UNIT - I

HASHING

★ **Hashing** -

→ • Sequential searching requires O(n) & Binary search require O(log n)

- Time required for searching depends on number of elements

- Hashing provides better techniques for storing data in sorted way.

- It is a process of indexing and retrieving elements in data structure to provide faster way of finding element using hash key.

■ **Definitions** -

i) **Hash Table** -
It is a data structure, an array, that maps key (data) with the help of hash function

ii) **Hash Function** -
It is a function used to place data or retrieve data from hash table

Date
Page

- **Definitions -**

iii) Load Factor -

It is defined as (m/n) where, n is the total size of hash table and m is preferred number of entries which can be inserted before an increment in size of hash table.

iv) Collision -

Collision is a situation in which hash function returns the same address for more than one record.

v) Overflow -

When hash table becomes full and new records need to be inserted then it is called overflow

★ Characteristics of Good Hash Function

→ • Good Hash Function avoids collisions
- Good Hash Function tends to spread keys uniformly in array
- Good Hash Function is easy to compute
- Good Hash Function depends on every bit of the key

★ Collision Resolution Strategies -

→ • If collision occurs, then it should be handled by applying some techniques, known as collision handling techniques.

- There are several strategies for collision resolution :

1) Seperate Chaining :

- In this technique, a seperate list of all elements mapped to the same values is maintained
- Avoid seperate chaining when memory space is limited

2) Open Addressing Hashing :

- It is an alternate method of handling collision and consists of 3 sub-types :

ⓐ Linear Probing ⓑ Quadratic Probing ⓒ Double Hashing

- In Linear probing whenever there is a collision, cells are searched sequentially for an empty cell

A) Linear probing with chaining (without replacement) - All records mapped to same location are stored in chain

B) Linear probing with chaining (with replacement) Problem of misplaced starting location of chain can be handed through chaining with replacement.

★ Modulo Division & Folding Methods of Hash function -

→ ▲ Modulo Division -

- In this method, we use modular arithmetic system & divide the key value by some integer divisor m
- It gives us location value, where the element can be placed, we can write

$L = (K \bmod m) + 1$, where

L is location in table

K = key value

m = table size

▲ Folding method -

- There are 2 folding techniques

1) Fold Shift -

In this, the key is divided into seperate parts where size matches with size of required address. Then left & right parts are shifted & added with middle part

