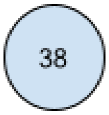
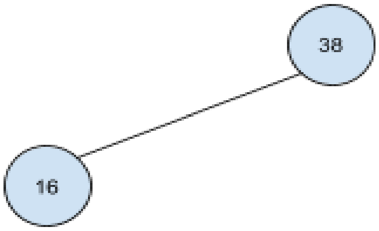
Q1. [6 Marks] Insert 38, 16, 27, 34, 31, 30, 18 one by one in an AVL tree, starting from an empty tree. After inserting all elements delete 38. Draw the tree after every insertion and deletion. Clearly mention which rotations are performed (if applicable) during every insertion and deletion.

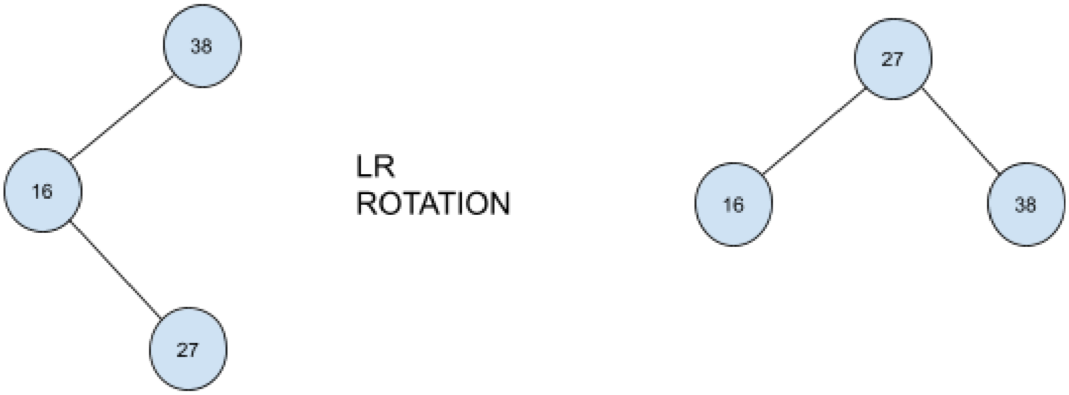
Inserting 38 // 0.25 mark



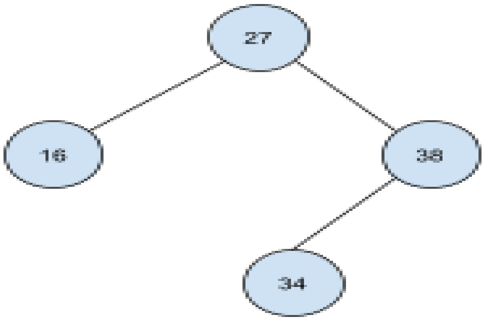
Inserting 16 // 0.25 mark



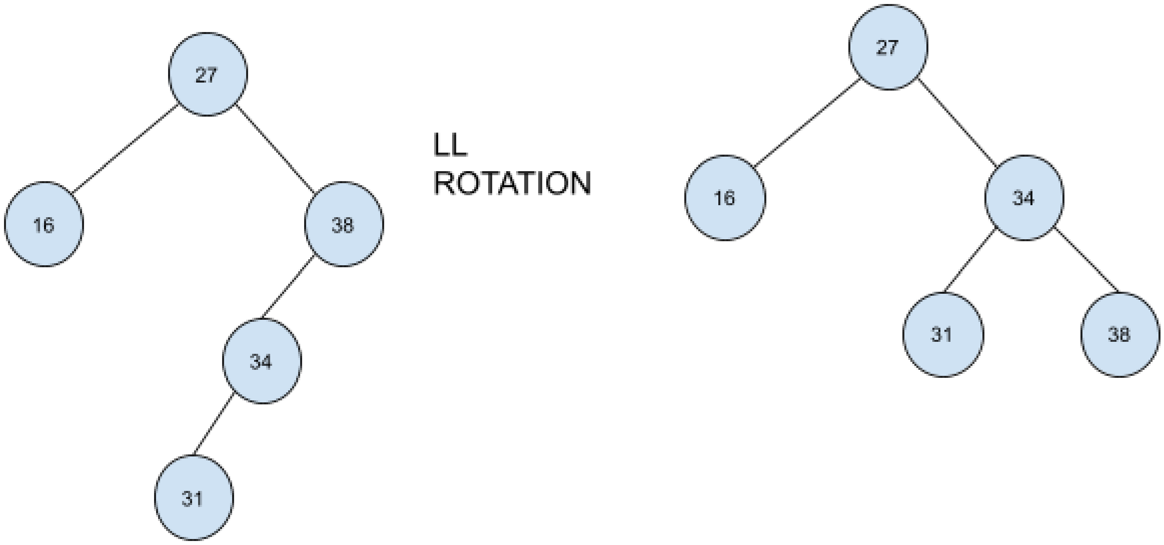
Inserting 27 // 1 Mark if the rotation is correct



