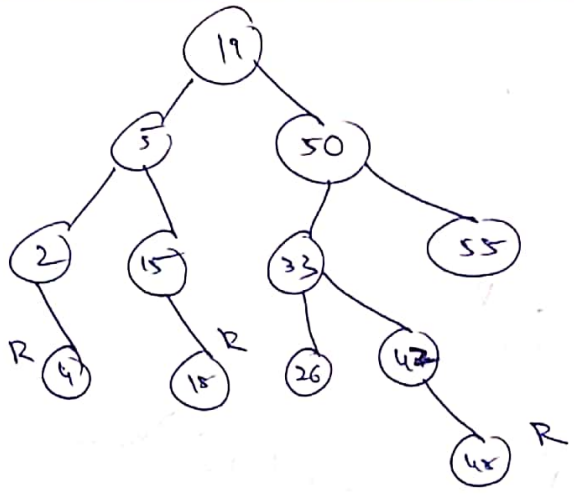
1) Node

```
{ int val;
  node * next;
}
Main()
{
  Bool ret;
  ret= Bad (head, head.next)
}
Bool Bad (Node n1, Node n2)
{
  if (n1==n2)
    return true;
  else if (n2== null)
    return false;
  else if (n2.next == null)
    return false;
  else return Bad (n1.next, n2. next.next);
```

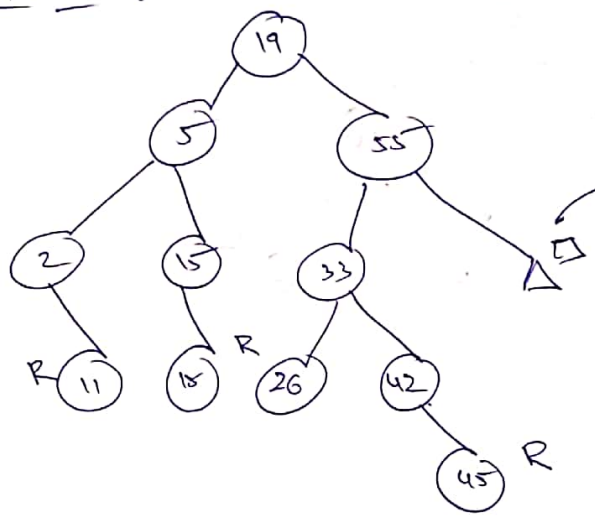
2) Algorithm :-

```
bool heap(node *root1)
{
  int i = 0
isheap(root1, i, sizeof(root));
}
bool isheap (node *root, int i; int n)
{
  if (root == null)
    return true;
  if (i >= n)
    return false;
  if((root -> lchild && root -> <= root -> val)
    (&) (root -> rchild && root.val ≤ root -> val))
    { return false;
    }
  return isheap (root -> lchild, 2*i+1, n) &&
         isHeap( root -> rchild, 2*i+2, n);
}
int size (node *root)
{
  if (root == null)
  { return 0;
  }
  return 1+ size (root -> lchild) + size (root -> rchild)
}
```

3) (a)

