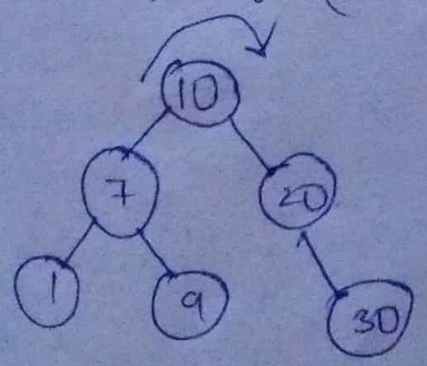
Splay Trees:-

Splay tree can be defined as the self adjusted tree in which any operation performed on the element would rearrange the tree so that the element on which operations has been performed becomes the root node of the tree.

* splay tree has one extra property that makes it unique in splaying
* All the ~~splay tree~~ operations in the splay tree are followed by splaying

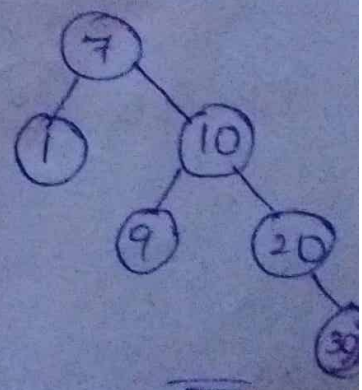
Rotations used for splaying:-

1) Zig (Right rotation)
2) Zag (Left rotation)
3) ZigZag (Right followed by left)
4) ZagZig (left followed by right)
5) Zig Zig (Two right rotations)
6) Zag Zag (Two left rotations)

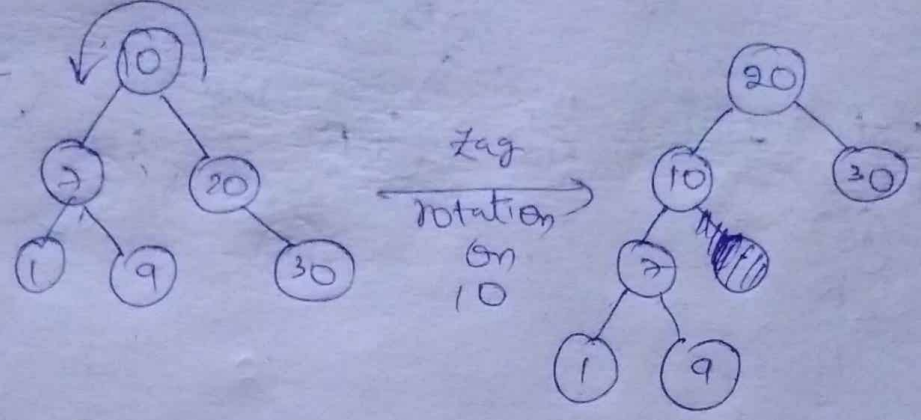


Search 7 in tree

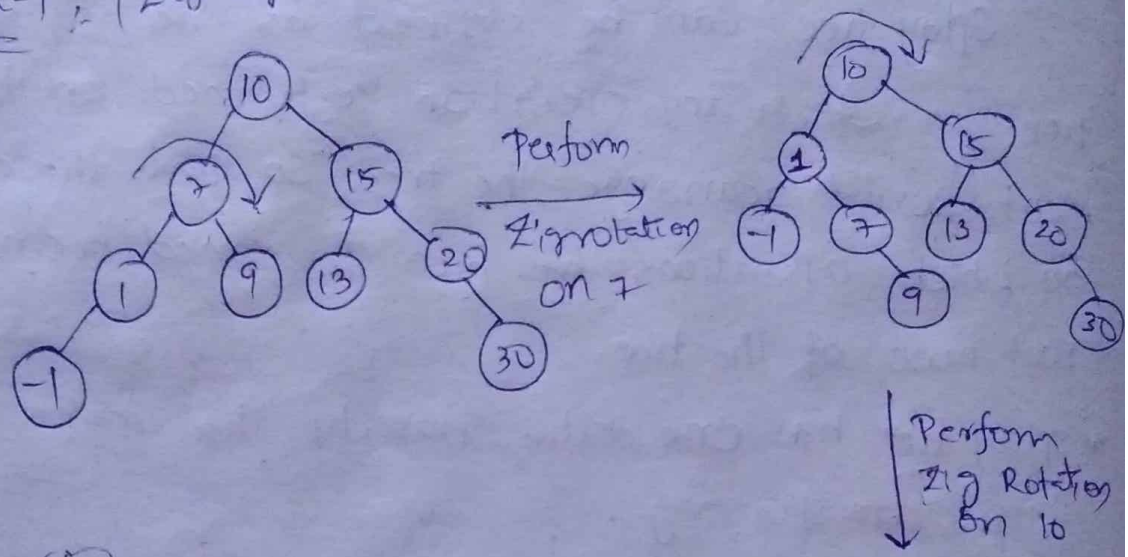
→ Perform right rotation on 10 (zig)

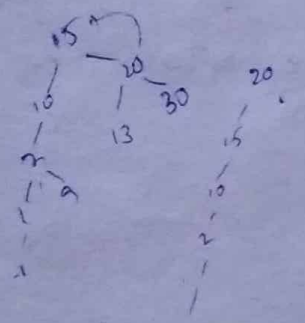


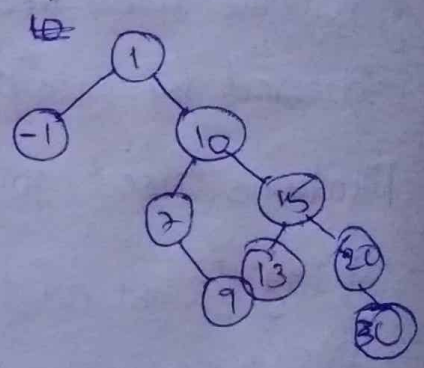
Q-2 :-



Search-1 :- (Zig-Zig)