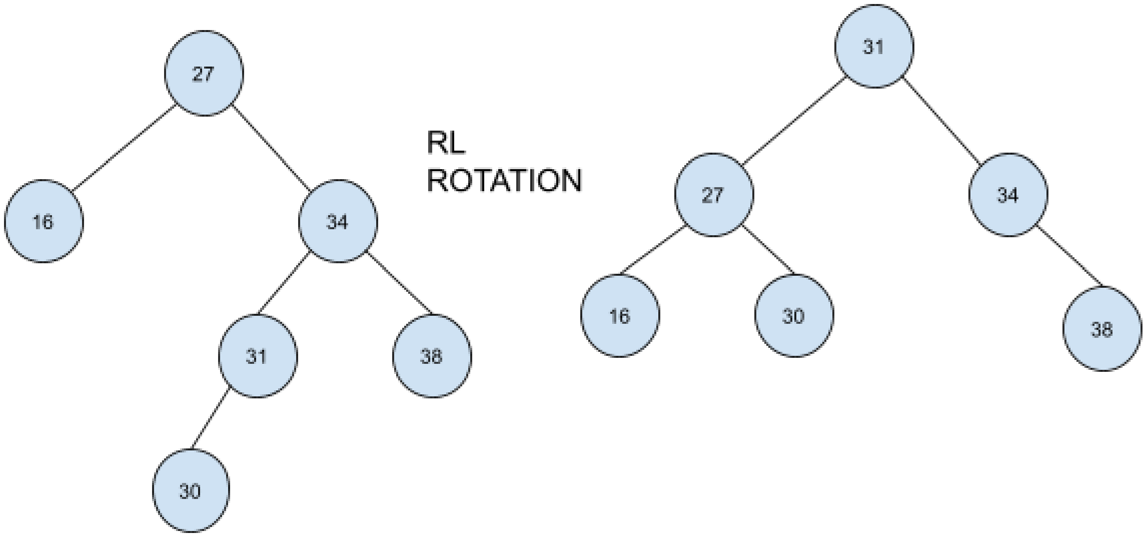
Inserting 34 // 0.25 mark



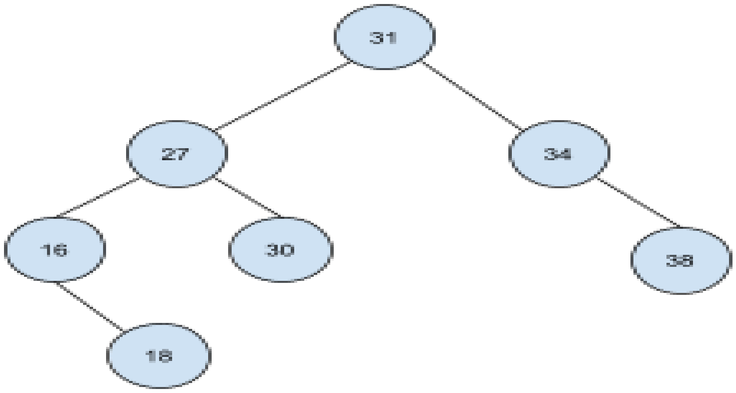
Inserting 31 // 1 Mark if the rotation is correct



