1. more the num of collisions, slower is the performance for add & search.
2. to reduce collisions better (uniform) hash fn should be implemented.
3. it is observed that multiply by prime num makes more uniform hash fn.

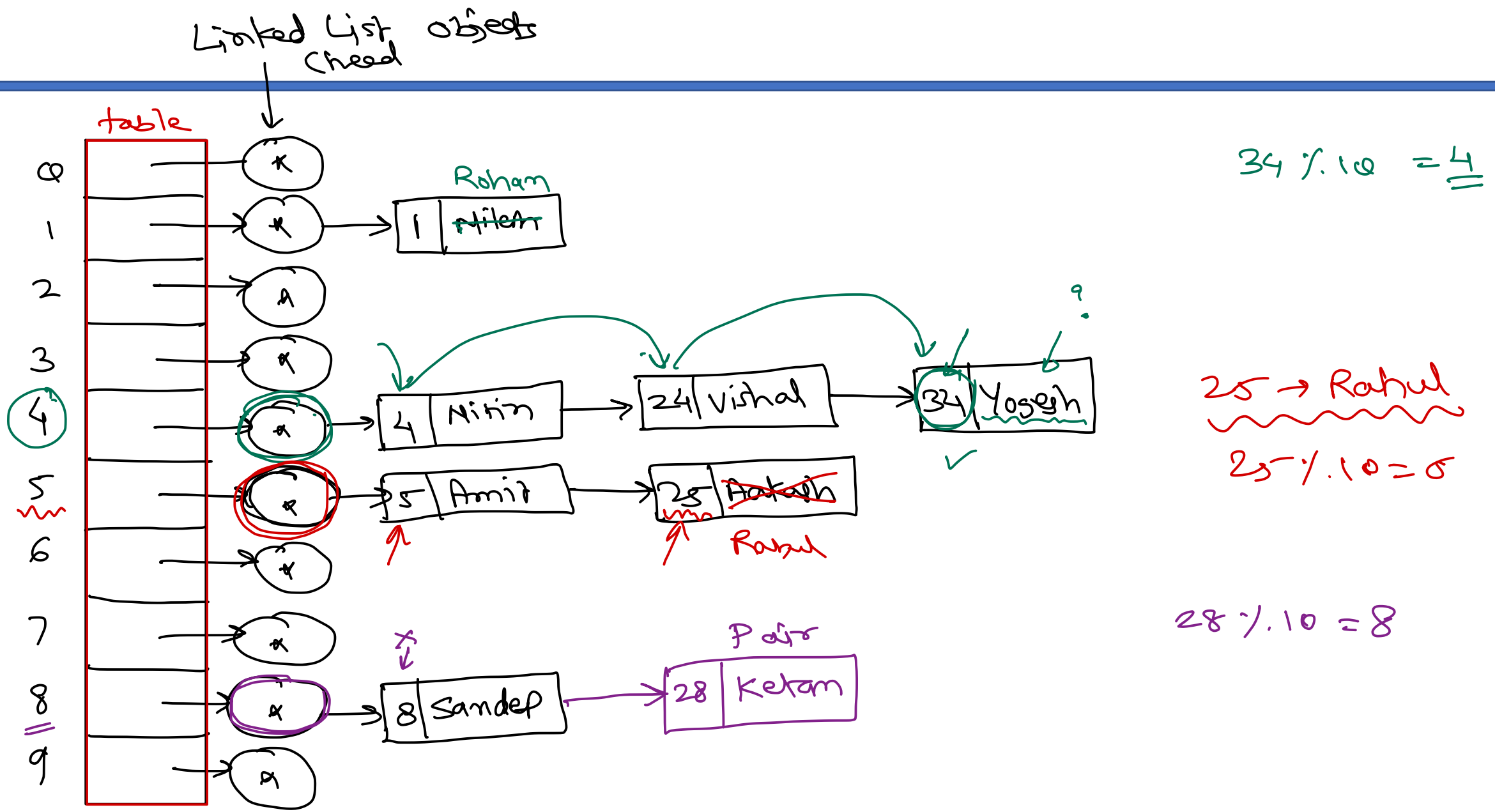
e.g. h(key) = (key * 31) % slots.