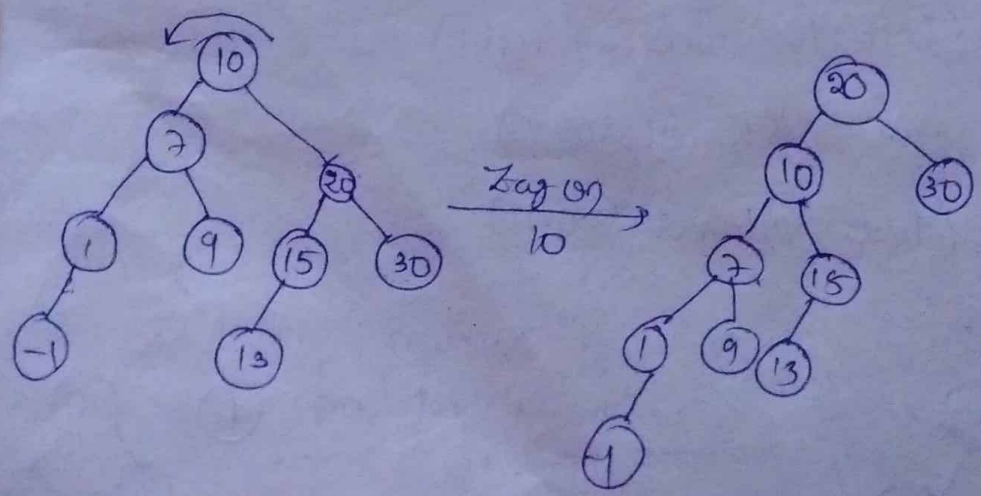




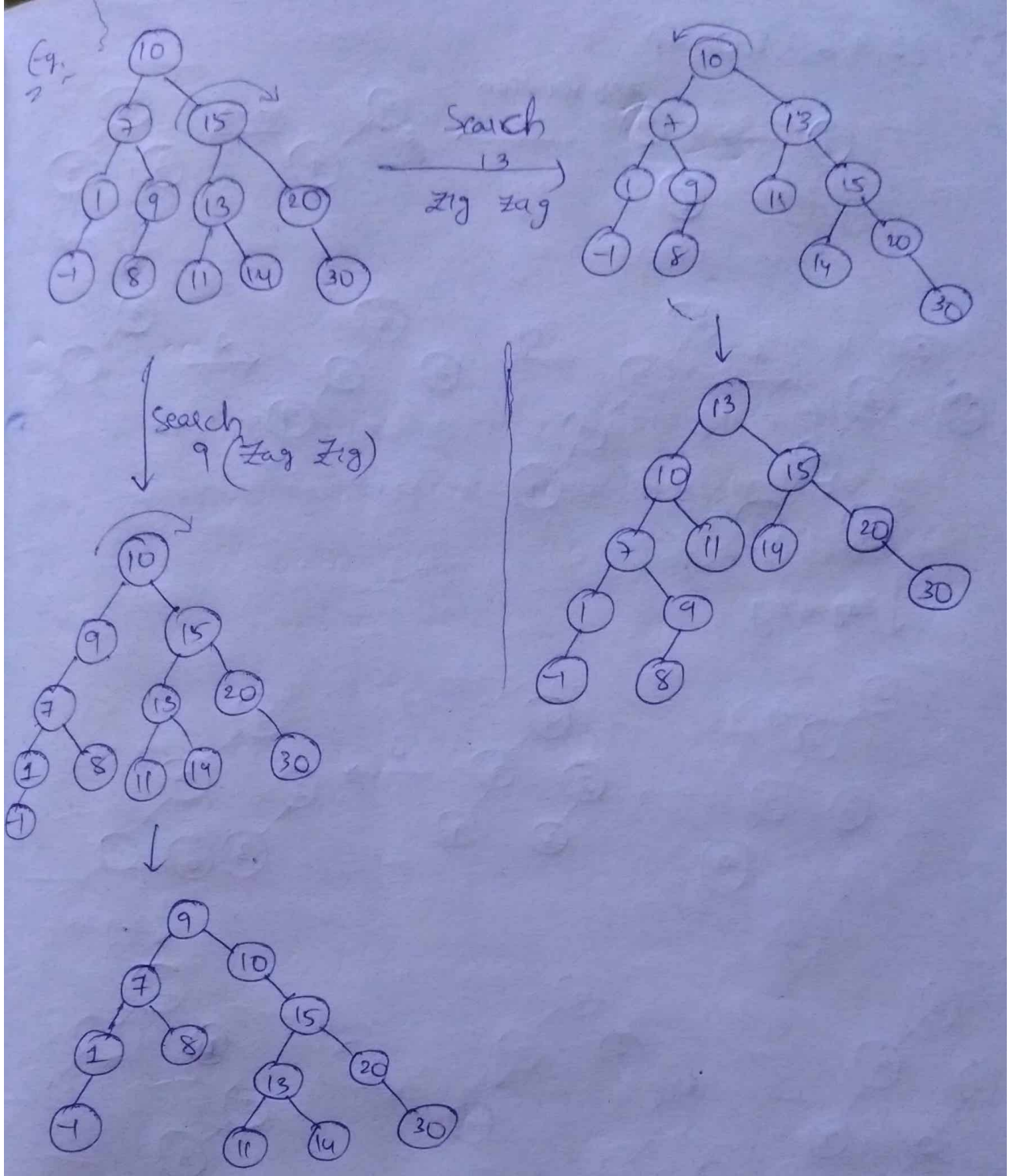


Search 20 :-

Zag rotation on ~~10~~ 15



Eg:-



Splay trees are two types :-

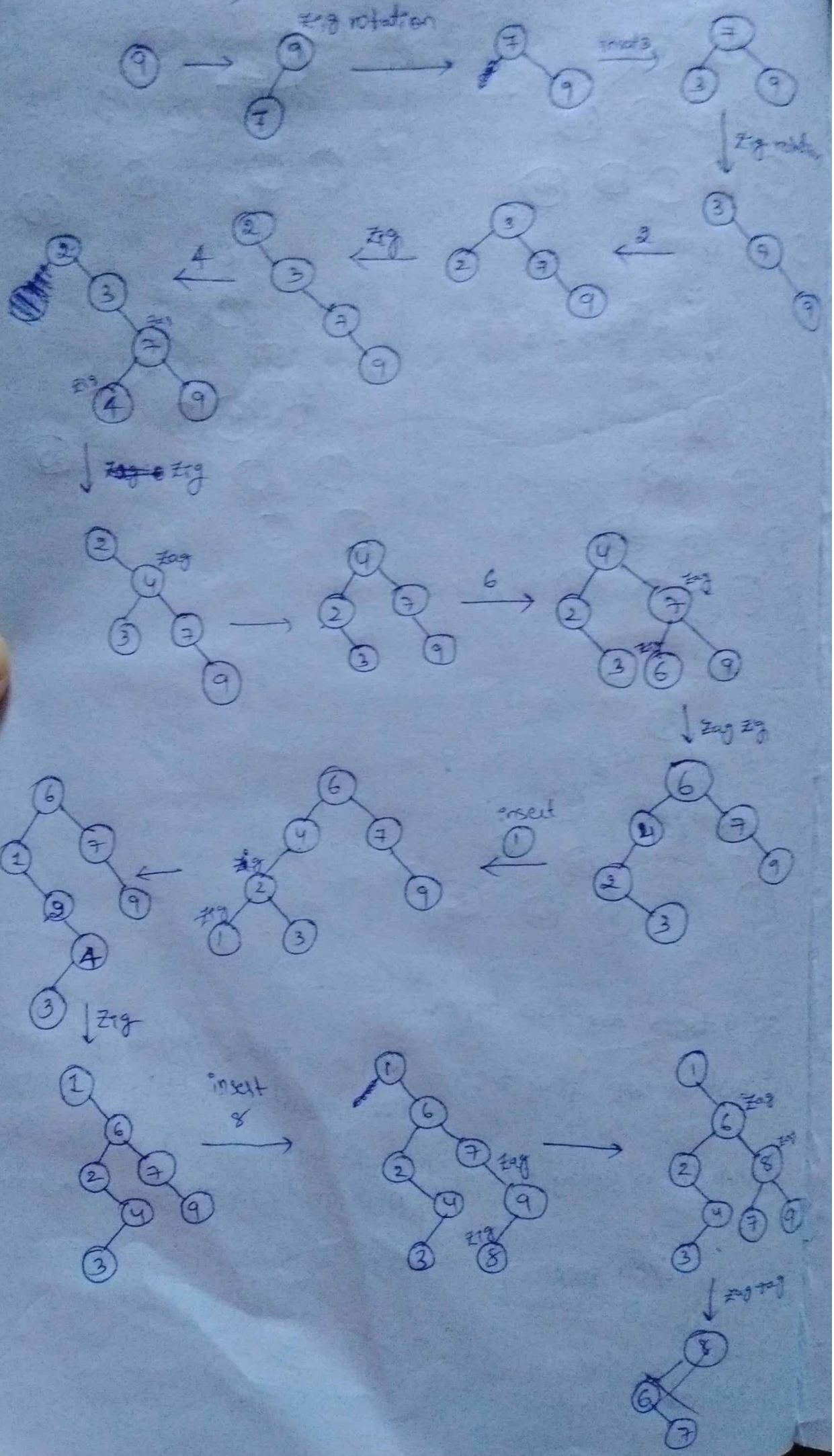
1) Bottom up splay tree - Perform splay operation from newly inserted node to root node

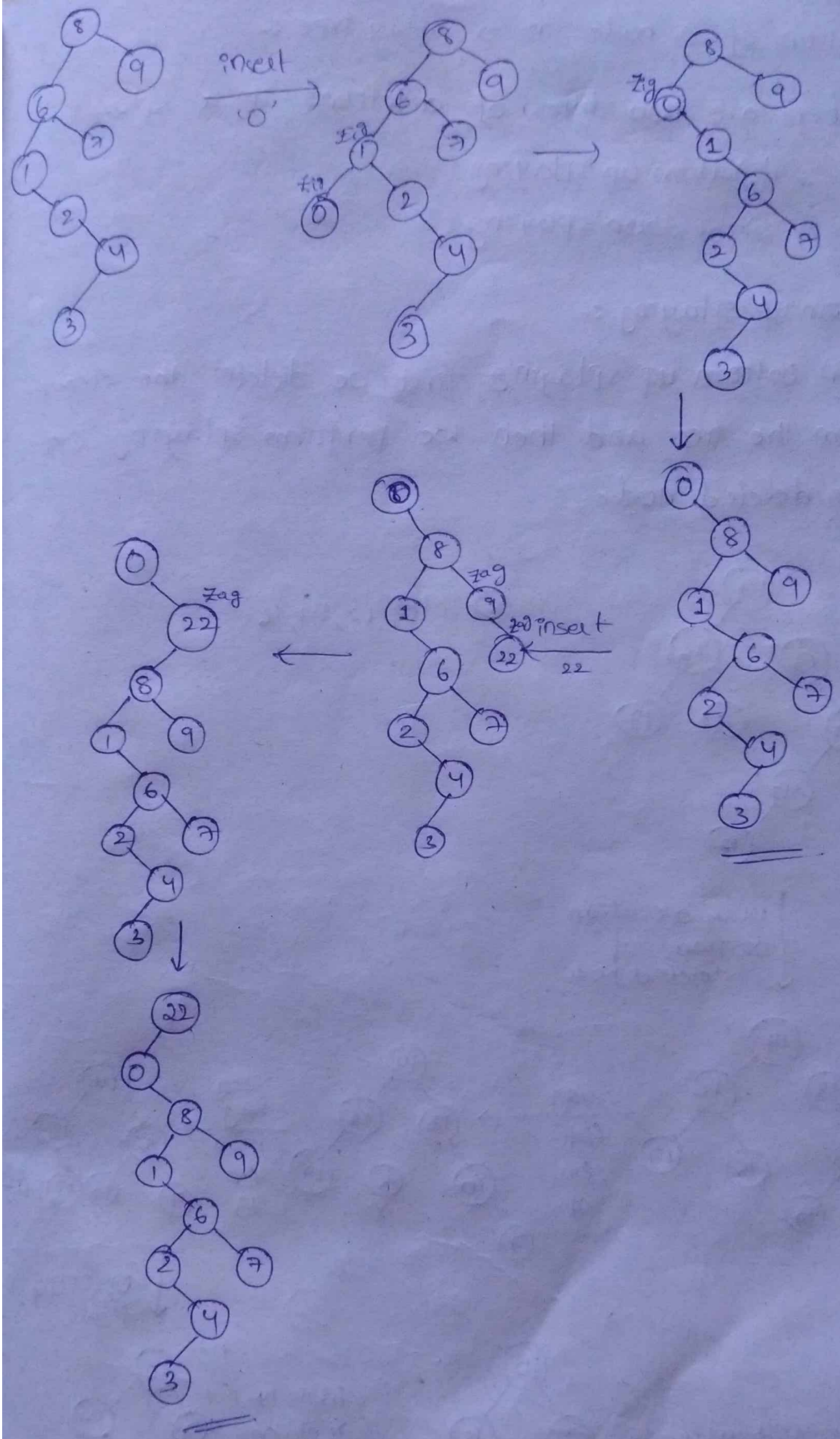
2) TOP down splay tree - Perform splay operation from root node to newly inserted node.

Bottomup splay tree operations :-

1) Insert
2) Delete
3) Search.

① 9, 7, 3, 2, 4, 6, 1, 8, 0, 2




insert
'0'
zig
zig
zig
zig
zag
zag
zag
insert
22

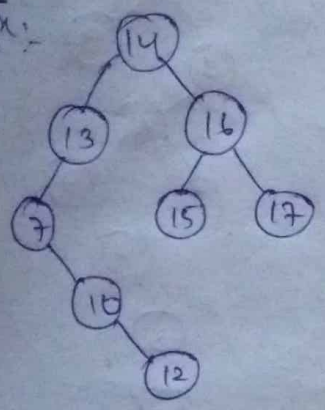
Deletion of a node in a splay tree :-

There are two types of deletions in a splay tree :- 1) Bottom up splaying
2) Top down splaying

Bottom up splaying :-

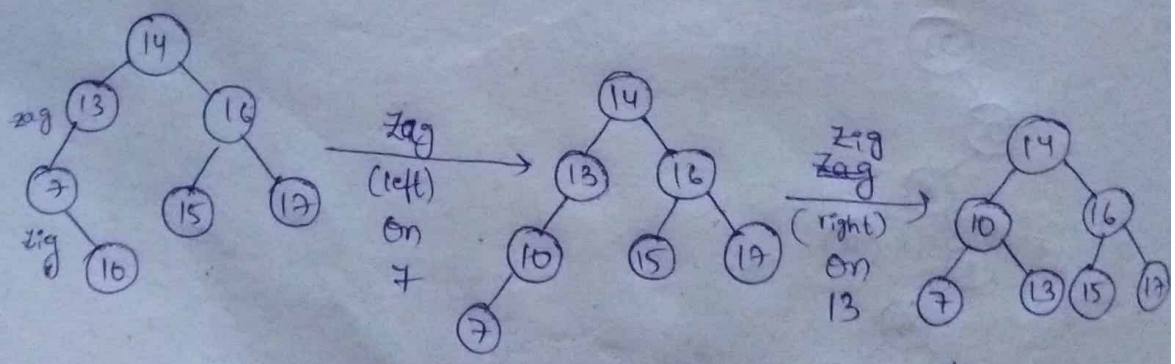
In bottom up splaying, first we delete the element from the tree and then we perform splaying on the deleted node.

Eg:-

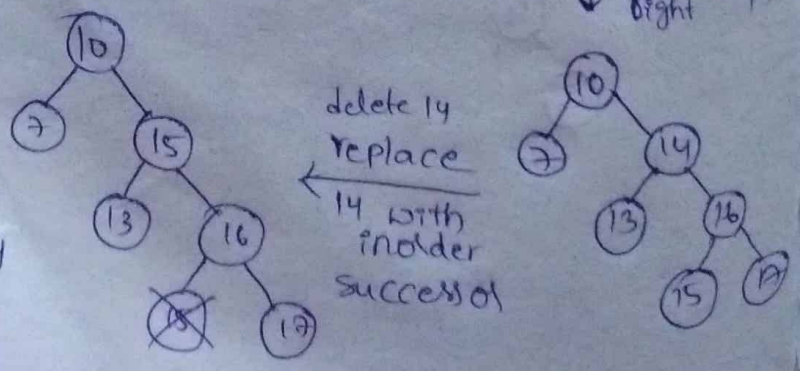


Delete 12, 14, 16

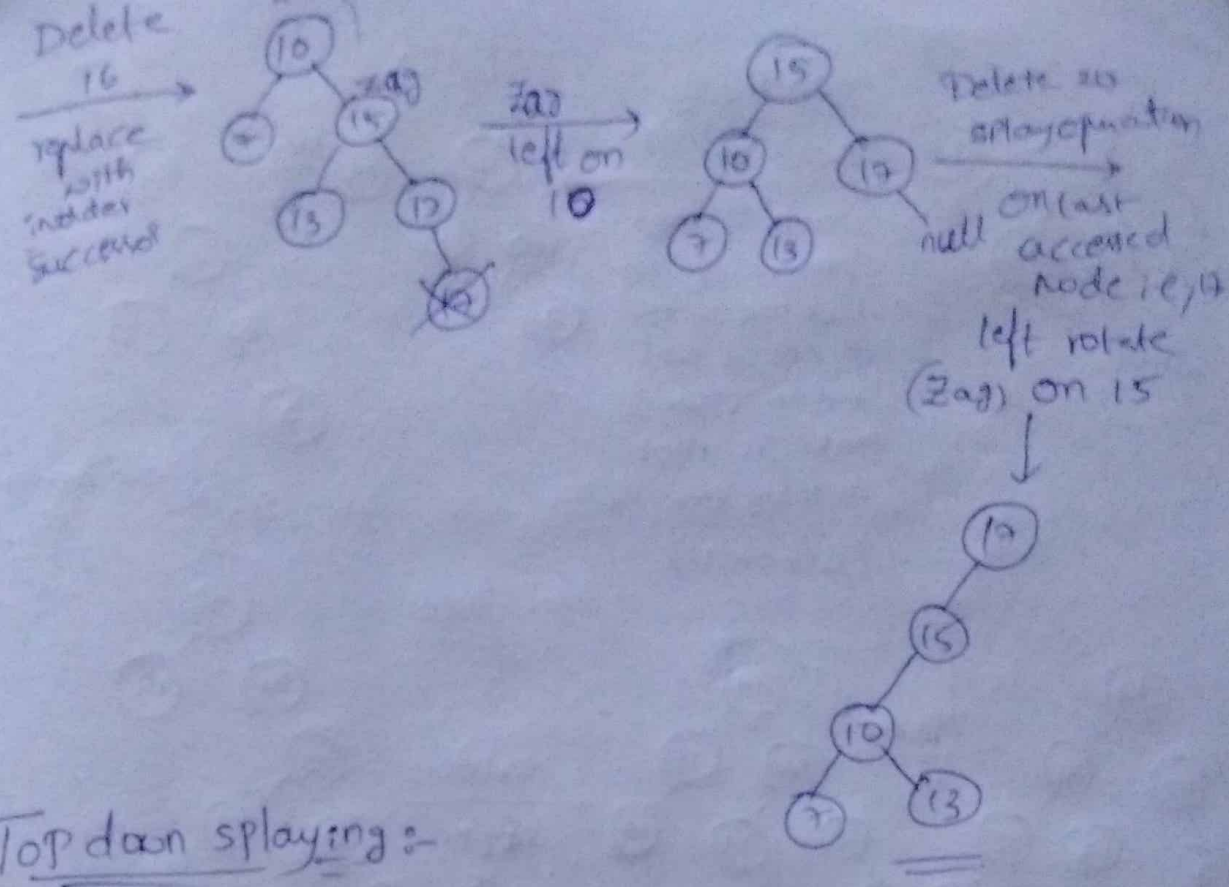
splay operation on parent of deleted node