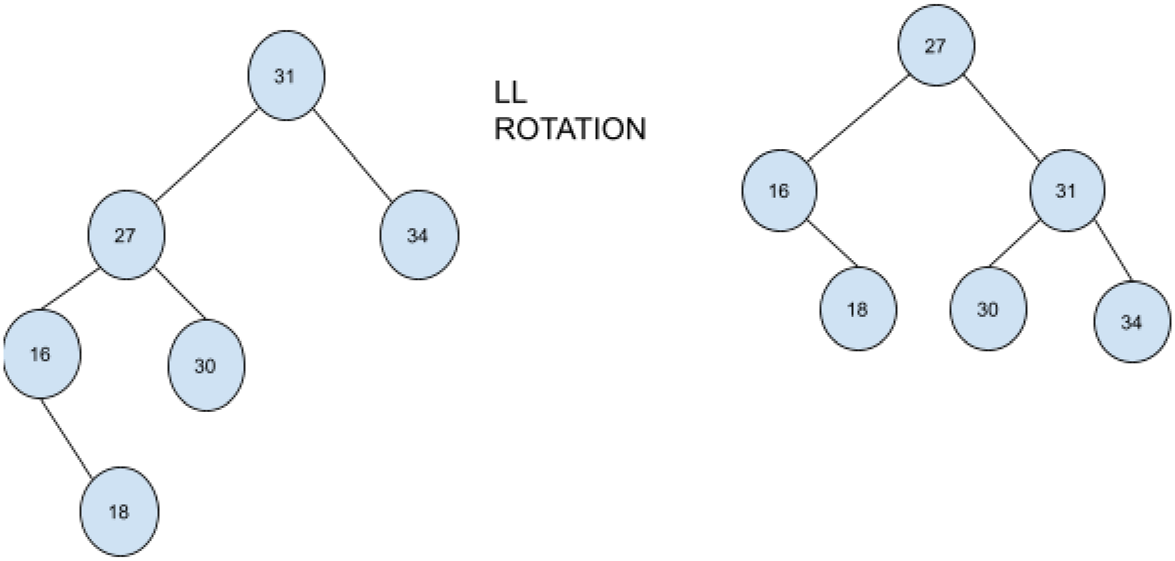
Inserting 30 // 1 Mark if the rotation is correct



Inserting 18 // 0.25 mark



Deleting 38   // 2 Mark if the rotation is correct



Q2. [2+5 Marks] Write an algorithm for DFS that also computes the discovery (or start) and finish time for each vertex. Derive a topologically sorted order using DFS for an acyclic directed graph represented using the following adjacency list. You need to start from vertex zero. Draw the DFS forest with discovery (or start) and finish time for each vertex. Write the corresponding topological sorted order.