e.g. store students as per their height

key = height, value = student

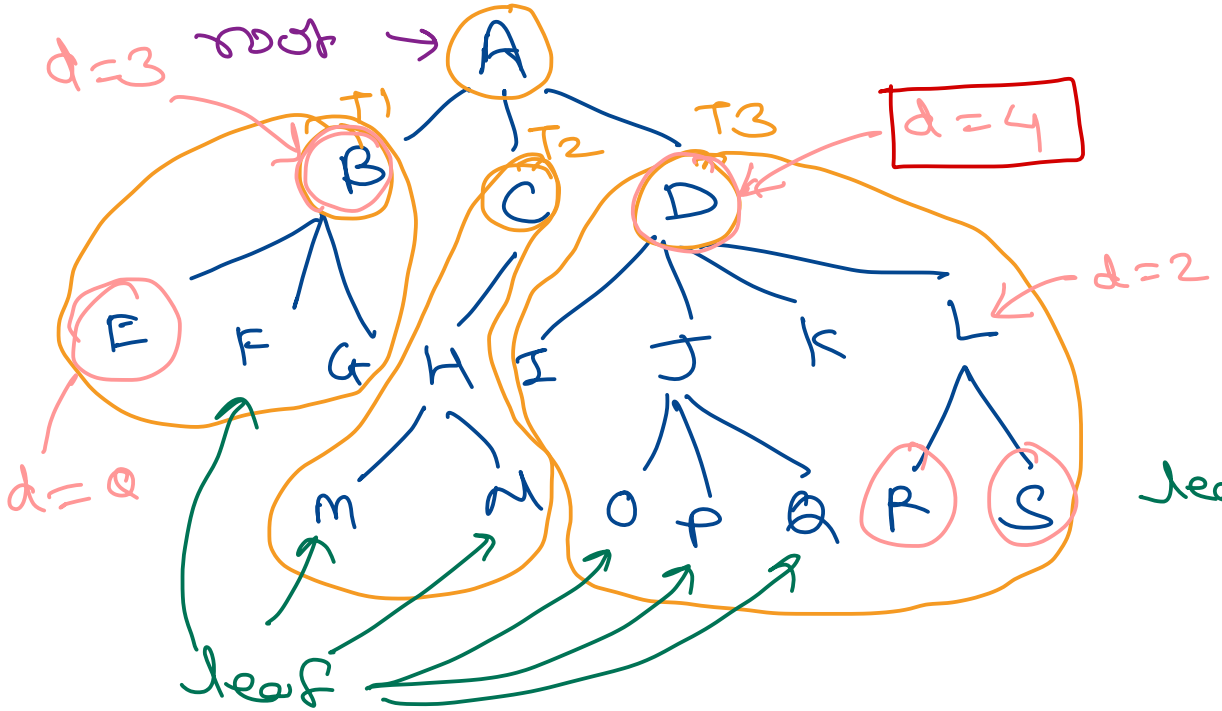
feet/inches.