Zig (on 14) right



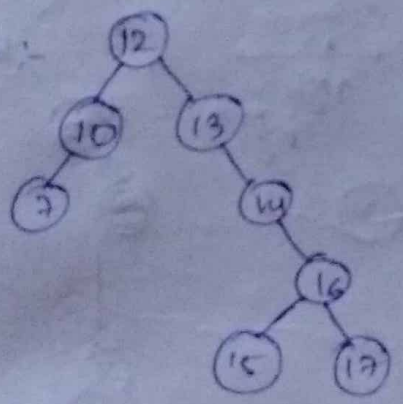
[Here 10 is already root of tree so no need to perform any splay operation]



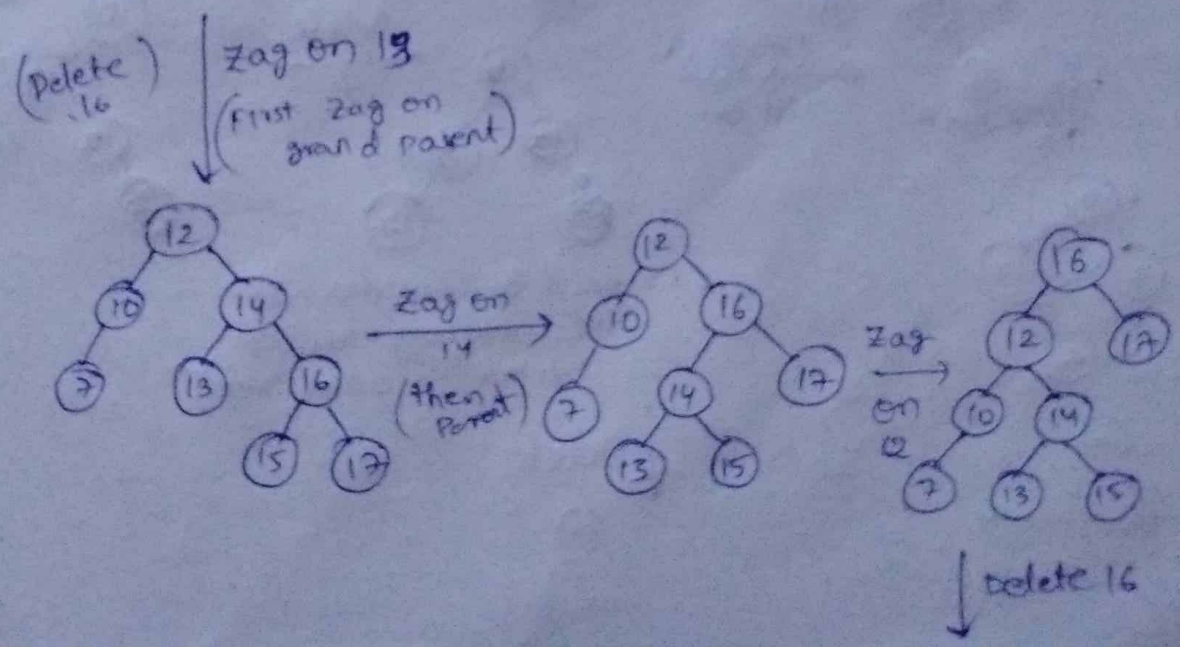
Top down splaying :-

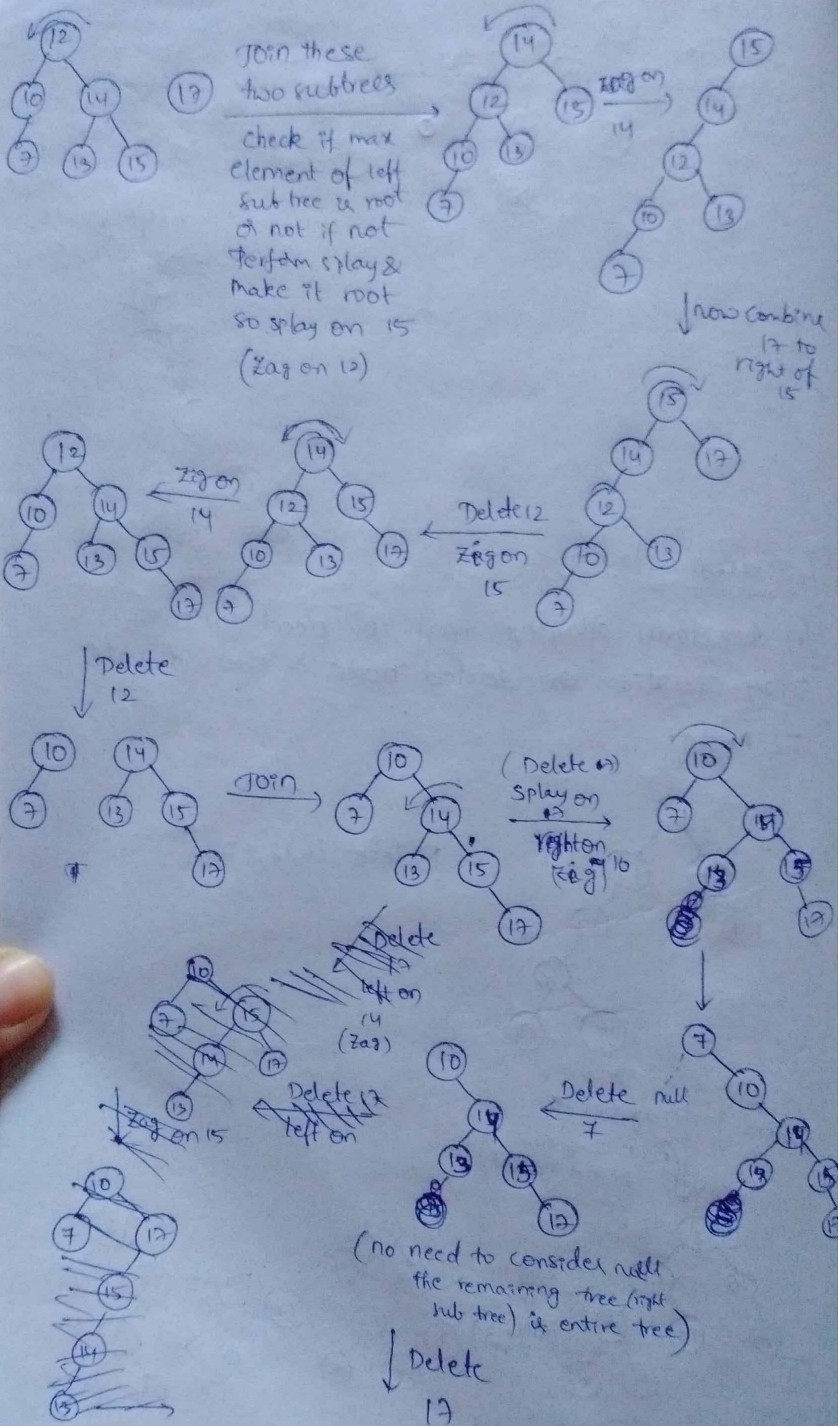
In top down splaying, first you need to perform splay operation on deleted node & then delete the node.

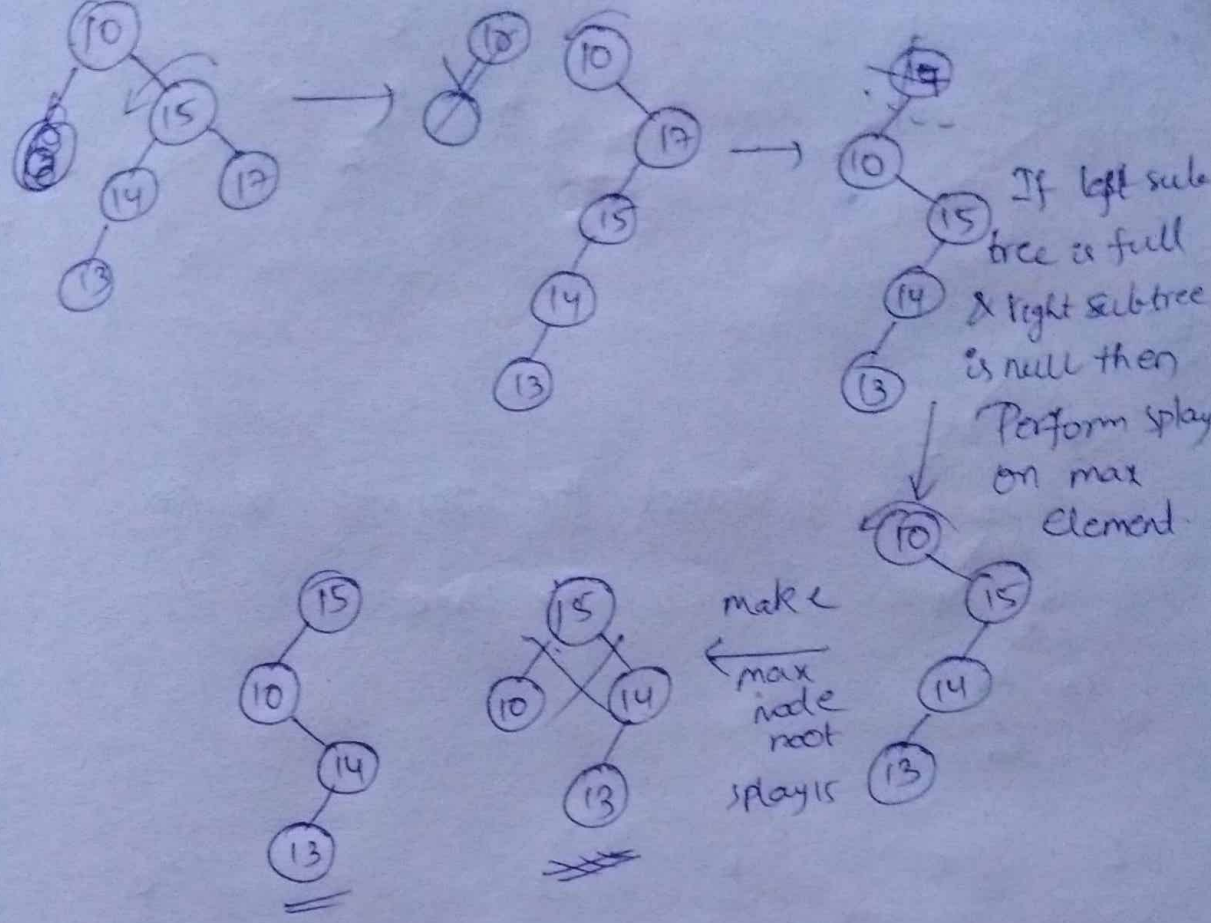
Q :-



Delete 16,12,17




Join these two subtrees
check if max element of left sub tree is root or not if not perform splay & make it root so splay on 15 (Zag on 12)
Zig on 14
now combine 17 to right of 15
Delete 12
Zig on 15
Zig on 14
Delete 12
Join
(Delete on) Splay on right on 10 (Zig)
Delete left on 14 (Zag)
Delete 17 left on
Zag on 15
Delete null 7
(no need to consider null the remaining tree (right sub tree) is entire tree)
Delete 17



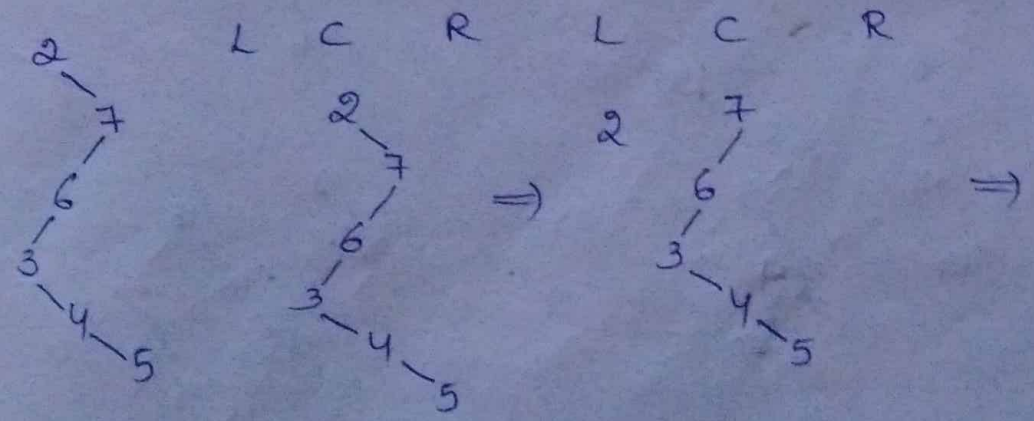
Top down splaying:-

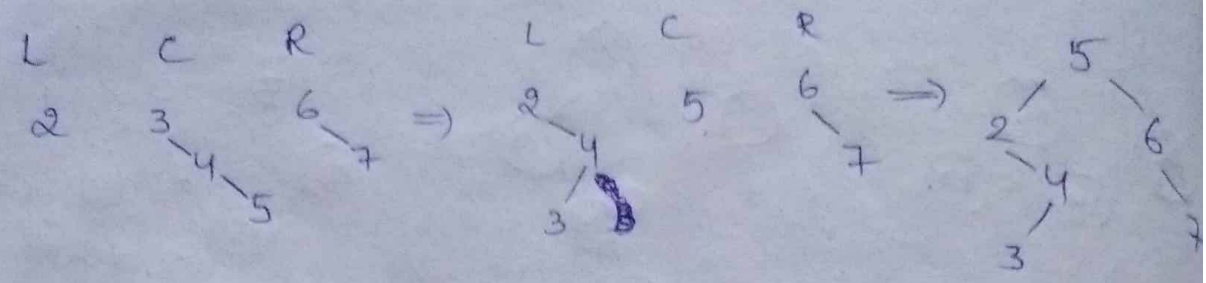
In the top down splaying the following four rotations are allowed:-

1) Zig
2) Zag
3) ZigZig
4) ZagZag

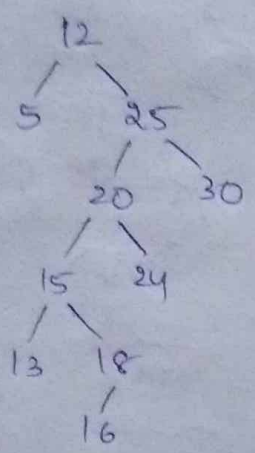
ZigZag, zagzig are not allowed in top down splaying.