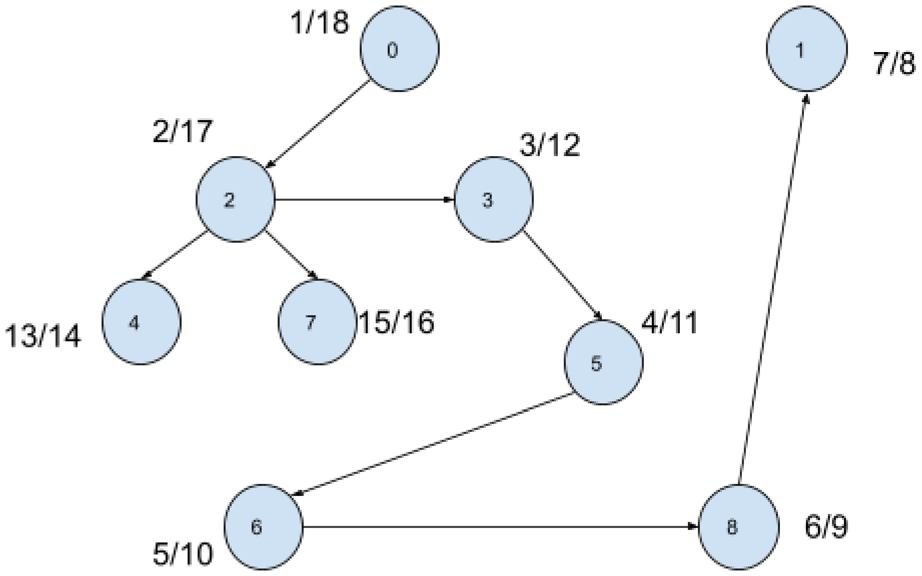
```
DFS(G)
   for each vertex u ∈ G.V
      u.color = WHITE
      u.π = NIL
   time = 0
   for each vertex u ∈ G.V
      if u.color == WHITE
         DFS-VISIT(G, u)

DFS-VISIT(G, u)
   time = time + 1  // update time
   u.d = time       // discovery time
   u.color = GRAY
   for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
         v.π = u
         DFS-VISIT(G, v)
   time = time + 1  // update time
   u.f = time       // finish time
   u.color = BLACK
```

Check for the bold part. If it's present in the algorithm give two marks. Give zero marks if the time, discovery time and finish time is not being updated.

The DFS forest is shown on the next page.

DFS Forest:



Give 0.5 marks for the correct start time, finish time, and the predecessor of a given vertex.
This is the only possible solution. Start time and finish time are separated using a forward slash. For example, 5/10 means 5 is the start (or discovery) time and 10 is the finish time. The edge represents the predecessor.

Topological sorted order: **// Give 0.5 marks for correct topological sorted order**
0 2 7 4 3 5 6 8 1

Q3. [7 Marks] Write an O(1) algorithm to find the third highest element from a max-heap of integers. For example, the third highest number among numbers 1, 2, 3 , 4, 5, 6, 7 is 5.

```
FindThirdMax(H)
// H is a max heap
// arr is the max heap array
// returns the third maximum element

if H->arr[2] > H->arr[3]
  SecondMaxIdx = 2
  ThirdMaxPossIdx = 3
else
  SecondMaxIdx = 3
  ThirdMaxPossIdx = 2

SecondMaxLeft = SecondMaxIdx * 2
SecondMaxRight = SecondMaxLeft + 1

if H->arr[SecondMaxLeft] > H->arr[SecondMaxRight]
  PossibleThirdMax = H->arr[SecondMaxLeft]
else
  PossibleThirdMax = H->arr[SecondMaxRight]

if PossibleThirdMax > H->arr[ThirdMaxPossibleIdx]
  return PossibleThirdMax
else
  return H->arr[ThirdMaxPossibleIdx]
```

No partial marking. Give full marks if the solution is correct.

Q4.Write an algorithm to print all **cross edges** encountered during the DFS of a directed graph.

```
DFS(G)
   for each vertex u ∈ G.V
      u.color = WHITE
      u.π = NIL
   time = 0
   for each vertex u ∈ G.V
      if u.color == WHITE
         DFS-VISIT(G, u)

DFS-VISIT(G, u)
   time = time + 1
   u.d = time
   u.color = GRAY
   for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
         v.π = u
         DFS-VISIT(G, v)
      else if v.color == BLACK and v.d < u.d        // look for this condition
         print “cross edge (u, v)”
   time = time + 1
   u.f = time
   u.color = BLACK
```

Give zero marks if the condition (v.d < u.d) is not present in the solution or the time is not being updated properly, as shown in bold. Give up to 5 marks if the solution is partially correct but v.d < u.d condition is present and the time is being updated in the solution.