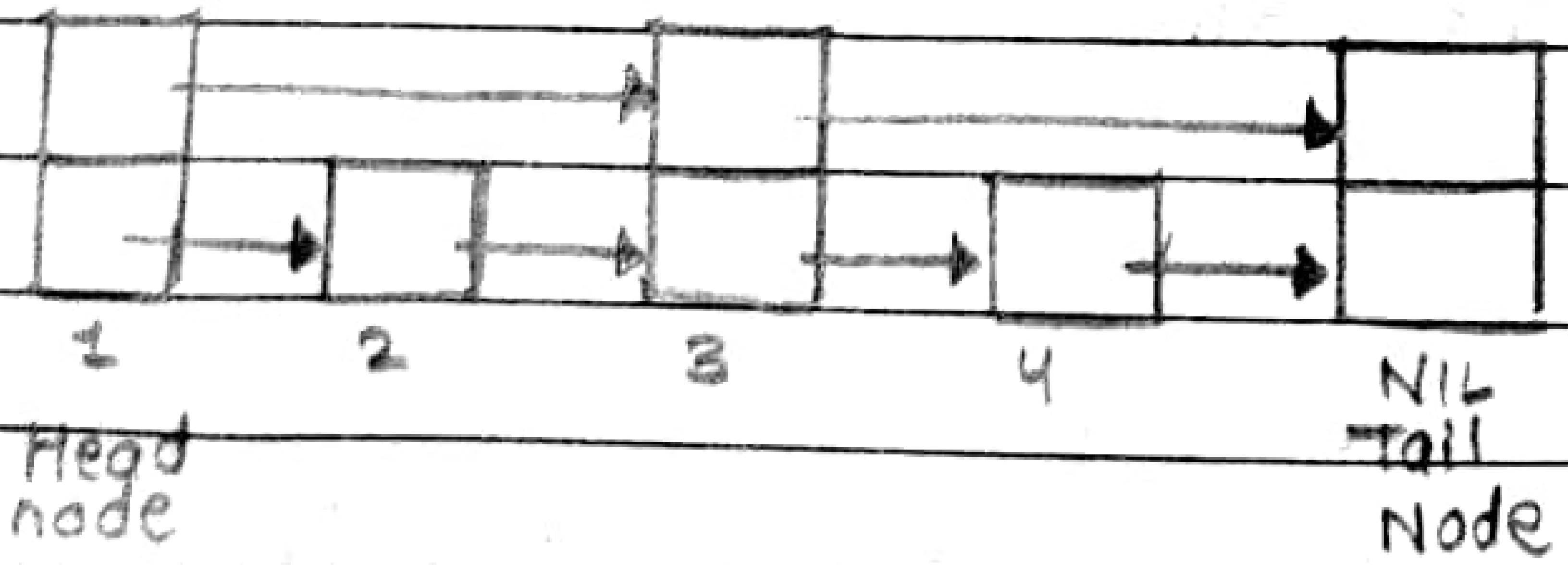
2) Fold Boundary -

In this, key is divided into seperate parts. Leftmost & rightmost parts are folded on fixed boundary & added with middle part.

★ Skip List -

→ • A list can be represented using linked list. Time required to search an element is proportional to no. of nodes that have to be examined

• Skip lists are made up of series of nodes one after other. Each node contains a key-value pair as well as one or more reference pointers.



- No. of references each node contain is determined randomly. & the no. of references is called its node level

- Skip list contains one head node (starting node) and one tail node (last node)

- Skip list is an efficient implementation of dictionary using sorted chain

Q.] Assume size of hash table as 8. The hash function to be used to calculate hash value of data X is X%8. Insert following values in hash table : 10, 12, 20, 18, 15. Use linear probing without replacement for handling collision

Same Numerical Nai Yenar

→ Table Size = 8
Hash function is X % 8

∴ 10 % 8 = 2
12 % 8 = 4
20 % 8 = 4 – Collision occurs place at $12c^5$
18 % 8 = 2 – Collision occurs place at $10c^3$
15 % 8 = 7