Ex:

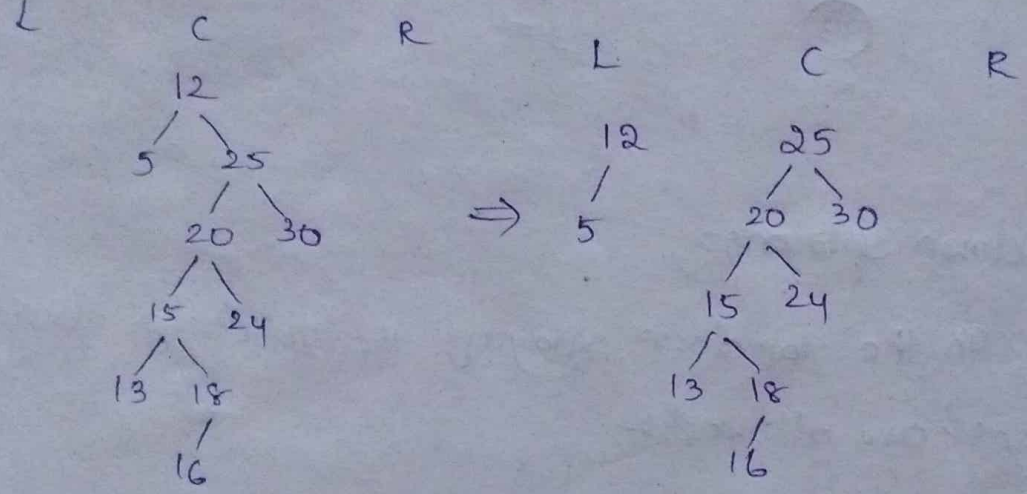


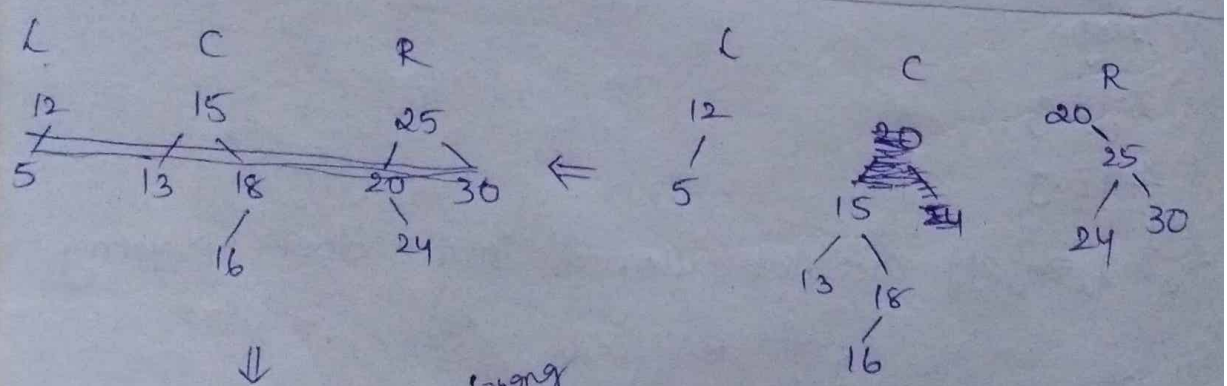


(2)

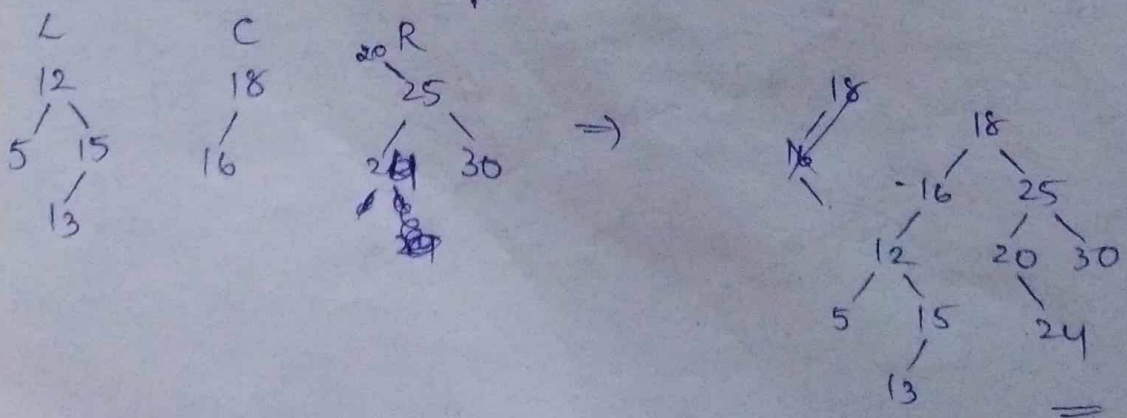


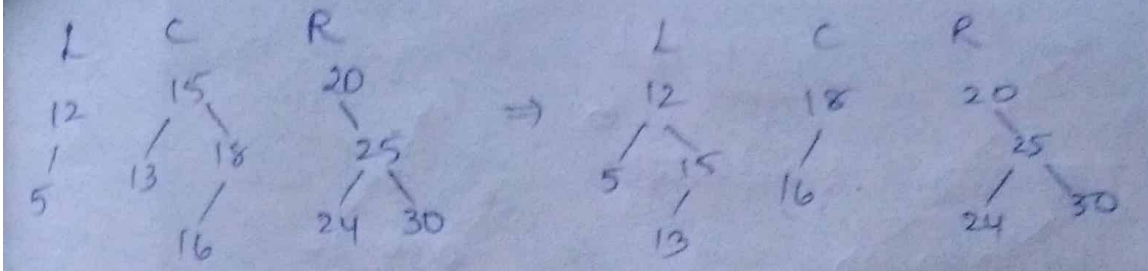
Search the element 18 in the tree.



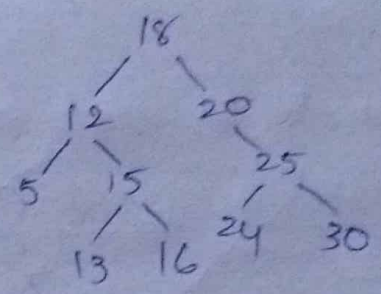


Wrong

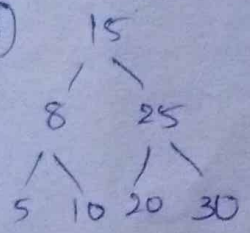




find max element

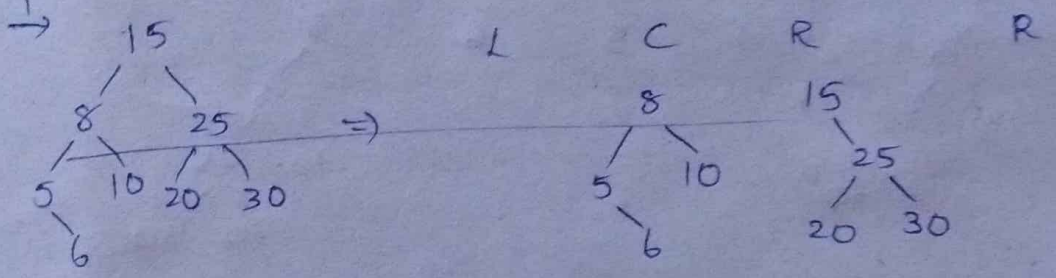


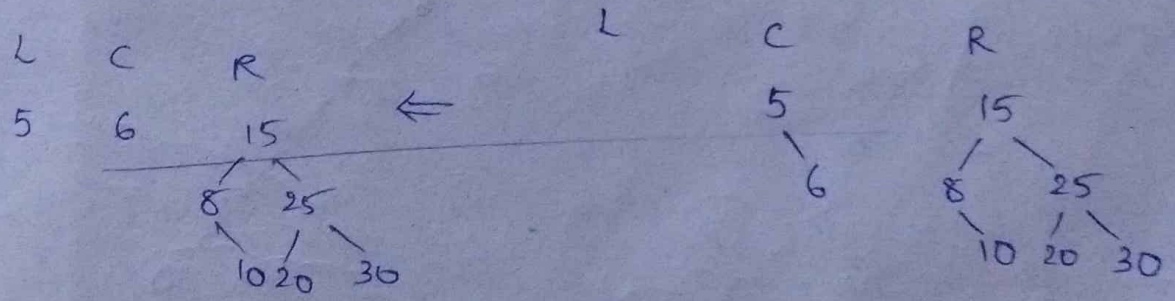
(3)

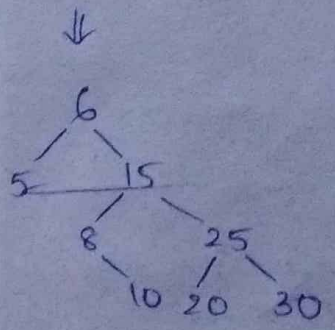


Insert 6, 12, 24

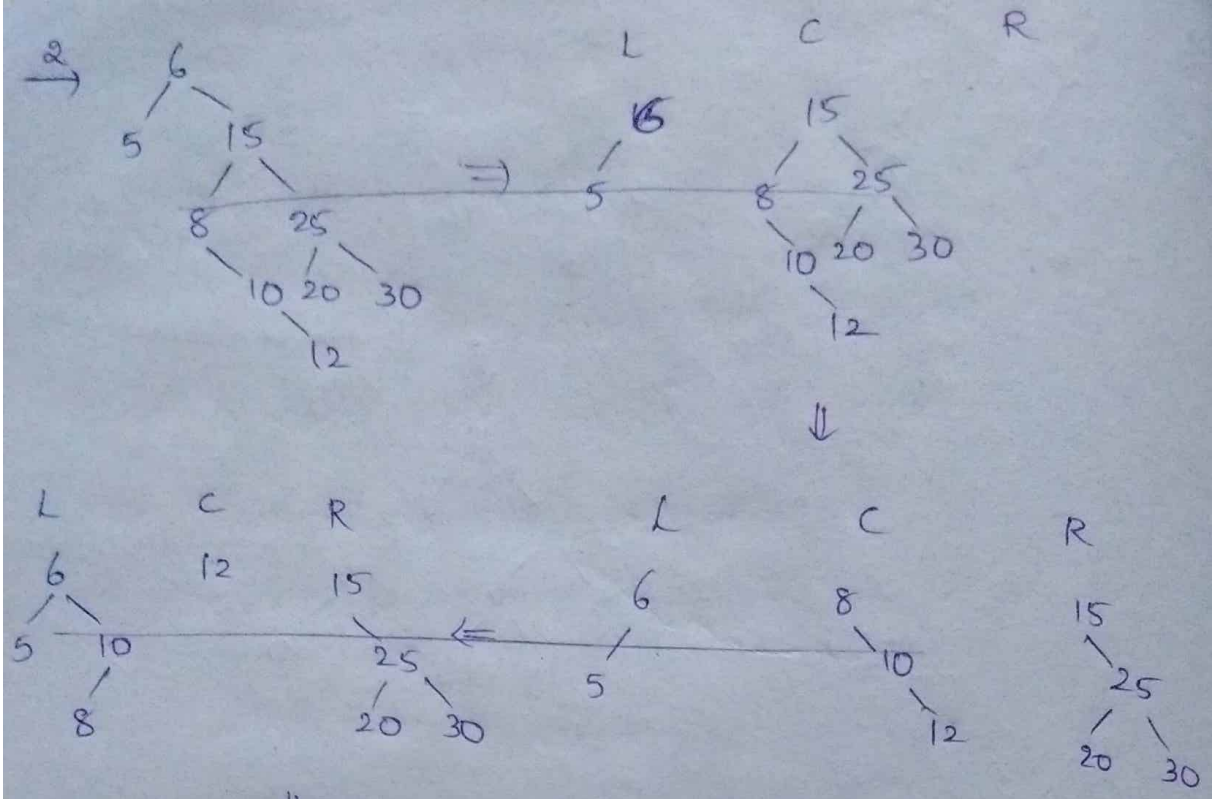
(Insertion as per BST & then splaying)