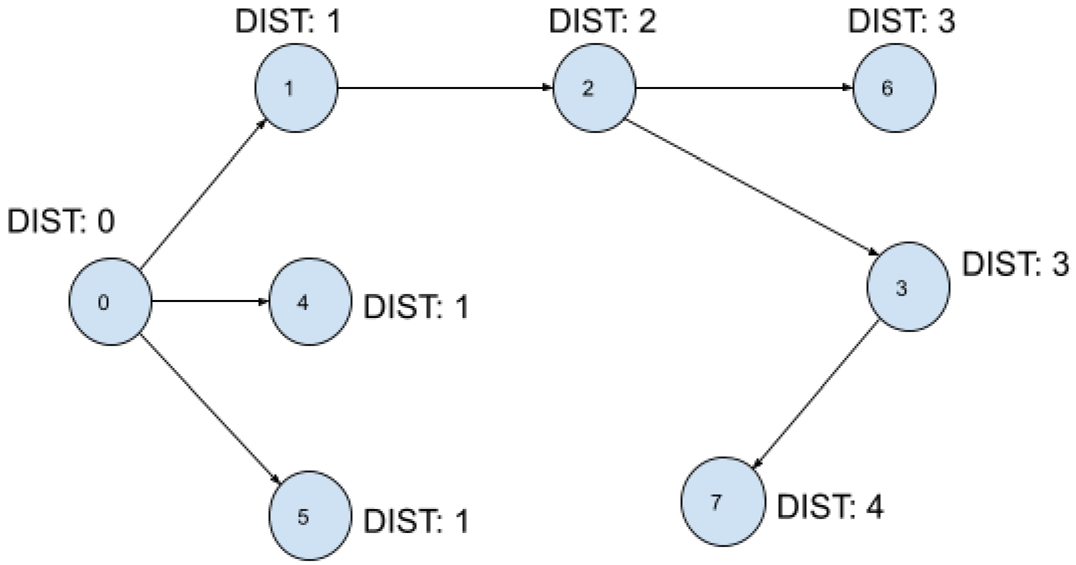
Q5. Write an algorithm for BFS. Compute the shortest distance to each vertex from vertex zero using BFS for an undirected graph represented using the following adjacency list. Show all intermediate enqueue and dequeue operations in the order they take place. Also, draw the final BFS tree with the shortest distance of each vertex.

```
BFS(G, s)
// G(V, E) is the input graph
// s is the source vertex
// each vertex contains three fields, color, d, π
// Output: the shortest distance in d and the predecessor in π

for each vertex u ∈ G.V – {s}
   u.color = WHITE
   u.d = ∞
   u.π = NIL

s.color = GRAY
s.d = 0
s.π = NIL
Q = Φ
ENQUEUE(Q, s)
while Q ≠ Φ
   u = DEQUEUE(Q)
   for each vertex v in G.Adj[u]
      if v.color == WHITE
         v.color = GRAY
         v.d = u.d + 1
         v. π = u
         ENQUEUE(Q, v)
    u.color = BLACK
```

Check if the code marked in bold is present in the pseudocode. The dequeue operation must be outside the if condition inside the loop. The enqueue operation must be inside the if condition inside the loop. The distance must be updated inside the if condition. Give zero in the pseudocode if any of the enqueue, dequeue, and updation of distance are not performed properly. Give two if the pseudocode is correct.



Give one mark if the BFS tree and the distances are correct. Give zero if there is any mistake in the BST tree.

Sequence of enqueue and dequeue operations

enqueue 0
dequeue 0
enqueue 1 enqueue 4 enqueue 5 // in this order
dequeue 1
enqueue 2
dequeue 4
dequeue 5
dequeue 2
enqueue 6 enqueue 3 // in this sequence
dequeue 6
dequeue 3
enqueue 7
dequeue 7

Notice that there is no alternative sequence of enqueue and dequeue operations possible. Give 2 marks if all enqueue and dequeue operations are in the correct order. Give 1 mark if one enqueue/dequeue operation is misplaced. Give zero marks if more than one enqueue and dequeue operations are misplaced.

Q6. Prove that the time complexity of building a max heap from a given array of **n** elements using the bottom-up approach is **O(n)**.