	Data	Chain
0	-1	-1
1	-1	-1
2	10	3
3	18	-1
4	12	5
5	20	-1
6	-1	-1
7	15	-1

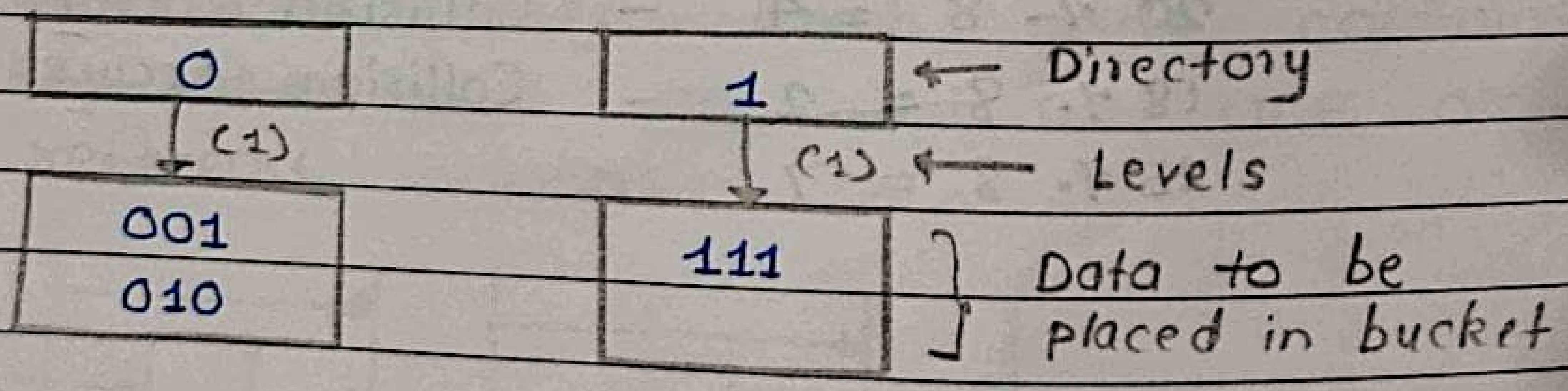
★ Extendible Hashing -

→ • It is technique which handles a large amount of data

• The data placed in hash table is by extraction of certain no. of bits

• Extensible hashing grow and shrink similar to B-trees

• Example -



★ Quadratic Probing -

→ • One way for reducing " Primary clustering " is to use quadratic probing to resolve collision

• Suppose key is mapped to location 'j' & cell 'j' is already occupied

• In quadratic probing, location (j+1), (j+4), (j+9) are examined to find first empty cell

• This table reduces primary clustering but it does not ensure that all cell in table will be examined to find empty cell.

★ Rehashing -

→ • Rehashing tells us what to do when hash table gets full, instead of waiting for hash table to get completely full, it is more efficient to rehash when table is 70-80% full

• Rehashing is costly operation & it happens frequently when hash table is small & there are lot of insertions

• Time required for rehashing is O(n)

UNIT II

TREES

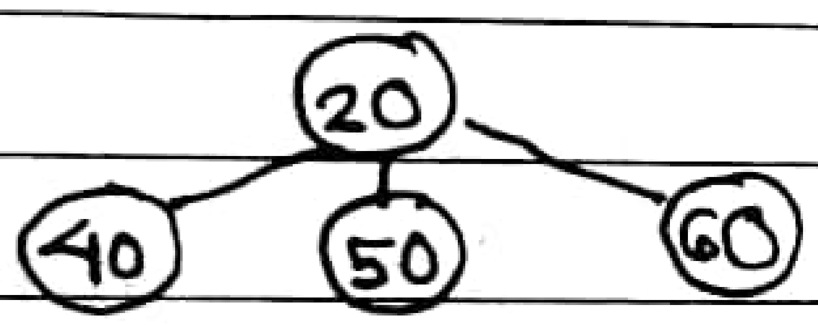
- **Basic Terminologies :**

i) Root - It is a unique node in tree to which further subtrees are attached.

ii) Leaves - These are terminal nodes of tree

iii) Degree of node -
The total number of sub-trees attached to that node is called degree of node

Eg,



← Degree is 3

iv) Degree of Tree -
Maximum degree in tree is the degree of tree

v) Level of Tree -
The Length of path from root to node plus 1

vi) Height of tree -
Maximum Level is height of the tree.

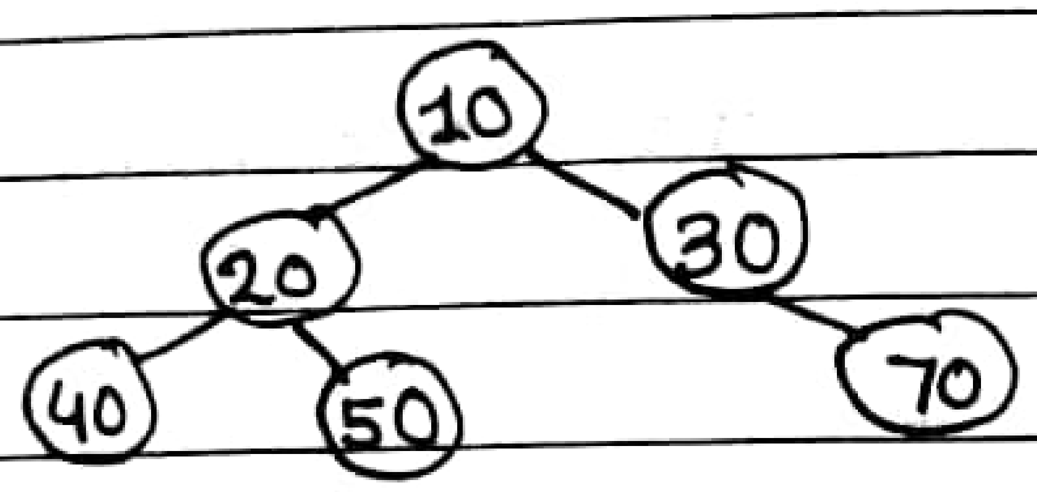
vii) Siblings -
Nodes with common parent nodes are called siblings / brothers