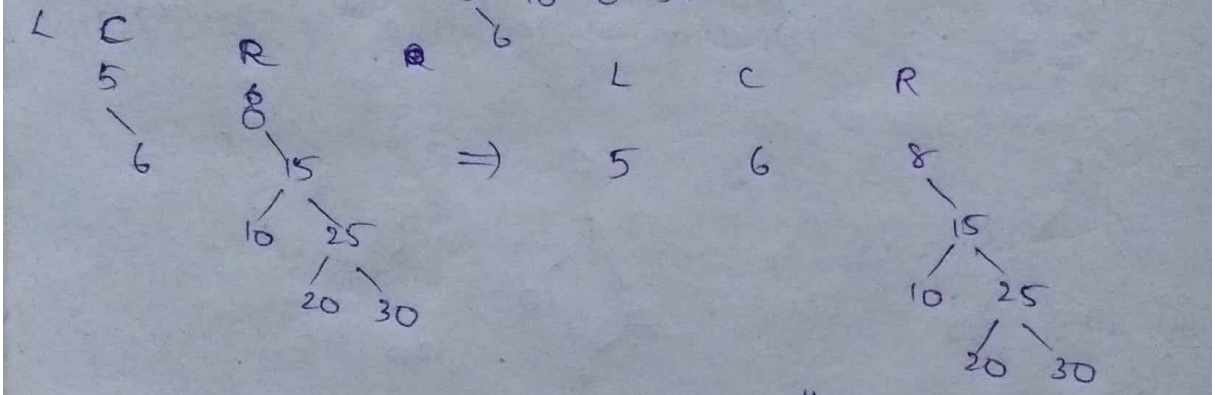




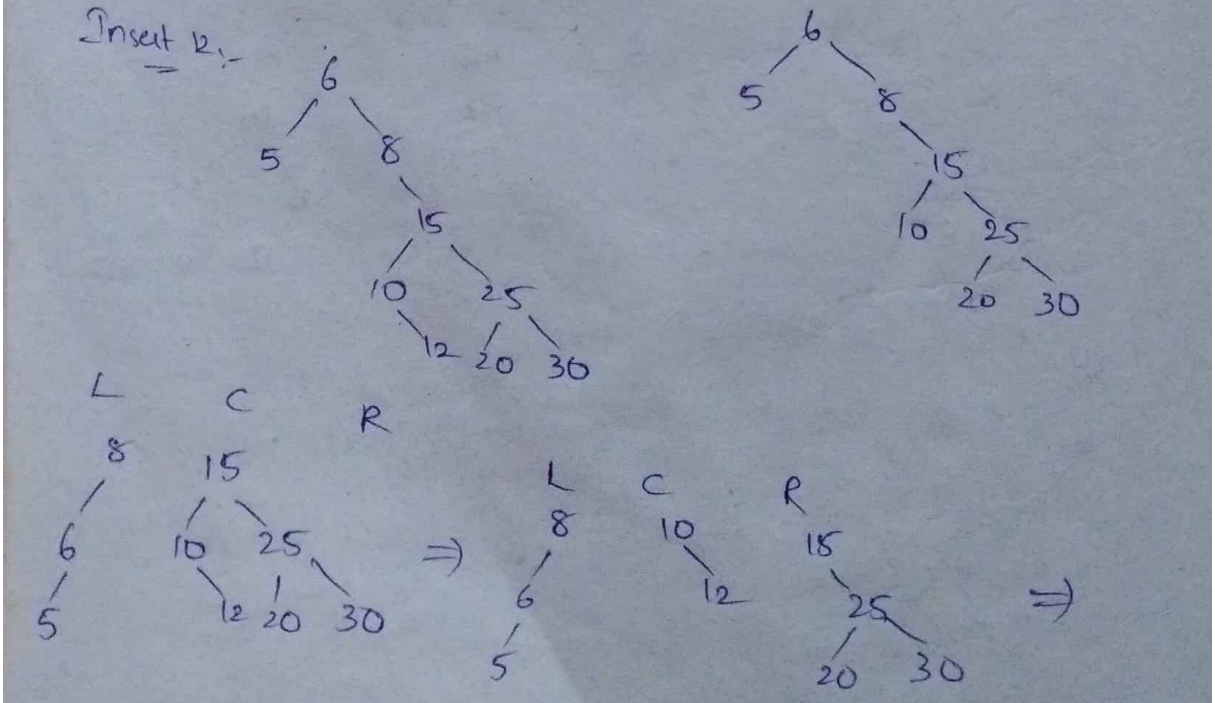
2 →

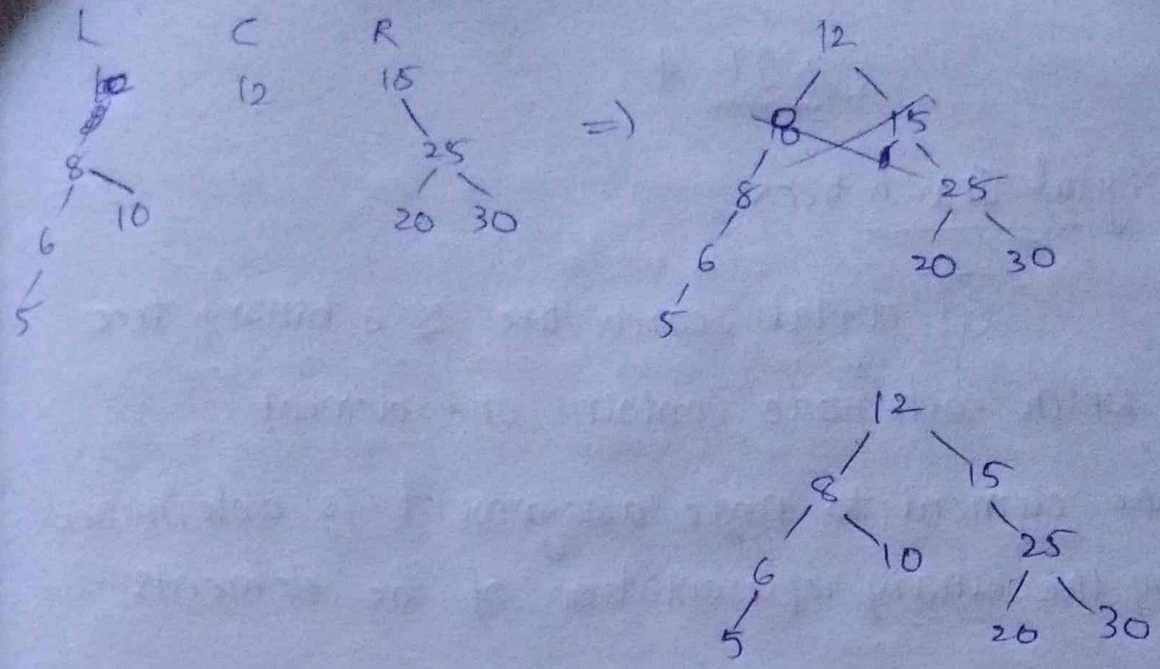


Insert 6 :-

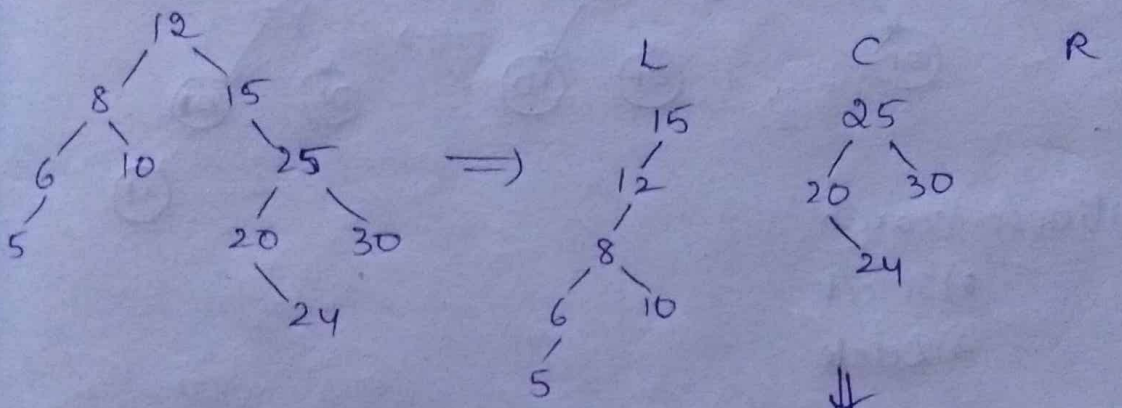


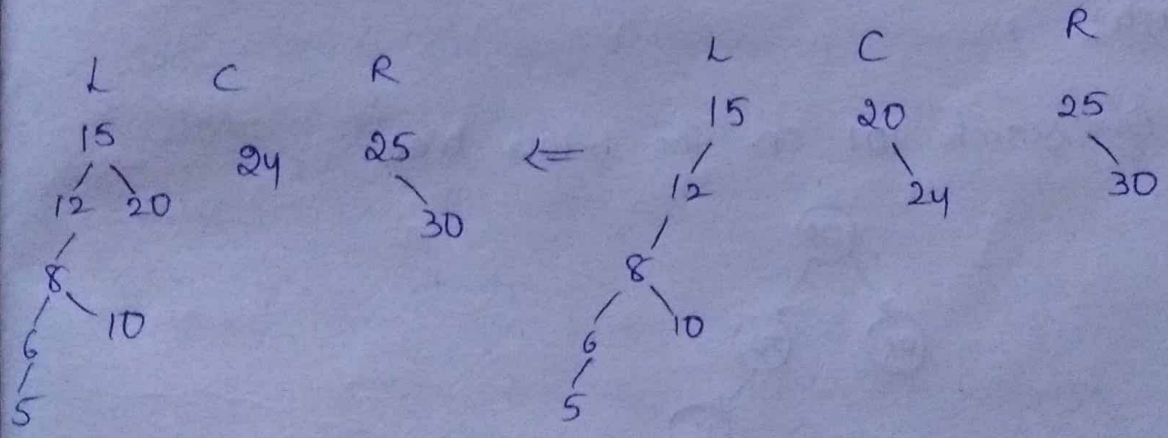
Insert 12 :-

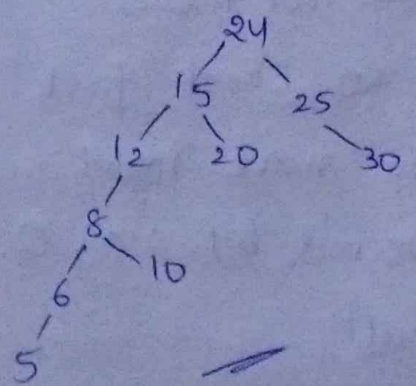



L
C
R
12
15
25
20
30
8
10
6
5
=>
12
8
15
25
20
30
6
5
12
8
15
10
25
6
20
30
5

Insert 24:-


12
8
15
6
10
25
5
20
30
24
=>
L
C
R
15
25
12
20
30
8
24
6
10
5


L
C
R
15
24
25
12
20
30
8
10
6
5
<=
L
C
R
15
20
25
12
24
30
8
10
6
5


24
15
25
12
20
30
8
10
6
5

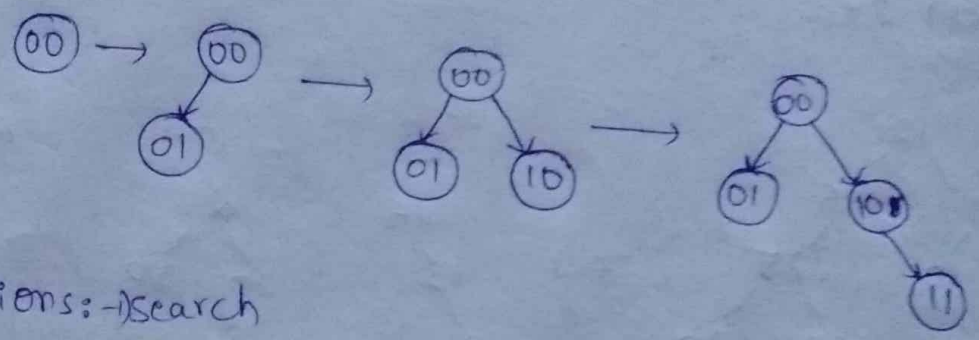
UNIT-4

Digital search trees :-

A digital search tree is a binary tree in which each node contains one element.