Number of operations required for nodes at level h-1 = 1 * c
Number of operations required for nodes at level h-2 = 2 * c
Number of operations required for nodes at level h-3 = 3 * c
…
Number of operations required for nodes at level 0 = h * c

Number of Nodes at level h-1 = 2^{h-1}
Number of Nodes at level h-2 = 2^{h-2}
Number of Nodes at level h-3 = 2^{h-3}
…
Number of Nodes at level 0 = 2^0
Therefore,

Time complexity:
$T(n) = c * (1 * 2^{h-1} + 2 * 2^{h-2} + 3 * 2^{h-3} + \ldots + h * 2^0)$

// Give 3 marks if the above step is correct
// multiplication with c is optional

$T(n) = c * 2^h ((1 / 2) + (2 / 2^2) + (3 / 2^3) + \ldots + (h / 2^h))$

$= c * 2^h * \sum_{i=1}^{h} (i / 2^i)$

$\sum_{i=1}^{h} (i / 2^i) < 2$ **// derivation of this formula is optional**

$T(n) = c * 2^{h+1}$
$2^h <= n <= 2^{h+1} - 1$
T(n) = O(n) **// Full marks if the entire derivation is correct**

Q7. [7 Marks] Prove that the height of an AVL tree is **O(log n)**.

Let’s say f(h) represents the minimum number of node in an AVL tree of height h

$f(0) = 1$, $f(1) = 2$

$f(h) = f(h-1) + f(h-2) + 1$ **// Give total two marks if this equation is correct**

$f(h) > 2 * f(h-2)$ **// Give total three marks if the solution up to this is correct**

$> 2^2 * f(h-4)$

$> 2^3 * f(h-6)$

…

$> 2^k * f(h-2k)$ **// Give total five marks if solution up to this is correct**

if h is even substitute $h = 2k$

$f(h) > 2^{h/2} * f(0)$

$> 2^{h/2}$

$h < 2 * \log_2 (f(h))$

$< 2 * \log_2 (n)$, because $f(h) <= n$

$= O(\log n)$ **// Give full marks if the solution up to this is correct**

if h is odd substitute $h = 2k + 1$

$f(h) > 2^{(h-1)/2} * f(1)$

$> 2^{(h-1)/2} * 2$

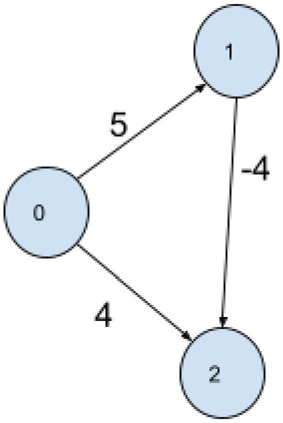
$> 2^{(h+1)/2}$

$h < (2 * \log_2 (f(h))) - 1$

$< (2 * \log_2 (n)) - 1$, because $f(h) <= n$

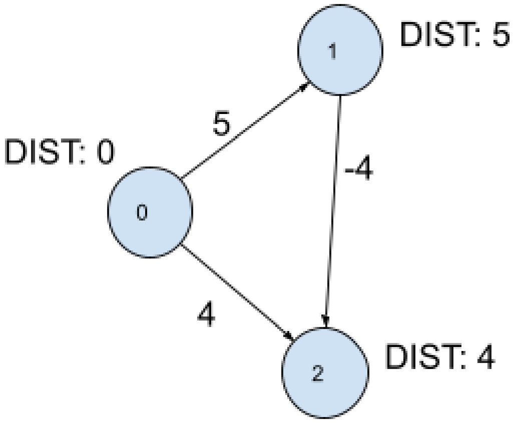
$= O(\log n)$ **// Give full marks if any of the two cases, odd or even, are handled**

Q8. [4 Marks] Give an example to demonstrate that the Dijkstra algorithm for single-source shortest-paths doesn't work for an acyclic directed graph that may also contain negative weight edges.