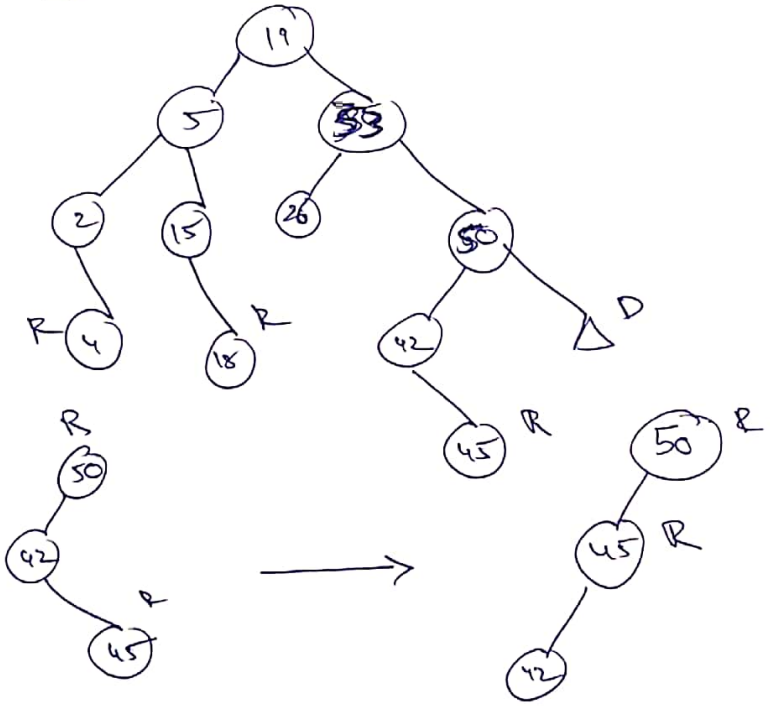


Delete 55 :-



Replace with null we have an imbalance of black nodes. So put black triagle.

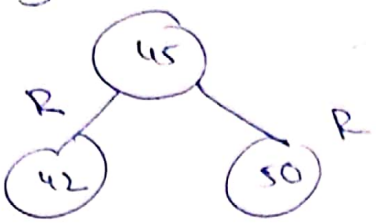
-> Now check Sibling and nepleco. Sice sibiling is Red, rotate. null node and it remains to the right of 50 with the black square.

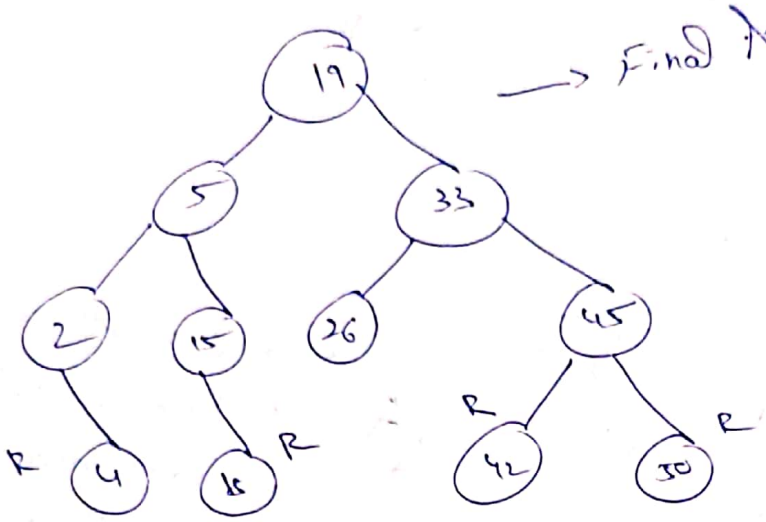


:

:- It is in Zig-Zag

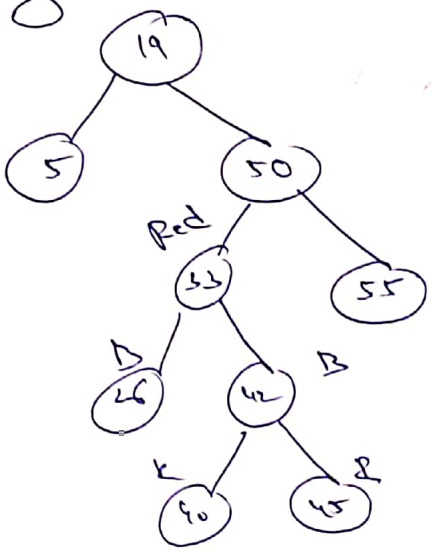
2. Straight it up





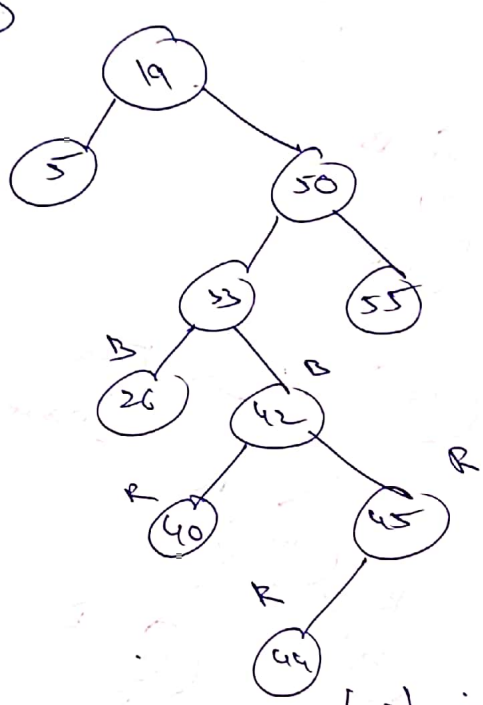
→ Find Answer.

3)(b) Inserting 40



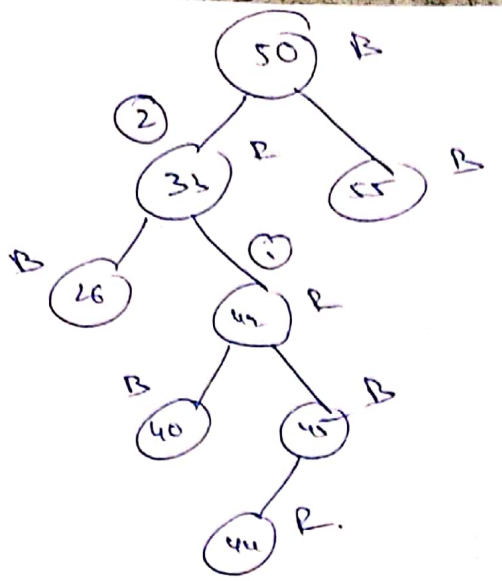
-> Here node Red.

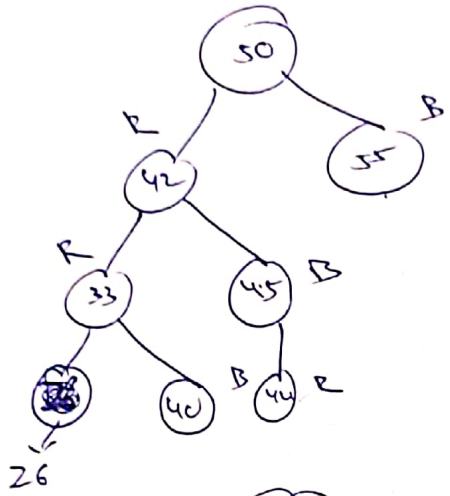
Inserting 44

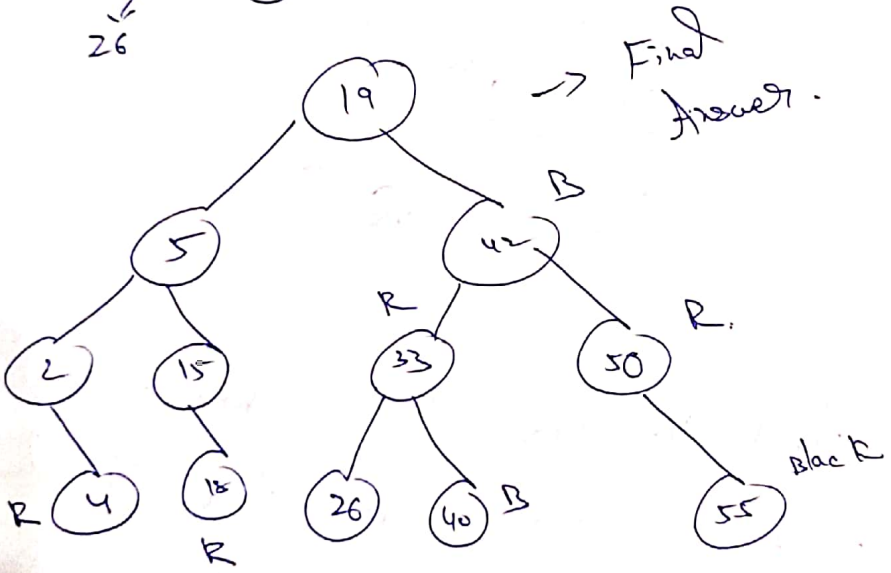


Check the node if it is Red (or) Black.