→ The element to node assignment is determined by the binary representation of the elements.

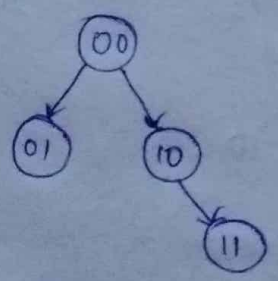
keys : 00, 01, 10, 11



Opeartions :- 1) search
2) Insert
3) Delete

Search :-

Ex :- search 101 in the below tree

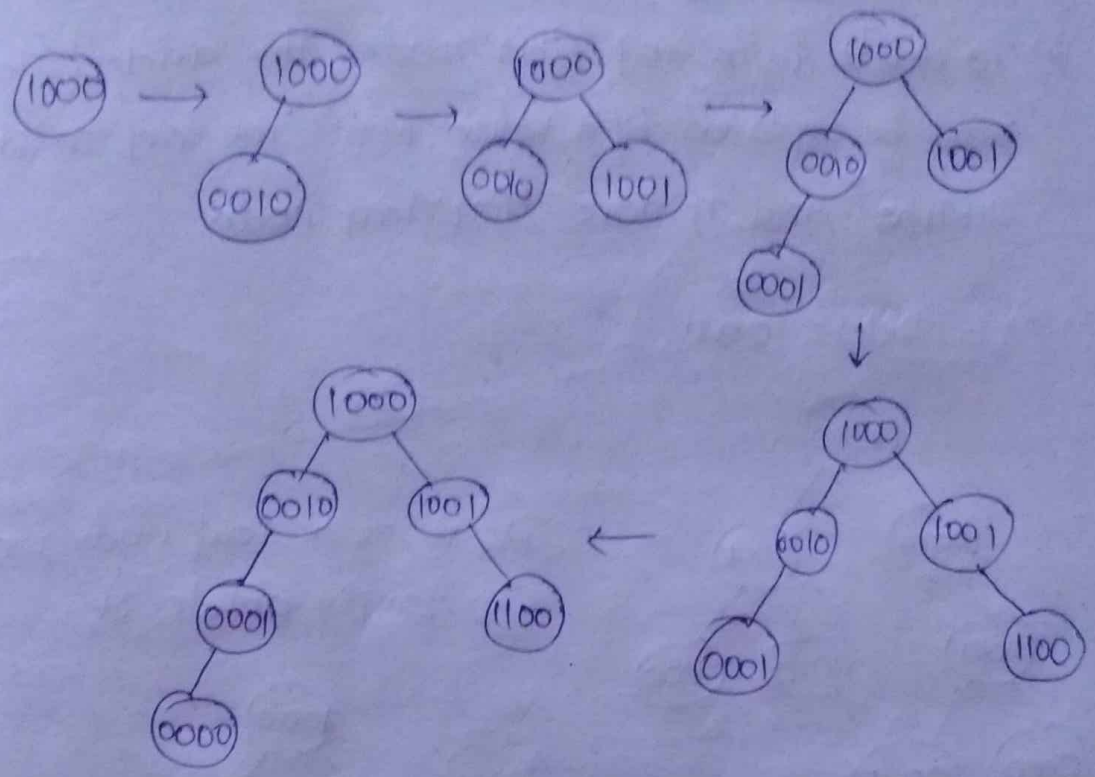


In 101 - first compare with root node if Not equal (00 ≠ 101) so check first bit of search key, it is 1 so move right compare 10 ≠ 101 so check second bit i.e, 0 so move left and it is null

Hence 101 is not present in digital search tree

→ Construct the DST with following keys:-

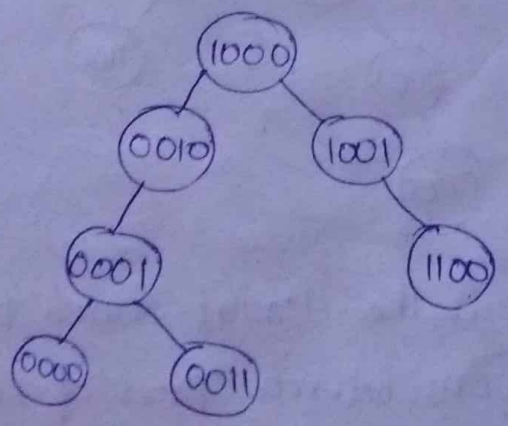
1000, 0010, 1001, 0001, 1100, 0000



Insert Operation :-

→ Before inserting the key in the tree, we should check whether the key is already present in tree or not, if not present then only we can insert (Duplicates are not allowed)

Insert 0011 in above constructed tree

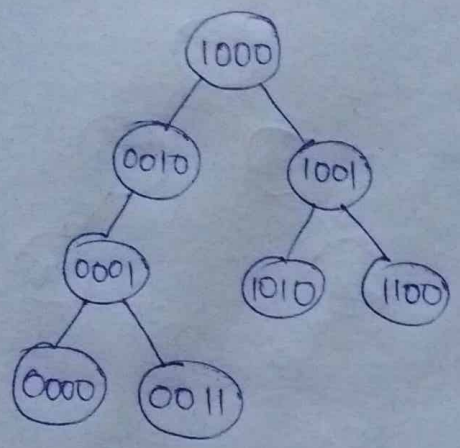


Delete operation :-

1) Deletion of an element in a leaf node is done by removing leaf node.

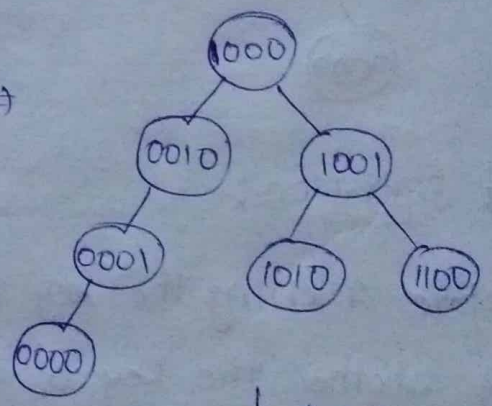
2) To delete from any other node, the deleted item must be replaced by a value from any leaf in its subtree and remove that leaf node

Eg :- Delete 0011

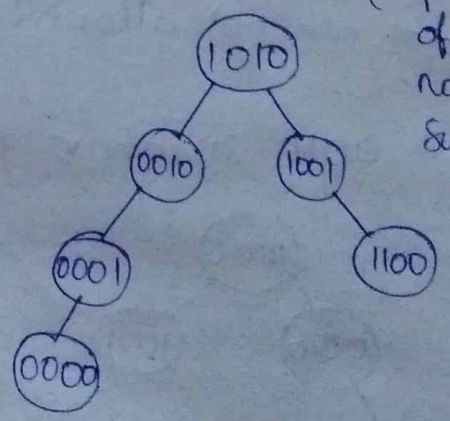


As it is a leaf node we can simply delete it.

⇒



↓ delete 1000 (Replace with any of the leaf nodes of its subtrees)



Since searching a key in the digital search tree requires many comparisions between pairs of keys, so digital search trees are inefficient search structures.

To reduce the no. of key comparisions done during searching a key is done by using a related structure called Particia

Particia :- Practical algorithm to retrive information Coded in alpha numeric.

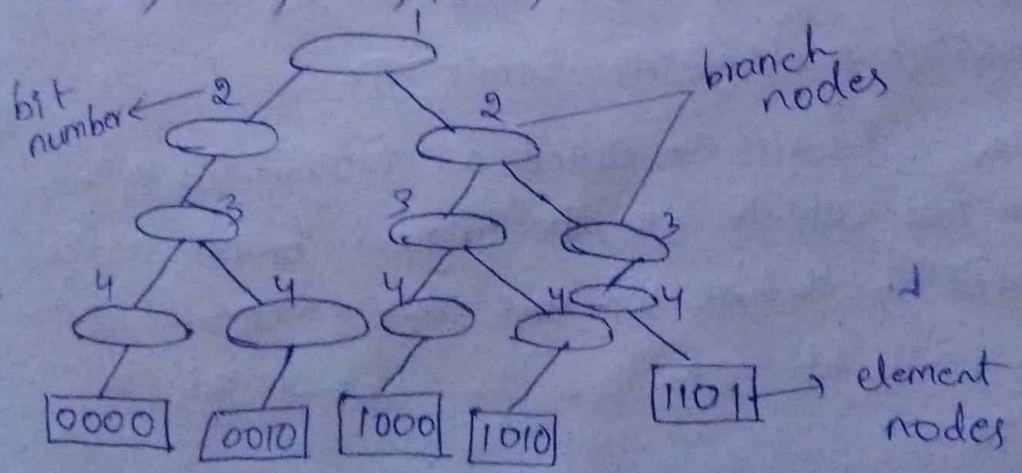
steps involved :- 1) Binary trie
2) Convert binary trie into Compressed binary trie
3) develop Particia from Compressed binary trie.

Binary Tree :-

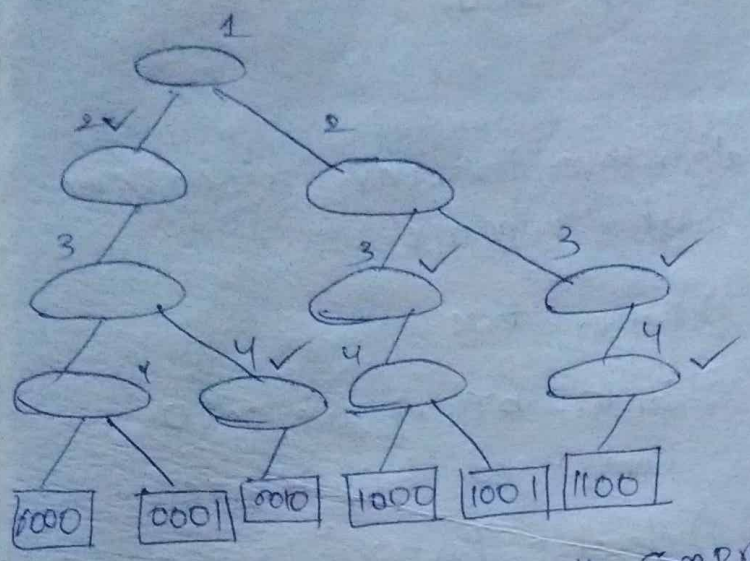
It is a binary tree which has two kinds of nodes :- 1) Branch nodes
2) Element nodes

- Branch nodes has two data members i.e., left child & right child
- Element node has single data member i.e, data
- Branch nodes are used to build a binary tree searc ~~set~~ structure similar to that of digital search tree.
- Representation of binary trie :-

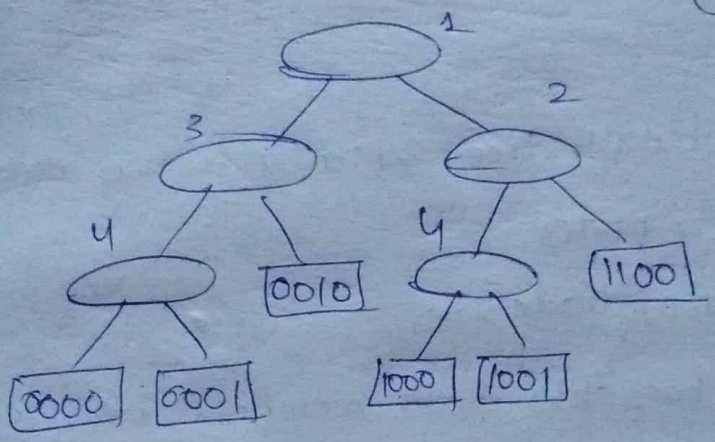
0000, 0010, 1000, 1010, 1101



Compressing a binary trie :-



⇓ Compression (Removing single degree branch nodes (i.e., branch nodes with single child))

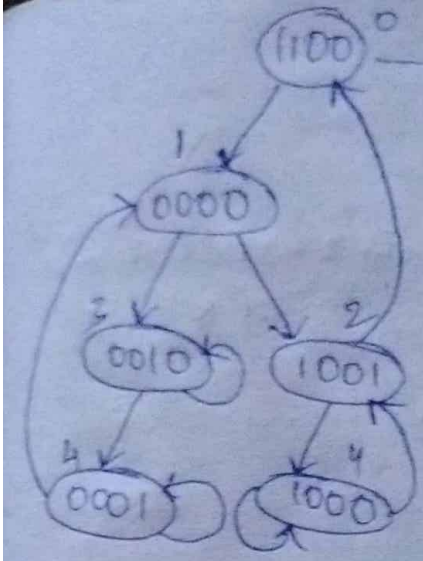


A compressed binary trie is a binary trie that has been modified in this way to contain no branch nodes of degree 1

Patricia :-

Compressed binary tries may be represented using nodes of ~~single trie~~ single type.