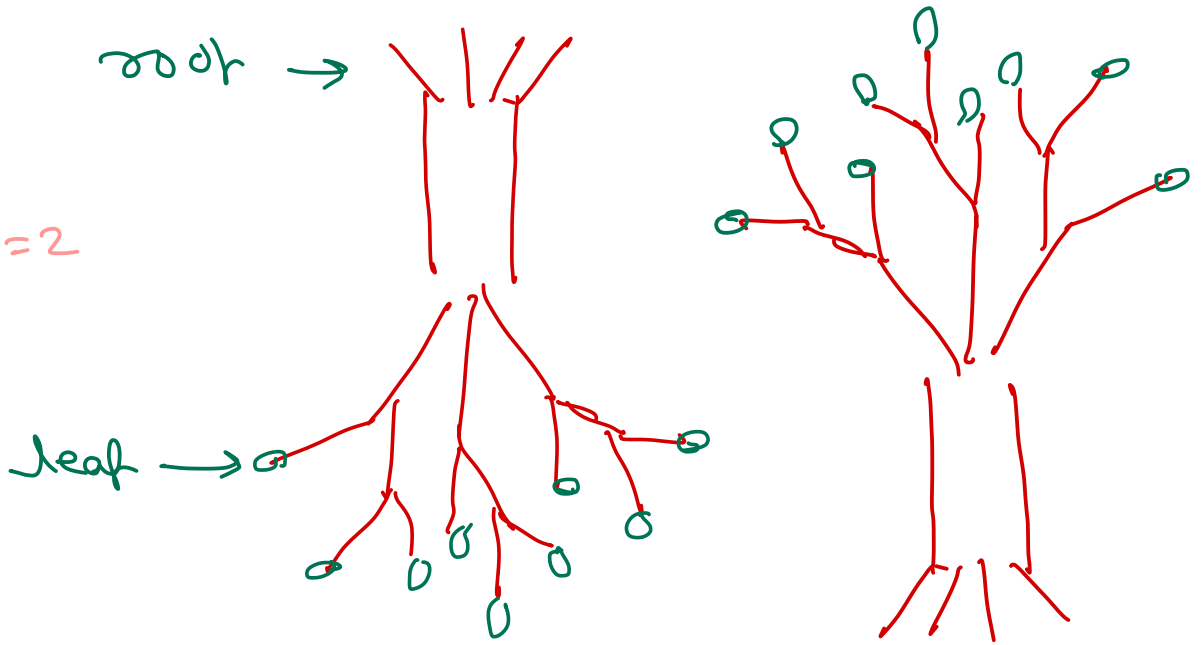
h(key) = (feet * 31 + inches * 31) % slots;


Linked List objects
Chead
table
0
1
2
3
4
5
6
7
8
9
Roham
1 Nilam
4 Nitin
24 Vishal
34 Yogesh
5 Amit
25 Rahul
Rahul
8 Sandep
28 Ketan
Pair
34 % 10 = 4
25 → Rahul
25 % 10 = 5
28 % 10 = 8

Tree Definition

- **Tree** is a finite set of nodes with one specially designated node called the "**root**" and the remaining node are partitioned into disjoints sets T1 to Tn, where each of those sets is a TREE.
- T1 to Tn are called **sub-trees** of the root