- **Definition -**

Binary Tree :

A binary tree is a finite set of nodes which is either empty or consists of a root and two nodes attached to it

Eg,



- **Definition**

Skewed Binary Tree -

The Tree in which each node is attached as left child of parent node or attached as right child of parent node is called Skewed Binary Tree

★ **Linked Representation of Tree -**

→ • In binary tree, each node will have left child, right child, and data field

Left Child	Data	Right Child

• Structure of node in binary tree -

```
typedef struct node
{
int data;
struct node *left;
```

```
    ... node *right;
  } bin;
```

* Binary Tree Traversals -

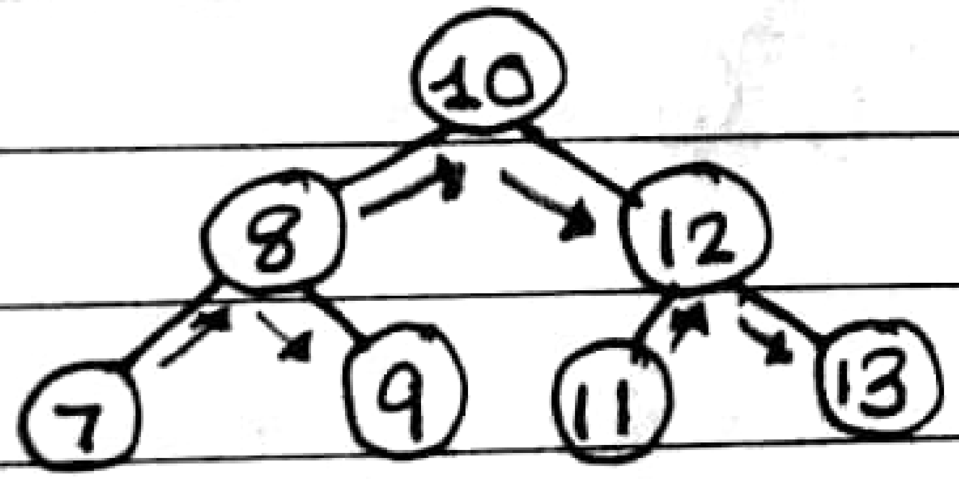
→ • Ordered rooted trees are often used to store information and we need procedures for visiting each vertex to access data

- Procedure for systematically visiting every vertex of ordered rooted tree are called Traversal Algorithms.

1) In-order Traversal :