→ The new nodes are ~~called~~ called as Augmented Branch ~~nodes~~ nodes which are the original branch nodes augmented by ~~the data~~ the ~~rule~~ data ~~non~~ member data. So the resultant structure is known as Partitia.

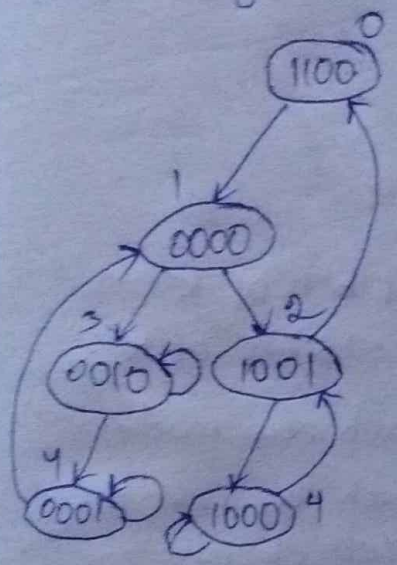


→ header node

This node doesn't contain Right child

Operations of Patricia:-

1) Searching



K = 0000

Path = 0000 - 1100, 0000, 0010, 0001, 0000

K = 1100

Path = 1100 - 1100, 0000, 1001, 1100

K = 0011

1100, 0000, 0010 → searching failed no element in tree

K = 1011

1100, 0000, 1001, 1000, 1001

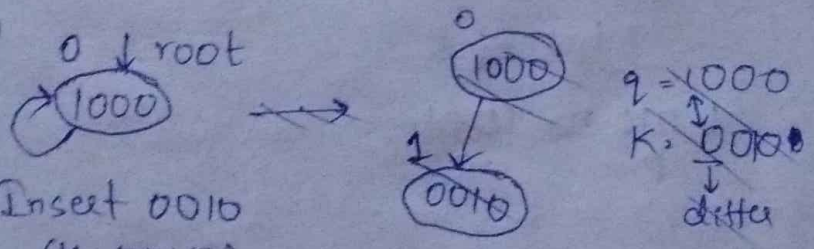
↳ Searching failed.

(Searching is Performed until the bit less than or equal to Previous bit is found)

2) Inserting the following heap into the Patricia:-

1000, 0010, 1001, 1100, 0000, 0001

①



Insert 0010
(K = 0010)

→ search for Key in Previous Patrica

Search stopped at 1000

$\therefore q = 1000$

$K = 0010$

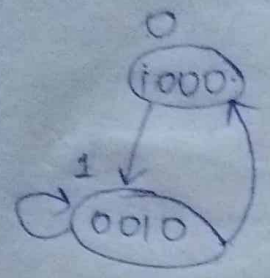
First differed bit in q and K is 1 → Bit number of K is 1

$\Rightarrow j = 1$

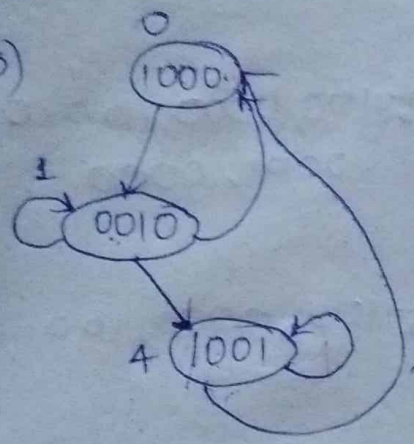
Now search for j-1 bits of K.

i.e, $1 - 1 = 0$

2)



3)



$K = 1001$

$q = 1000$

$j = 4$ j-1 bits of K

$\Rightarrow$ 3 bits of K

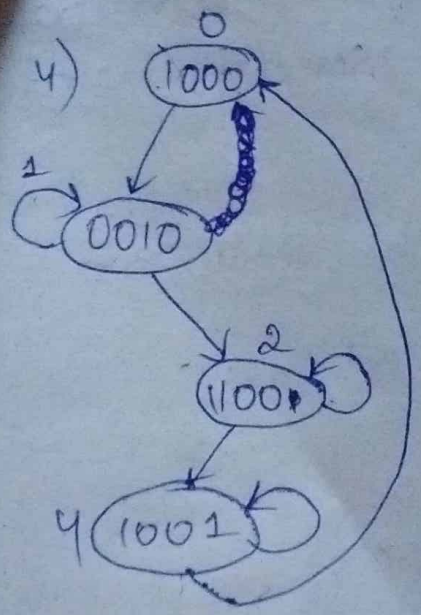
Predecessor – attach new node

successor – attach the right or left child

0010 to 1000

P S

4)



$K = 1100$

$q = 1000$

$j = 2$

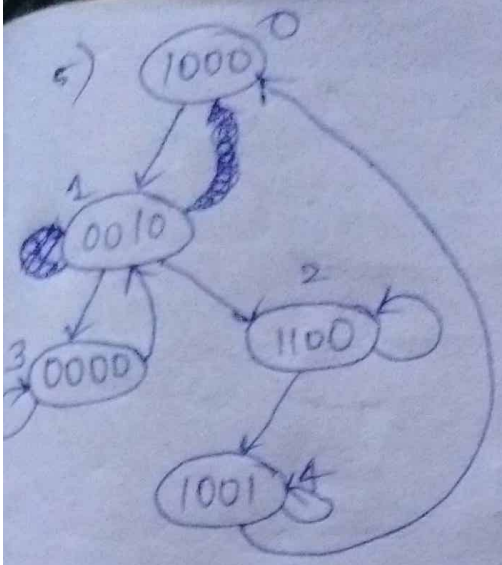
$j - 1 = 1$ bit

1 bit of K

0010 to 1001

P S

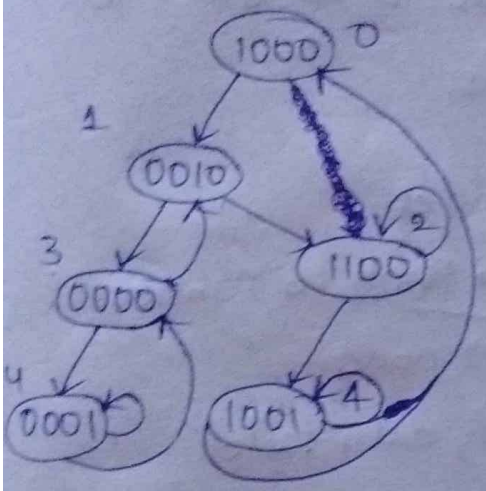
5)



k = 0000
q = 0010
j = 3
j-1 bits = 2 bits of k

0010 — P
s

6) 0001

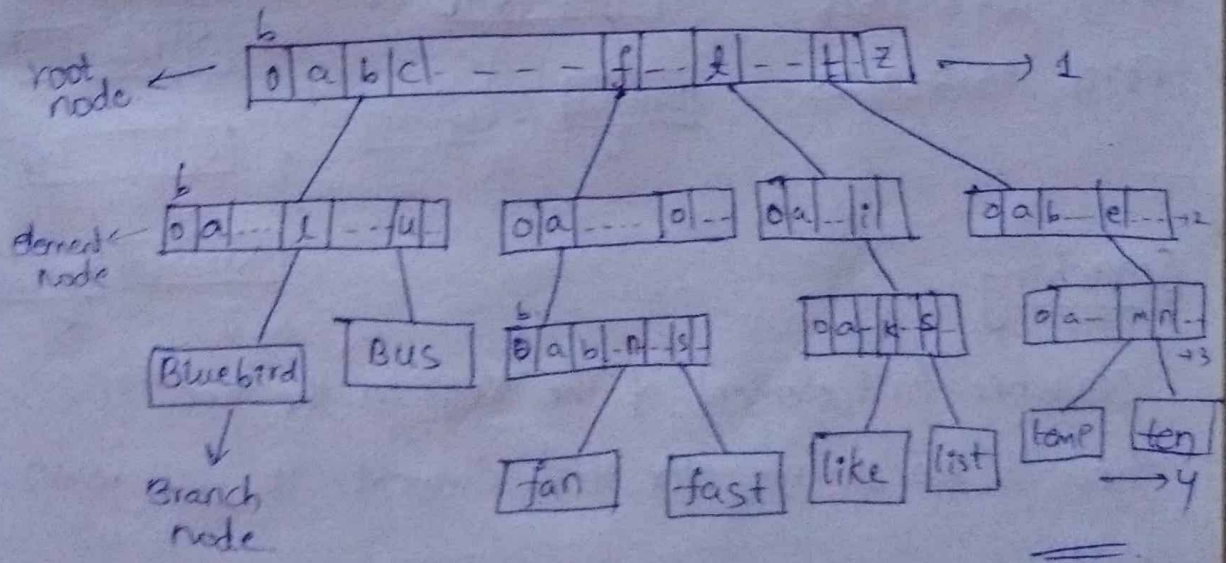


k = 0001
q = 0000
j = 4
j-1 = 3 bits of k

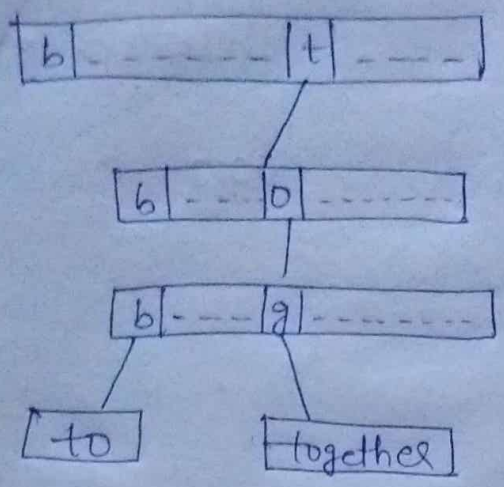
P and s - 0000

Multiway tries :-

S = {Bluebird, Bus, fan, fast, like, list, ten, temp}



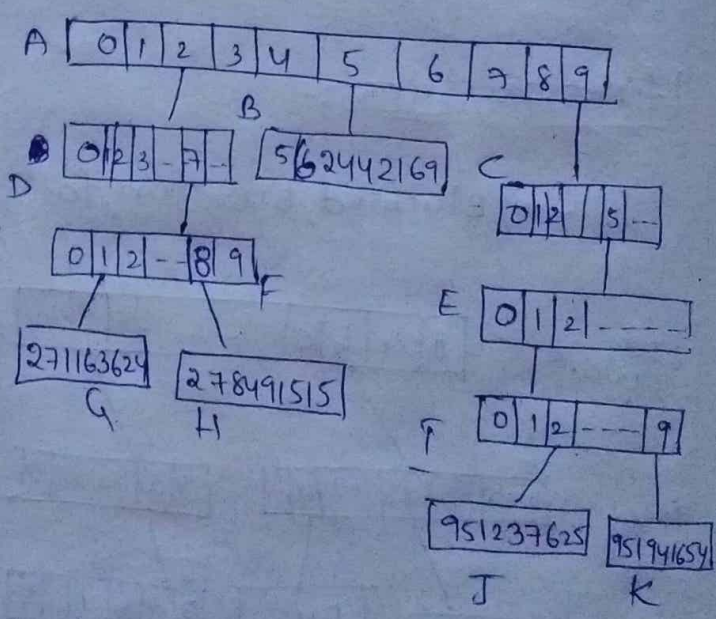
Q-2 :- S: {to, together}



Blank is used whenever there is no other letters remain in any one of the words

2 types of nodes :- 1) Branch nodes
↳ Points to subtrees
2) Elementary nodes
↳ only data field.

Name	SSN
Jack	951941654
Jill	562442169
Bill	271163624
Kathy	278491515
April	951237625



Searching :-

Searches first element of the key in the tree if there is no subtree for that element it will return null.