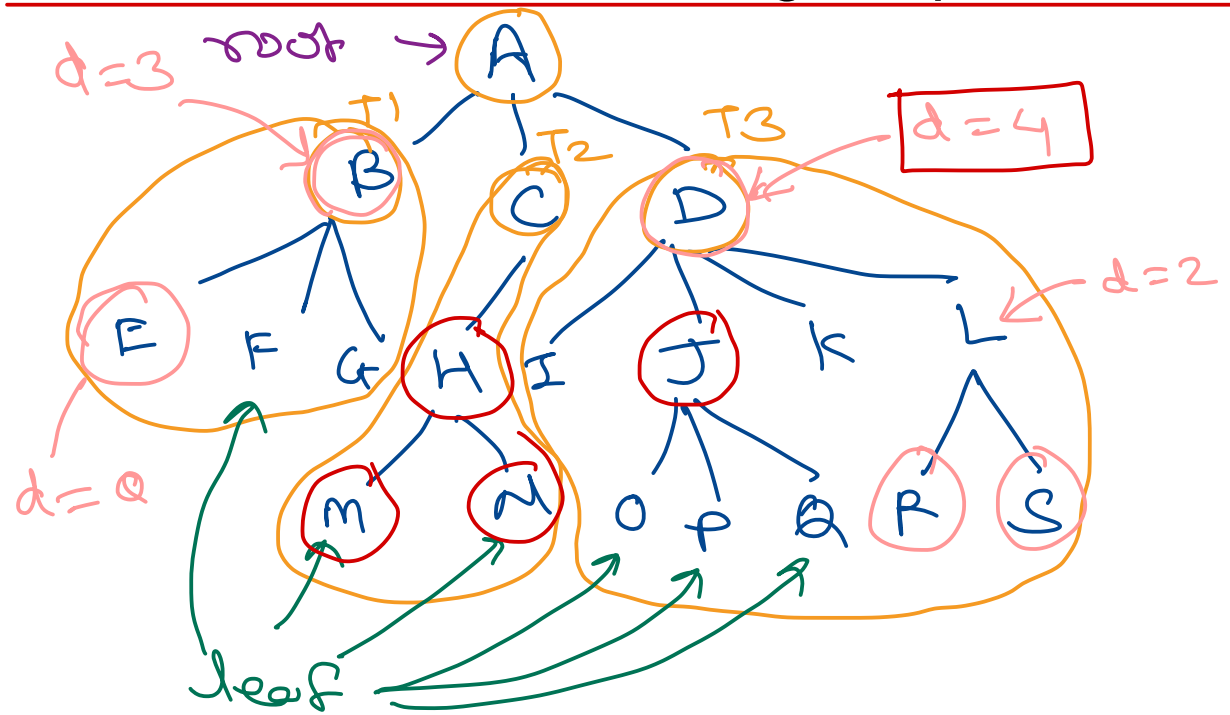


Tree terminologies

- Node: A item storing information and branches to other nodes
- Null Tree: Tree with no node
- Leaf Node: Terminal node of a tree & does not have any node connected to it
- Degree of a Node: No of sub trees of a node
- Degree of a tree: Degree of a tree is maximum degree of a node in the tree

Tree terminologies

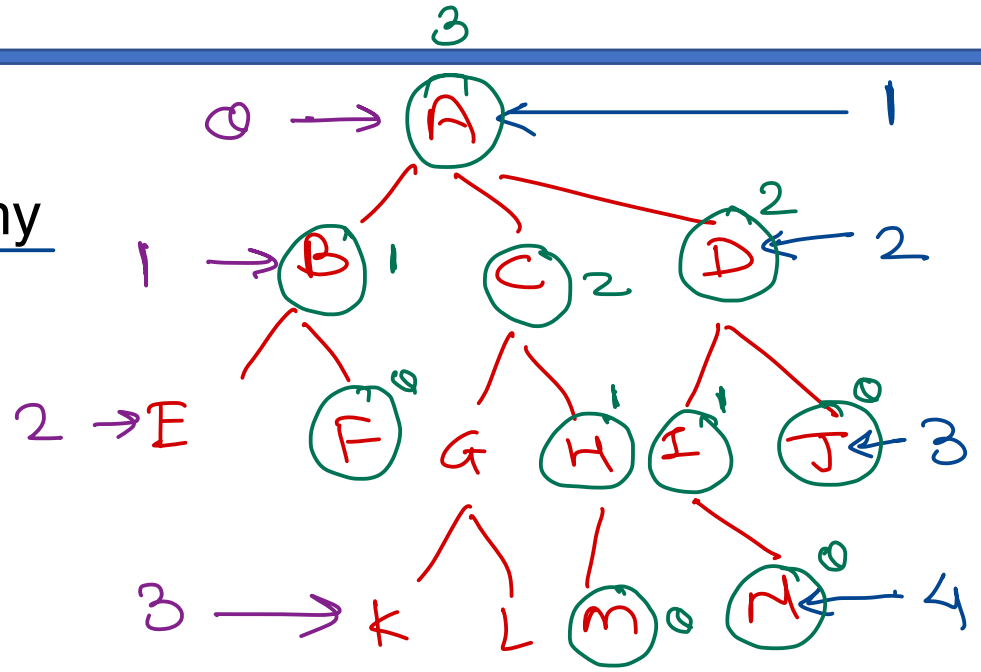
- Parent Node: node having other nodes connected to it
- Siblings: Children of the same parents → OPQ , BCD , MN , ...
- Descendants: all those node which are reachable from that node J → O P Q
- Ancestor: all the node along the path from the root to that node C → H M N