- In this technique, leftnode, parent and then right node visit is followed

- Eg,

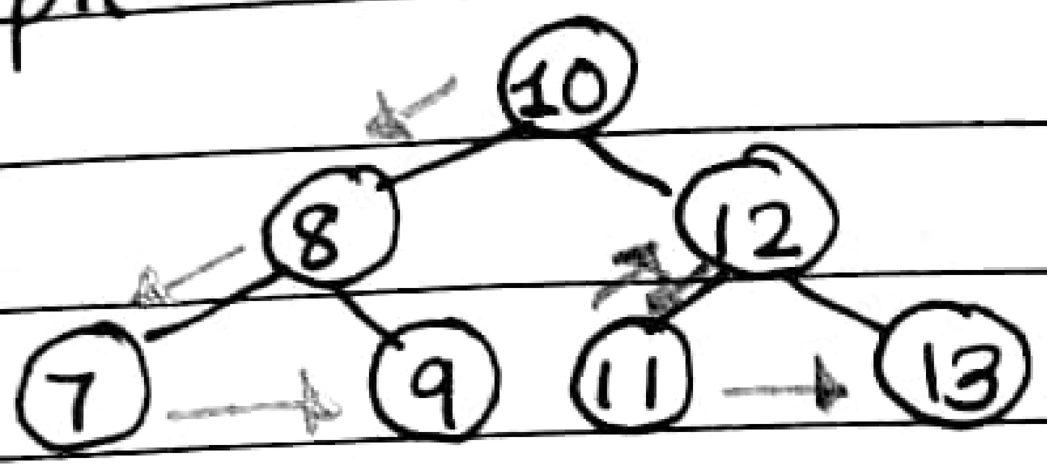


Sequence is : 7 - 8 - 9 - 10 - 11 - 12 - 13

2) Pre-order Traversal -

- In this,
  Parent - leftnode - Rightnode visit is followed

- Example -

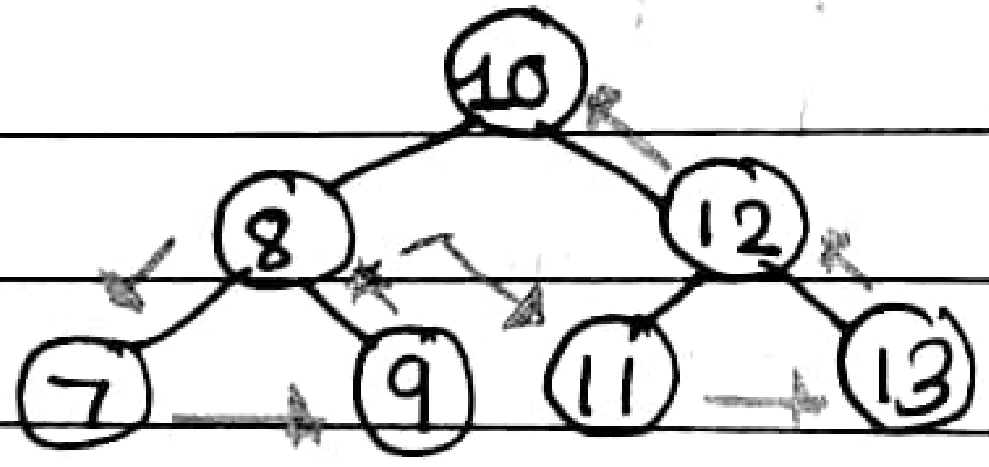


Sequence - 10, 8, 7, 9, 12, 11, 13

3) Post-order Traversal -

- In this,
  Left node - Right node - Parent Node
visit is followed

- Example -



Sequence :- 7 - 9 - 8 - 11 - 13 - 12 - 10

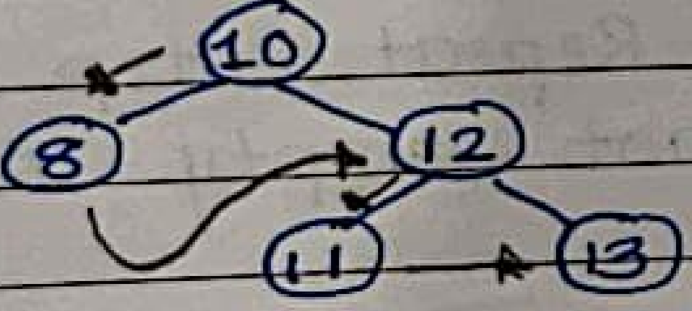
★ Depth - First Search -

→ • In this traversal technique, the tree is traversed according to its depth and visited vertex is printed.

• Algorithm -

```
Step 1 : Visit the root node. Push it onto stack
Step 2 : Pop the node & display it as output
Step 3 : If right child is not NULL, push
         it onto stack
Step 4 : If left child is not NULL, push it
         onto stack
Step 5 : Repeat step 2 to step 4 until
         stack is not empty
```

• Example,



DFS sequence is 10, 8, 12, 11, 13

★ Breadth First Search -

→ • It is a searching technique in which all nodes of same level are displayed.