ORIGINAL GRAPH

SHORTEST PATH COMPUTED VIA DIJKSTRA FROM VERTEX 0



The Dijkstra algorithm works as follows. First, it sets the shortest distance to the source vertex, 0 in this example, to zero and the shortest distance to other vertices as infinity. The distance to vertex zero is final and doesn't change after this step.
In the first iteration, it updates the distances of the adjacent vertices of the source vertex. It then picks a vertex, v, whose distance is minimum among all the vertices whose distance is not final. It then marks the distance of v as final.
It then updates the distances of the adjacent vertices of v, picks a vertex with the minimum distance, and marks it as final. It repeats the same steps until the distances of all vertices are finalized.

If we run this algorithm, in the example shown above, the distance of vertex 2 will be computed as 4, which is wrong because the distance to vertex 2 is

1 if we reach 2 from 0 via 1.

There is no partial marking. Give full marks if the solution is completely correct.

Q9. [6 Marks] Insert the following keys one by one in a hash table using the double hashing scheme starting from an empty hash table. For each insertion, write the number probes that are required and the index at which the element is inserted in the hash table. For example, the number of probes required to insert 7 is zero. Both the functions used for the double hashing are the same and defined as follows. The size of the hash table is 13.

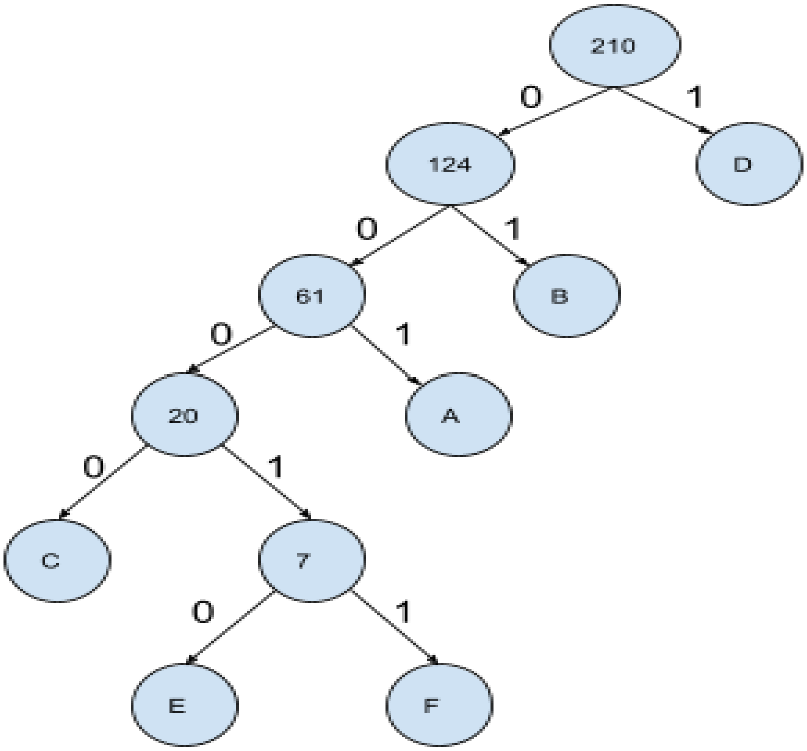
```
int HashFunction(int key) {
        return key % 13;
}
```

Keys: **7 33 8 14 46 27 21**

Keys	Index	Number probes
7	7	0
33	1	1
8	8	0
14	2	1
46	9	4
27	3	2
21	11	2

Deduct one mark for each wrong number of probes.
Deduct two marks for each wrong index.
Give zero if the total marks gets negative using the above formula.

Q10. [4 Marks] Generate the Huffman codes for the characters **a, b, c, d, e, f**. The frequency of their occurrences is given in the table below. Draw the tree and mark each character with its corresponding encoding.



Character	Frequency	Encoding
A	41	001
B	63	01
C	13	0000
D	86	1
E	5	00010
F	2	00011

The tree in the answer could be different but the following constraints must be satisfied.

The number of bits in the encoding of A must be 3, and the number of edges in the path from root to A must be 3.