Q. 941237625

9 - subtree present
↓
4 - no subtree

c.link['y'] = Null

Sampling strategies

To reduce no: of levels in multiway trie we use some strategies, and those are called as sampling strategies

$sample(x, i) \longrightarrow x_{n-i+1}$, $n \rightarrow$ size of key, $i \rightarrow$ level

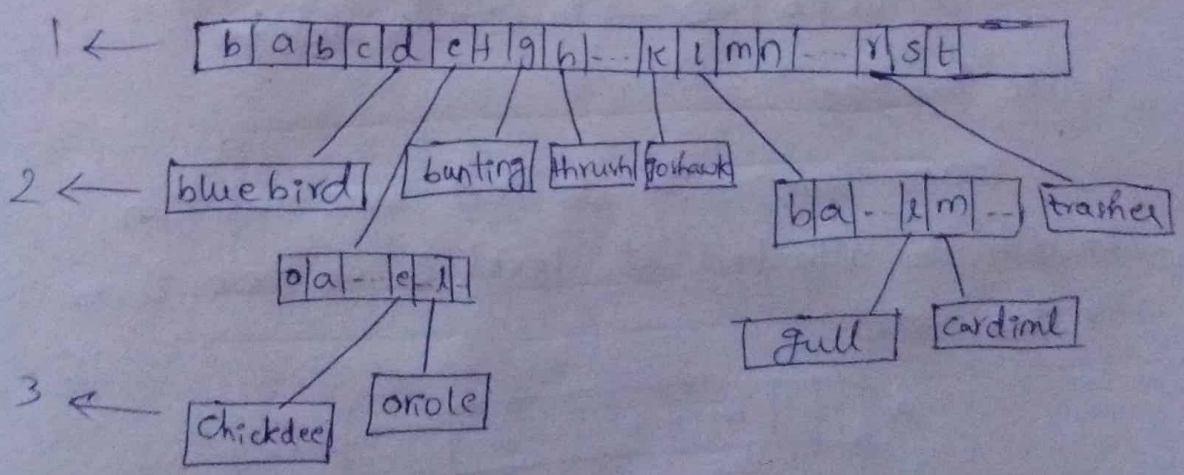
$sample(x, i) \rightarrow x_{r(x_i)}$, $r \rightarrow$ random

$$sample(x, i) \longrightarrow \begin{cases} x_{i/2}, & i \text{ is even} \\ x_{n-(i-1)/2}, & i \text{ is odd} \end{cases}$$

Ex:- S = {bluebird, bunting, chickdee, oriole, thrush, goshawk, cardinal, wren, gull, trasher} $l = 1$

x_i bluebird $\longrightarrow$

x_{n-i+1} bluebird $\longleftarrow$ (from last index)

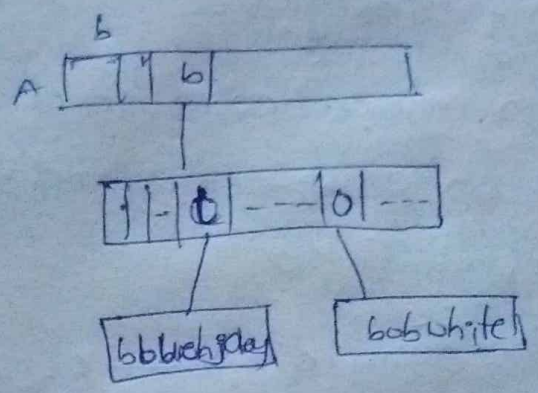


$x_{r(x_i)} \longrightarrow$ 'random index' is taken and moved towards left from that index.

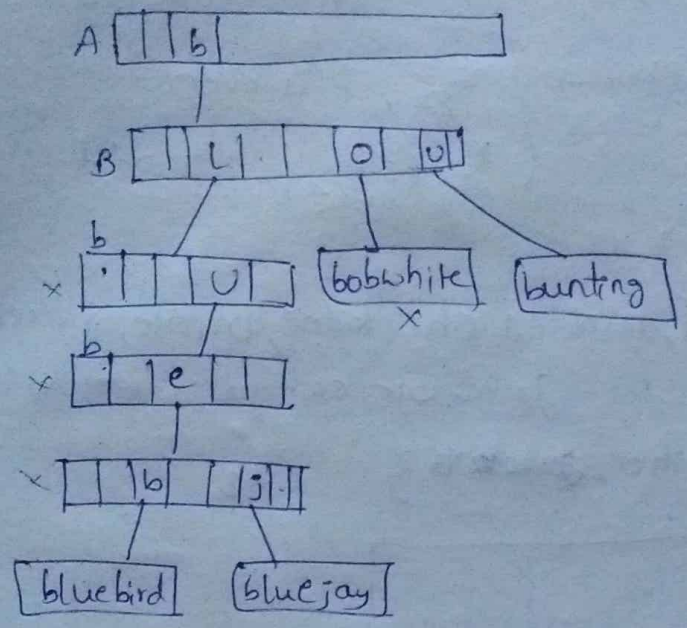
Insert S = { bobwhite, bluejay }

A.link['b'] != 0
B.link['L'] = 0

A.link['b'] != 0
B.link['o'] = 0



→ 1

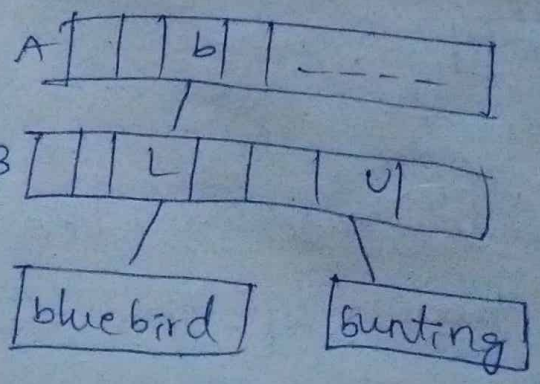


Delete 'bob white'

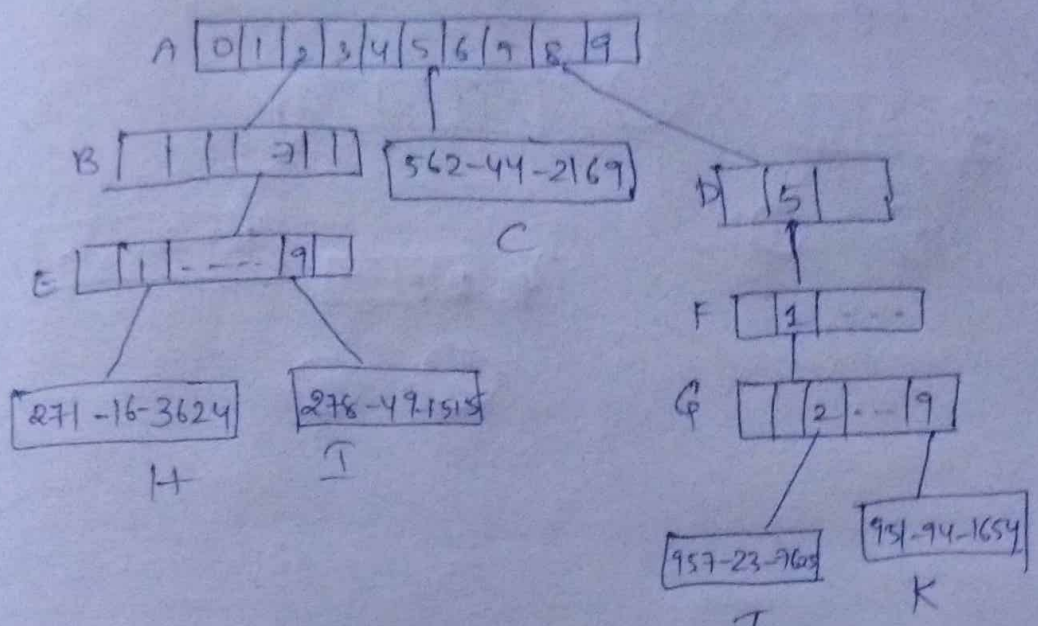
B.link['o'] = 0 → so simply delete

Delete 'bluejay'

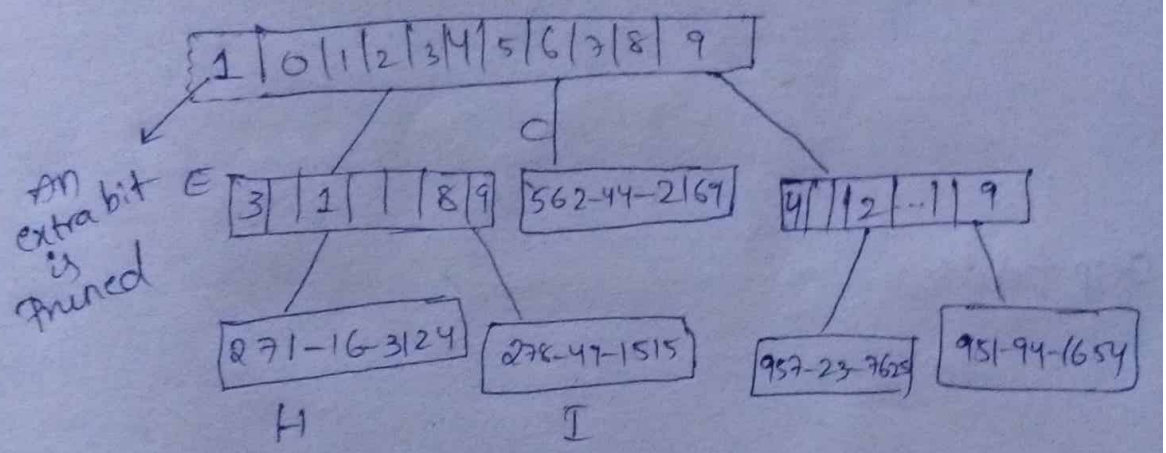
Since bluejay is deleted bluebird can be directly attached at level B as there is no similar word.



Compressed tries:-



Remove fields while single child



Searching:-

9451-23-7625

dN=1

A.child['9'] = G

G.child['2'] = J

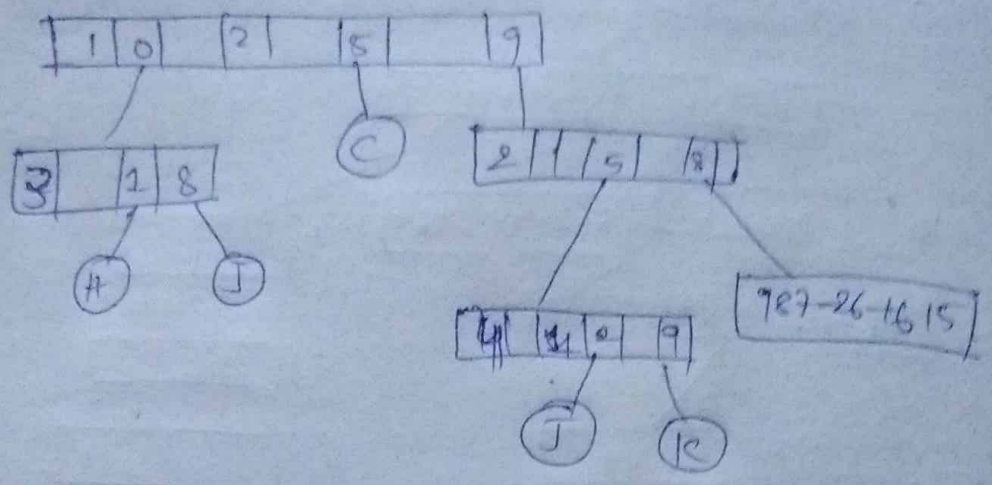
If J = search key – Found.

Insertion:- 987-26-1615

Search element in tree

J ≠ search element

J = 957-23-7625 → Check number that is differing.



Skip fields:-

Knuth Morris Pratt (KMP) Algorithm :-

→ Also known as straight forward Algorithm

→ Given a string, find a pattern whether present in the string or not

→ for linear search it takes $O(mn)$ time complexity

m - length of string
n - length of pattern

String - abcdefgh
Pattern - def

We use 2 pointers and search for the pattern.

KMP:

Eg:- String: abcdabcab
Prefixes: a, ab, abc, abcd, abcda ....
Suffixes: b, ab, cab, bcab, abcab ...

π Table

a	b	c	d	a	b	c	a	b	f
0	0	0	0	1	2	0	1	2	0

Ex:-2

1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	3

Ex:-3

a	a	a	a	b	a	a	c	d
0	1	2	3	0	1	2	0	0

String	a (i)	b	a	b	c	a	b	c	a	b	a	b d

Pattern π :-	a (j)	b	a	b	d
	0	0	1	2	0

Compare : 'i' and 'j+1'

If there is match move both the pointers.

If there is mismatch ~~move j +~~, see the π value of j

and move j to that value

If we reach 0 ~~then~~ for j, then we move i value and again do the Process.

We found j==m

Algorithm:-

Step 1:- Draw a 'π' table for the given pattern

Eg:- P a a b c a d a a b e

0 1 0 0 1 0 1 2 3 0

Step 2:- Place the Pointers 'i' & 'j' on string and Pattern respectively

Step 3:- Compare 'i' and 'j+1'

if same move both 'i' and 'j' to its next index.

else: 'i' - will remain as it is

'j' - updated to its Pie(π) value

Step 4:- Repeat this step until j reaches the length of the pattern.

KMP (P-pattern of length 'm'
T-string of length 'n')

Preconditions $1 \le m \le n$

```
j ← 0
i ← 0
while(i<n)
{
  if (P[j+1] = T[i])
  {
    j ← j+1
```

```
        i ← i+1
        if (j==m)
        {
          output :- S[i-j] → Pattern
        }
      }
      else
      {
        if (j==0)
        {
          i ← i+1;
        }
        else
        {
          j ← π(j)
        }
      }
    }
```

Time Complexity

$= O(m+n)$

m → to generate 'π' table

n → to search.