M → H , C , A
Q → J , D , A

Tree terminologies → can traversal

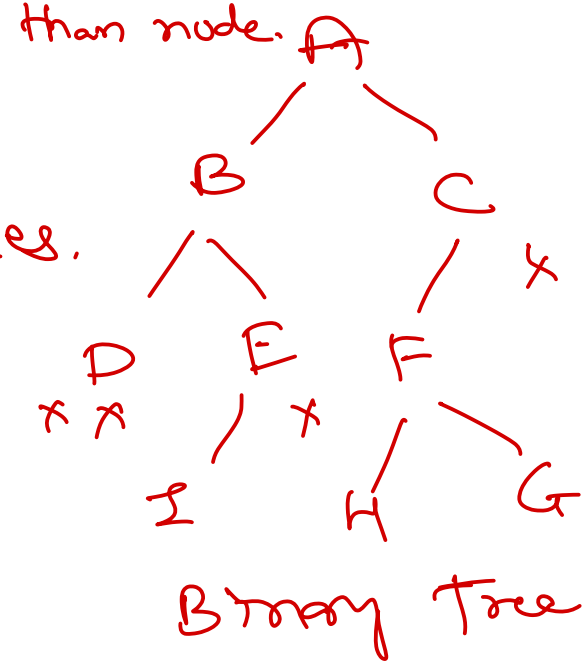
- Level of a Node:
  - Indicates the position of the node in the hierarchy
  - Level of any node is level of its parent +1
  - Level of root is 1
- Depth of a node:
  - Number of nodes from the root to the node.
  - Depth of root is 0
  - Level = Depth + 1
- Height of a node:
  - Number of nodes from the node to its deepest leaf.
  - Height of node = height of its child + 1
  - Height of empty/null tree is -1
- Height of a tree: Height of root of the tree.
- Traversal: Visiting each node of tree exactly once



Types of trees

- Binary Trees → max 2 child nodes for each node.
  - It is a finite set of nodes partitioned into three sub sets:- Root, Left sub tree, Right sub tree
- Binary Search tree
  - A binary search tree is a binary tree in which the nodes are arranged according to their values.

left child smaller than node
right child greater than node or equal



Multiway tree
or n-way tree.

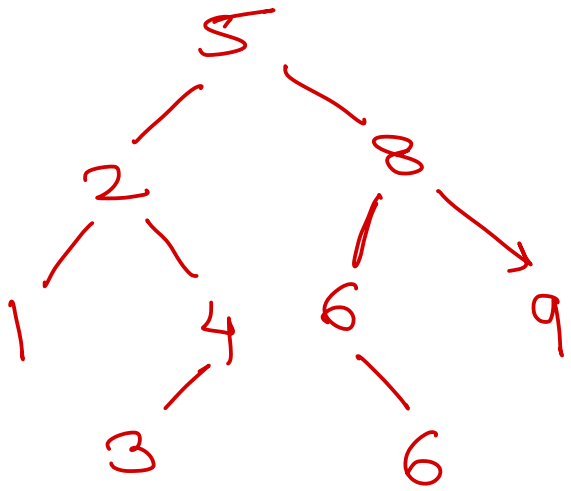
↓
each node have multiple child nodes.

```
class Node {
    int data;
    List<Node> children;
    =
    =
}
```

```
class Node {
    int data;
    Node left;
    Node right;
    =
}
```