The number of bits in the encoding of B must be 2, and the number of edges in the path from root to B must be 2.

The number of bits in the encoding of C must be 4, and the number of edges in the path from root to C must be 4.

The number of bits in the encoding of D must be 1, and the number of edges in the path from root to D must be 1.

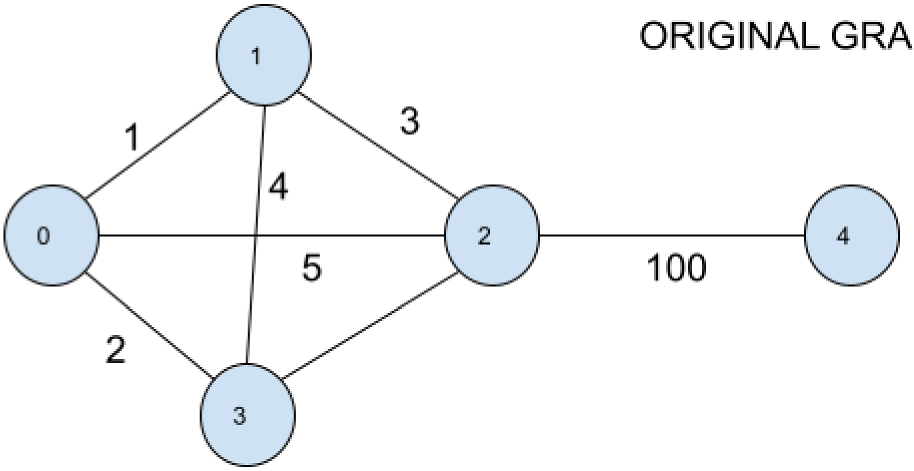
The number of bits in the encoding of E must be 5, and the number of edges in the path from root to E must be 5.

The number of bits in the encoding of F must be 5, and the number of edges in the path from root to F must be 5.

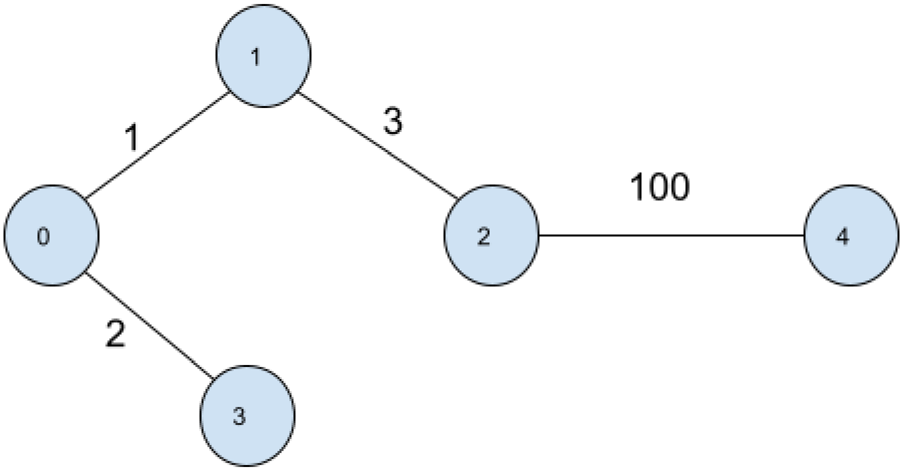
Deduct one mark if the constraints for a character are not satisfied.
If the total marks is negative make it zero.

Q11. [5 Marks] Give an example to demonstrate that the minimum spanning tree of a connected undirected graph with more than **n+1** edges, where **n** is the number of vertices, may contain the edge with the maximum weight. Each edge has a distinct weight. Show the graph and the corresponding minimum spanning tree.

ORIGINAL GRAPH



CORRESPONDING MST



Notice that the edge with the maximum weight must not be the part of a cycle. Give full marks if the number of edges are at least n+2 in the solution, where n is the number of vertices, and the maximum weight edge is the part of the MST.

If the number of edges is n+1 and the solution is correct, give three marks.

Give zero marks if the number of edges is less than n+1.