• Algorithm -

Step 1 : Visit the root node. Insert it in Queue from rear end

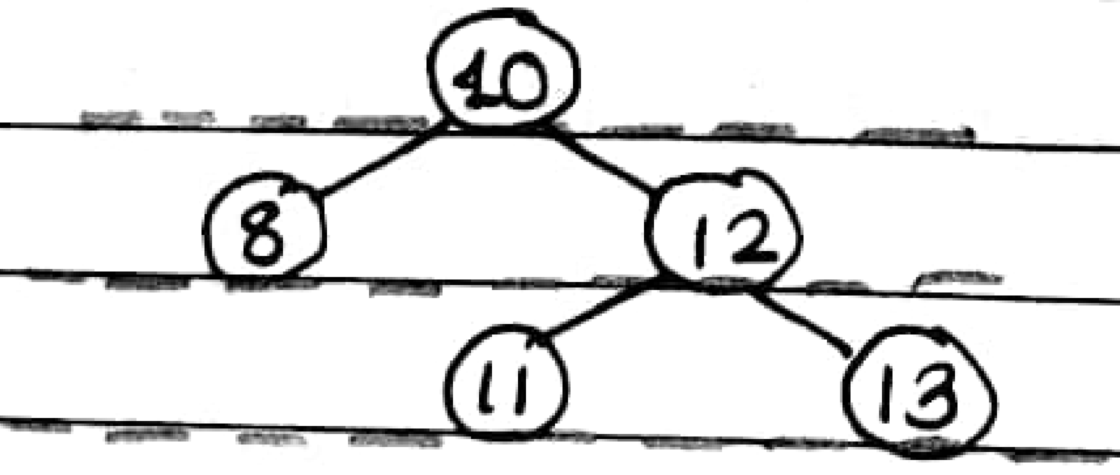
Step 2 : Delete the node from front end of Queue and display it

Step 3 : If left child is not NULL, insert left child in Queue

Step 4 : If right child is not NULL, Insert right child in Queue

Step 5 : Repeat step 2 to step 4 until queue is not empty & all nodes are visited

• Example -



BFS sequence is : 10, 8, 12, 11, 13

★ Binary Search Tree Operations -

→ • Operations performed on BST are -

1) Insertion of node :

▲ Algorithm :

1. Read value of node to be created and store it in New node
2. If (root != NULL), then root = New
3. Again read next value of node created in New
4. If (New-> value < root -> value) then attach New node as left child
   Otherwise attach New node as right child
5. Repeat step 3 & 4 for constructing tree

2) Deletion of element from binary tree -

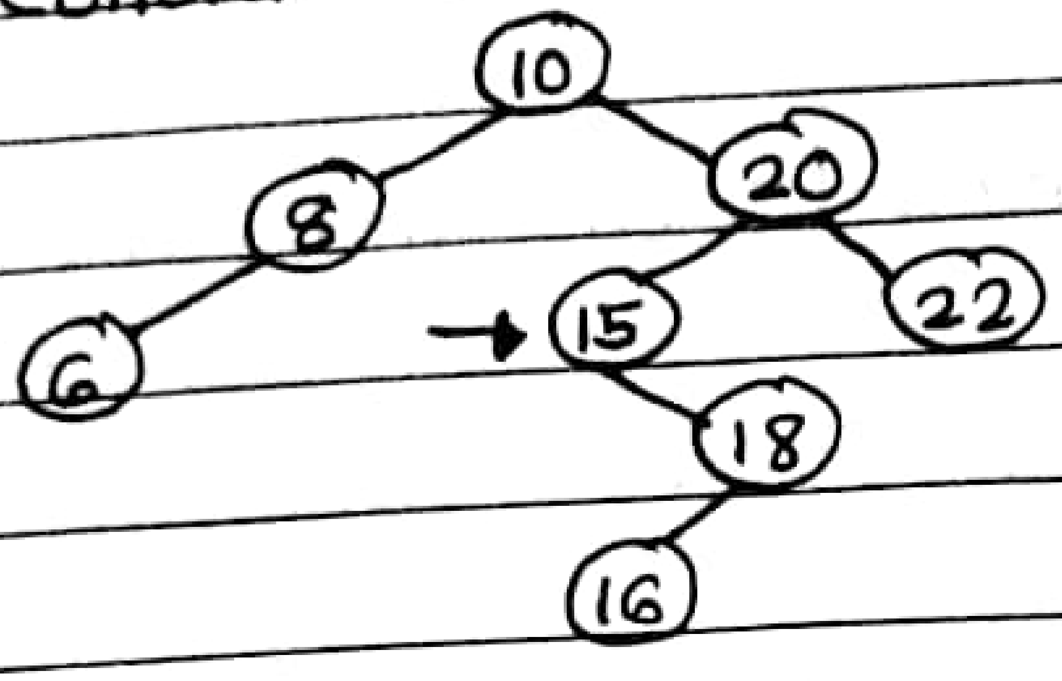
• For deletion of node, there are three cases :