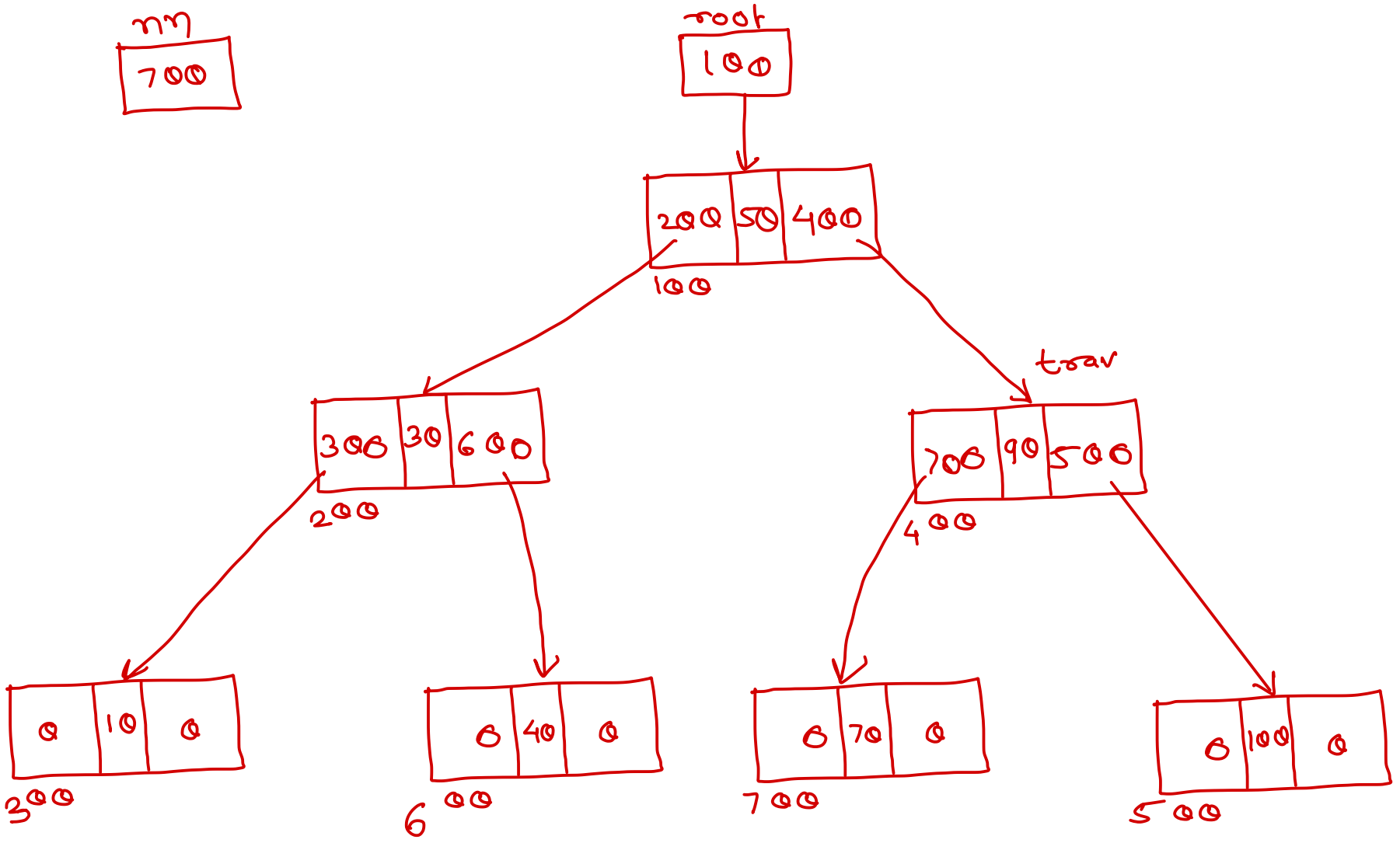
BST

5, 8, 2, 4, 6, 9, 1, 3, 6



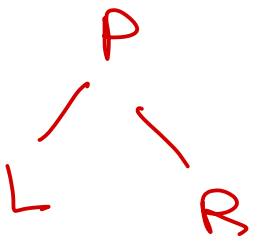
BST

50, 30, 10, 90, 100, 40, 70, 80, 60, 20



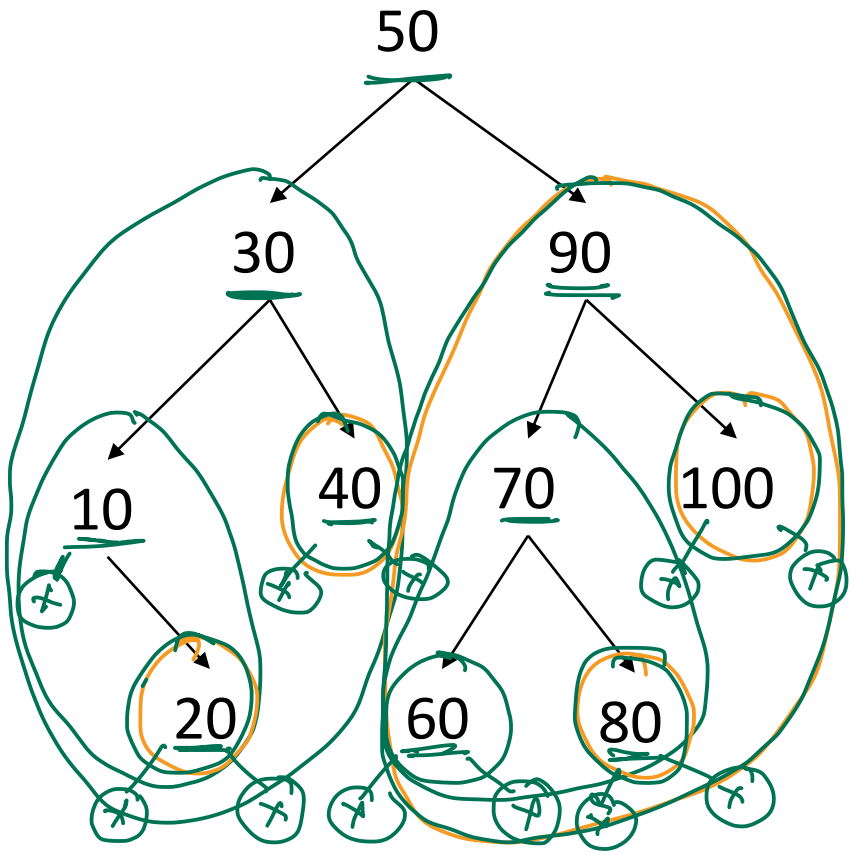
Binary Tree Traversal

- In-order: L P R
- Pre-Order: P L R
- Post-Order: L R P
- The traversal algorithms can be implemented easily using recursion.
- Non-recursive algorithms for implementing traversal needs stack to store node pointers.




```
preorder (trav).
    if(trav==null)
        return;
    print (trav.data);
    preorder(trav.left);
    preorder(trav.right);