-> Since the node is red change the parents. to black and grand parents to red.







4) 1 -> -1 -> -2 -> 6 -> 10

2 -> -1 -> 8

3 -> -1 -> 6

4 -> -1 -> -2 -> 10

5 -> -1 -> 4

6 -> -1 -> -2 -> -3 -> -4 -> -5

7 -> -1 -> 1

8 -> -1 -> -2 -> 10

9 -> -1

10 -> -1 -> -2 -> -4 -> -6 -> -8

11 -> -1 -> 6

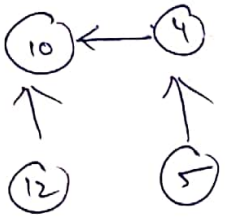
12 -> -1 -> 10

(1) Union (4,5)

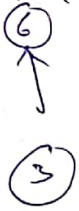
(2) Union (10,12)

(3) Union (1,7)

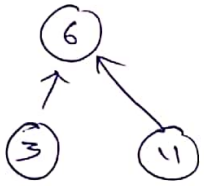
(4) Union (12,5)



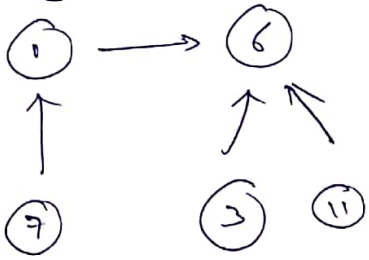
(5) Union (6,3)



(6) Union (3,11)

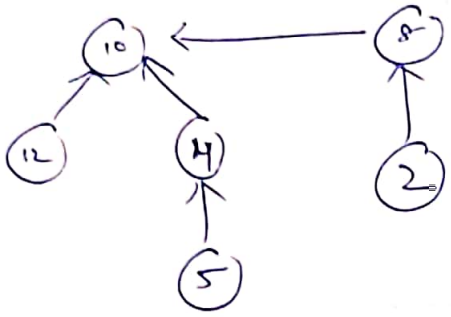


(7) Union (7,11)

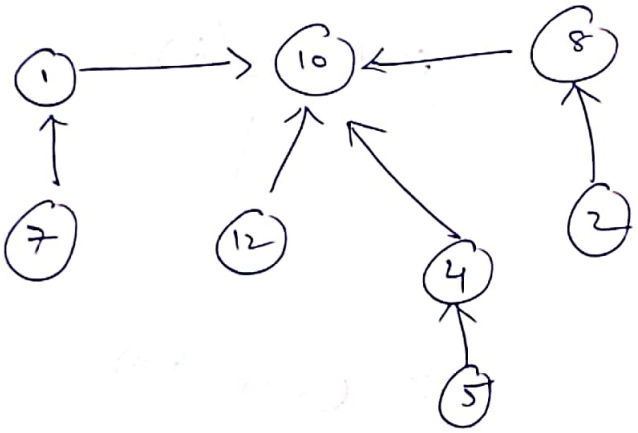


8) Union (8,12)

9) Union (12,2)



10) Union (7,12)



5) Given

3776, 1248, 0159, 0378, 0127, 0047, 4495, 0077,
3218, 0119, 0061, 1416, 3225

RADIX Sort LSD :

→ We need to Sort for maximum of 4 times.