i) Deletion of leaf node -

This is simplest deletion, in which we set the left or right pointer of parent node as NULL

ii) Deletion of node having one child -

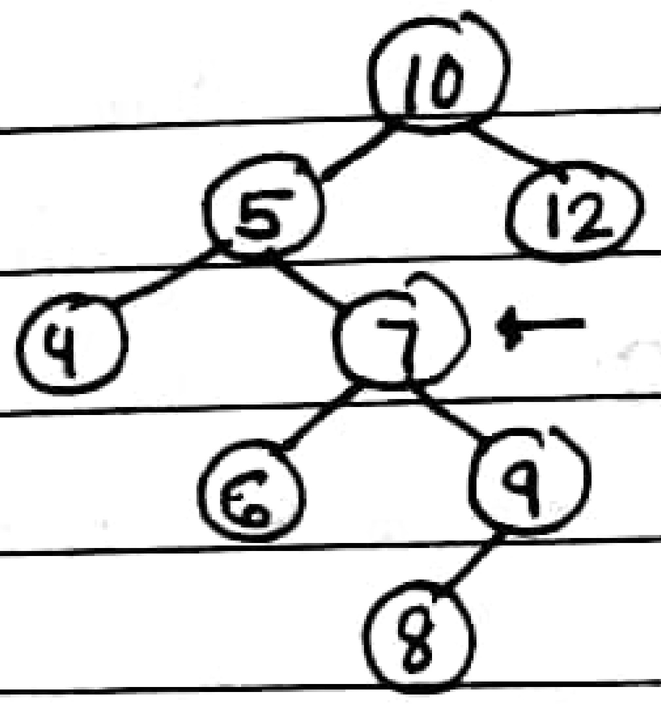
Consider a tree



If we want to delete node 15, then simply copy node 18 at place of 15 and then set node free

iii) Node having two children -

Consider a tree



If we want to delete node 7, we simply find out inorder successor of node 7. and copy it at location of node 7.