```

BST → preorder – recursive.

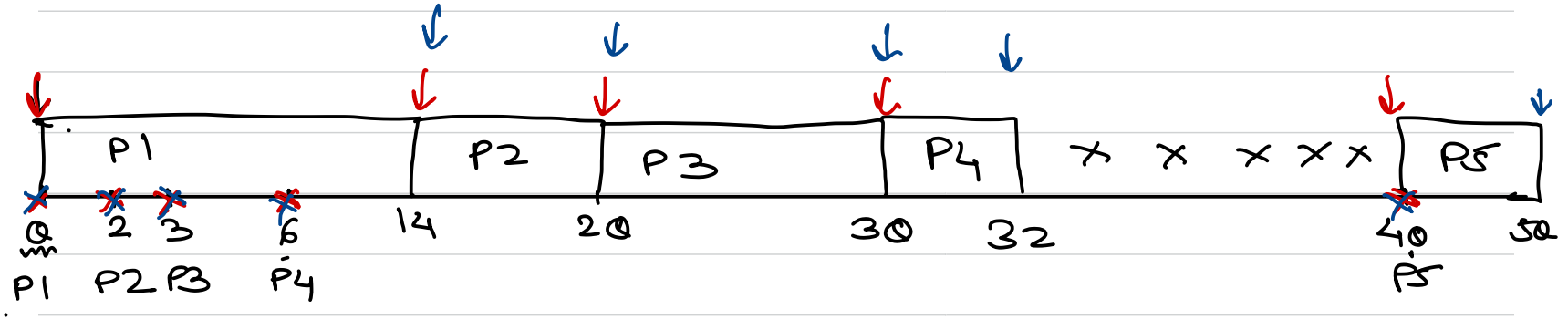



```
preorder (trav).
    if(trav == null)
        return;
    print (trav.data);
    preorder(trav.left);
    preorder(trav.right);
```

50 30 10 20 40

90 70 60 80 100

Assign 9



	waiting	turn-around
P1	0 − 0 = 0	14 − 0 = 14
P2	14 − 2 = 12	20 − 2 = 18
P3	20 − 3 = 17	30 − 3 = 27
P4	30 − 6 = 24	32 − 6 = 26
P5	40 − 40 = 0	50 − 40 = 10

class Job {
- ✓ id;
- ✓ arrival;
- ✓ burst;
- ? waiting;
- ? turnaroun

}

Queue<Job> ...

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>