0
1 0061,
2
3
4
5 4495, 3225
6 3776, 1416,
7 0127, 0047, 0077
8 1248, 0378, 3218
9 0159, 0119,

After 1 iteration we have

0061, 4495, 3225, 3776, 1416, 0127, 0047, 0077, 1248,
0378, 3218, 0159, 0119

0
1 1416, 3218, 0119
2 3225, 0127,
3
4 0047, 1248
5 0159
6 0061,
7 3776, 0077, 0378,
8
9 4495

After second iteration we have

1416, 3218, 0119, 3225, 0127, 0047, 1248, 0159, 0061,
3776, 0077, 0378, 4495

0 0047, 0061, 0077
1 0119, 0127, 0159
2 3218, 3225, 1248
3 0378
4 1416, 4495
5
6
7 3776,
8
9

After third iteration

0047, 0061, 0077, 0119, 0127, 0159, 3218, 3225, 1248,
0378, 1416, 4495, 3776

0 0047, 0061, 0077, 0119, 0127, 0159, 0378
1 1248, 1416
2
3 3218, 3225, 3776
4 4495
5
6
7
8
9

After 4th iteration, we get the sorted data.

0047, 0061, 0077, 0119, 0127, 0159, 0378, 1248, 1416,
3218, 3225, 3776, 4495

6) Given

Table size = 7

164, 378, 289, 94, 821, 632, 200, 120

:> 164 = 11 %. 7 = 4

378 = 18 %. 7 = 4 -> Already occupied => 4+1

289 = 19 %. 7 = 5 -> Already occupied => 5+1

0	
1	
2	
3	
4	164
5	378
6	289

For inserting 94, the load factor will be greater than. 0.5, so increasing table size to next prime number after doubling => 17

164 => 11 %. 17 = 11

378 => 18 %. 17 = 1

289 => 19 %. 17 = 2

94 -> 13 %. 17 = 13

821 => 11 %. 17 = 11 -> occupied

=> 11+1

632 -> 11 %. 17 = 11 -> occupied

=> 11+1 = 12 -> occupied

=> 11+2 = 13 -> occupied

=> 11+3 => 14

200 -> 2 %. 17 = 2 -> occupied

=> 2+1 = 3

120 -> 3 %. 17 = 3 -> occupied

3+1 = 4

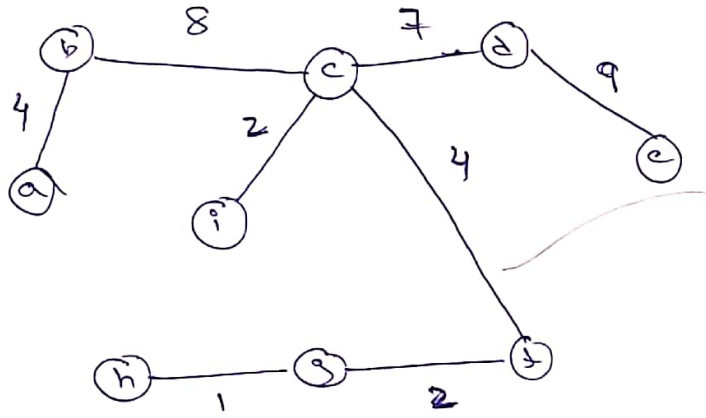
Final answer.

0	
1	378
2	289
3	200
4	120
5	
6	
7	
8	
9	
10	
11	-> 164
12	-> 821
13	-> 94
14	-> 632
15	
16	

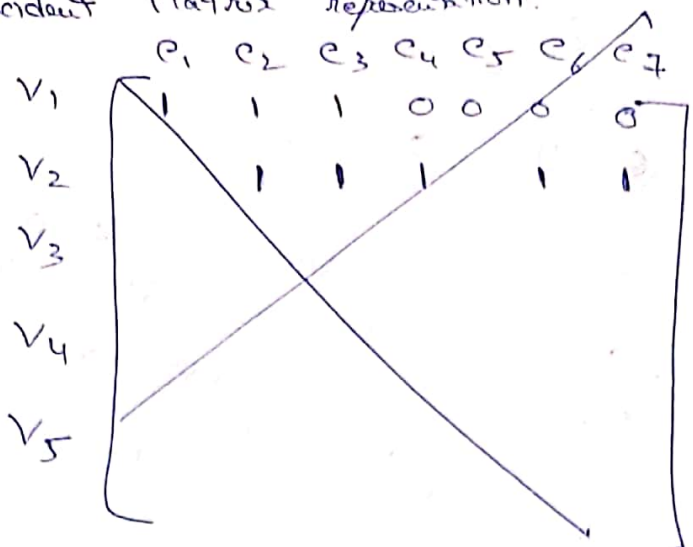
7) Sort the edge distance of vertices on ascending order.