3) Searching a node :-

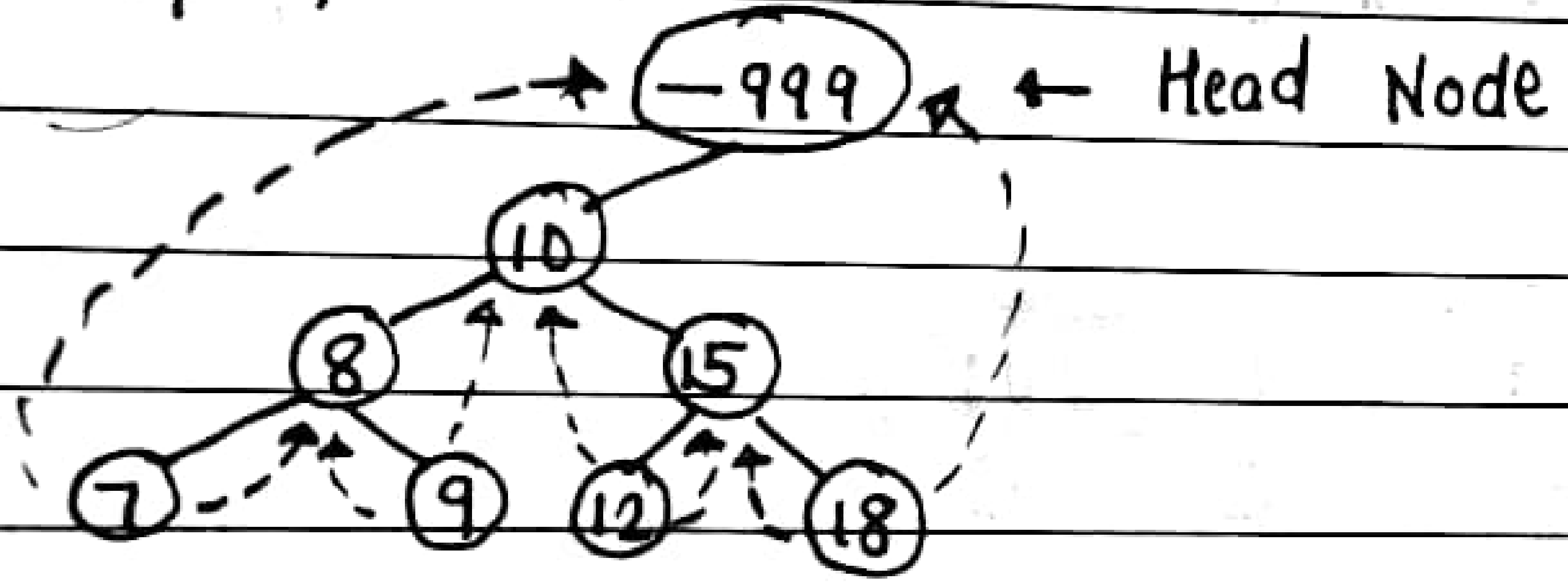
- In searching, node which we want to search is called key node.
- Key node will be compared with each node, if value is greater then search it on righ branch otherwise on left sub-branch.

★ Threaded Binary Tree -

→ - The leaf nodes of binary tree have no links or 0-links.
- These 0-links can be replaced by pointers called Threads

- The basic idea of inorder threading is that left thread should point to predecessor and right thread should point to successor.

- Example,



- If node A is pointed by left thread of node B, then node A becomes inorder predecessor of B
  e.g inorder predessor of 9 is 8

- If node C is pointed by right thread of node B then C becomes inorder successor of B
  e.g inorder successor of 9 is 10

★ Huffman Algorithm -

→ Step 1 : Arrange weights in increasing order

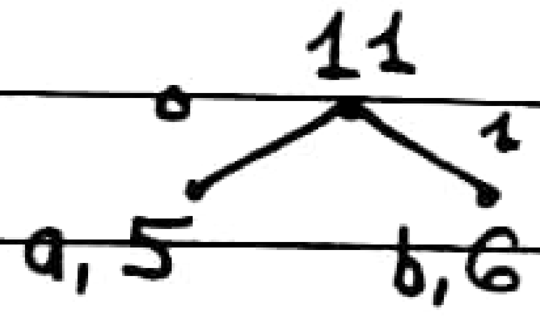
Step 2 : Consider two leaves with minimum weights. Join two leaves, to form a subtree, by adding weights of two leaves

Step 3 : Repeat above step, till no weights remains

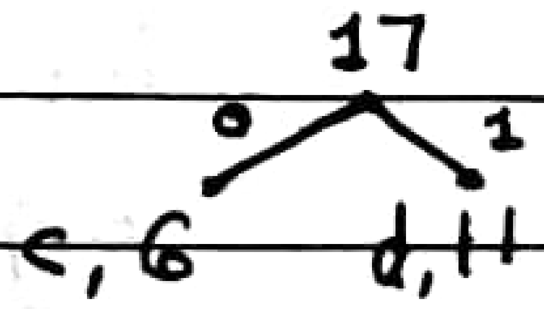
Step 4 : Tree obtained is optimal tree

- Example,

a	b	c	d	e
5	6	6	11	20



c	d	a+b	e
6	11	11	20



a+b	c+d	e
11	17	20