(h, g) = 1
(g, f) = 2
(c, i) = 2
(a, b) = 4
(c, f) = 4
(i, g) = 6
(h, i) = 7
(c, d) = 7
(b, c) = 8
(a, h) = 8
(d, e) = 9
(e, f) = 10
(b, h) = 11
(d, f) = 14

Now Start joining vertices by considering minimum edge weights which doesn't form any loops

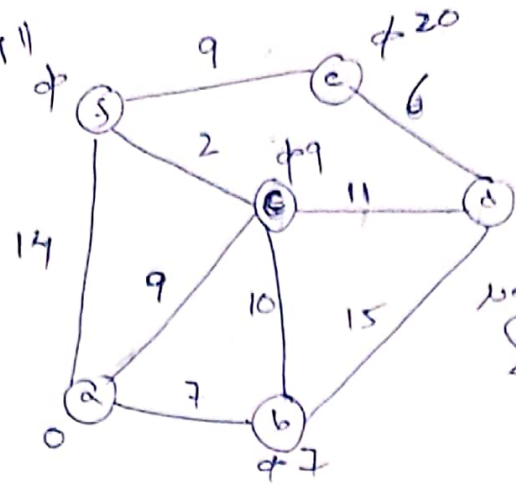


8) Incident Matrix representation.



	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	1	1	0	0	0	0	0
v_2	0	1	1	1	0	1	1	0
v_3	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	1	1
v_5	0	0	0	0	1	1	0	0

9)



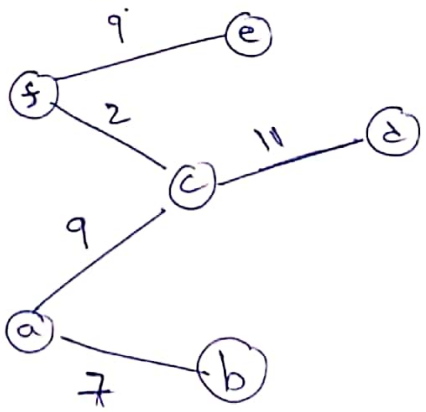
we have if

$$\begin{cases} d(u)+c(u,v) < d(v) \\ \quad d(v) = d(u)+c(u,v) \\ \text{else} \quad d(v) = d(v) \end{cases}$$

	a	b	c	d	e	f	
	0	∞	∞	∞	∞	∞	→ initially
a	0	[7]	9	~~∞~~	∞	14	
b	0		[9]	~~22~~	∞	~~14~~	
c	0			20	∞	[11]	
f				[20]	20	[11]	
d					[20]		
e							

∴ The shortest Path from a is

a - 0
b - 7
c - 9
d - 20
e - 20
f - 11



10) * Degree :-

-> In graphs, the degree of the vertex of a graph is nothing but the number of edges that are incident to the vertex.

-> The degree of a vertex is denoted by Δ.

* In-degree :-

-> The number of incoming edges to a vertex (v) is called In-degree.

* Out-degree :-

-> The no. of out going edges from a vertex is called "out degree".

11) Algorithm :-

-> Start from a random vertex (v) of a graph and perform DFS (G,v)

-> If DFS (G,v) didn't reach other vertex in the graph G, then there exists some vertex u, such that there is a no directed path from v to u. so it is not strongly connected.

-> If the DFS (G,v) reach to every other vertex, then there is a directed path from vertex v to every other vertex in graph G.