Tree

Trees are abstract data structures, used to manage data in a hierarchical way, making data retrieving much more efficient than other data structure methods. A tree is a collection of **nodes** that are linearly connected and don't have any cyclic relations as shown in Figure 1.

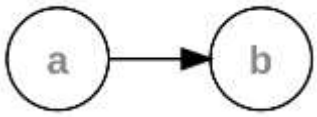
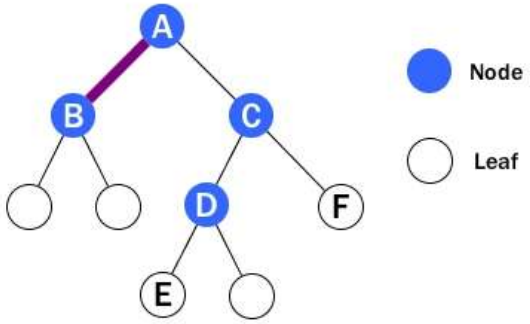


Figure 1: Basic Definition of a Tree

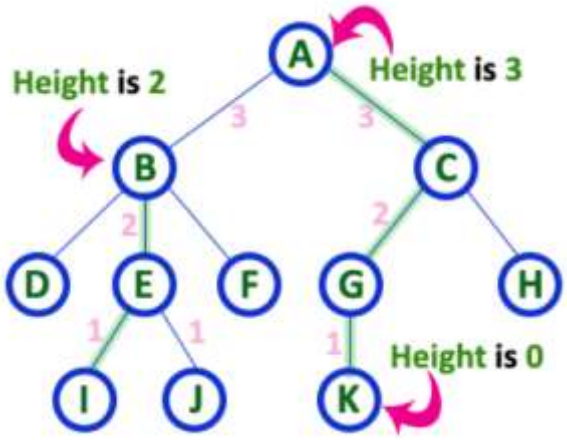
Tree Terminology

- A node is an entity that contains a key or value and pointers to its child nodes.
- Edge – connection between one node to another.

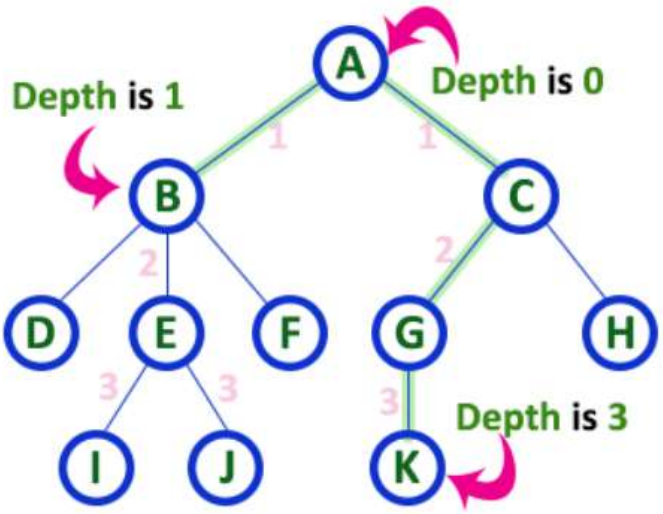


About Edge

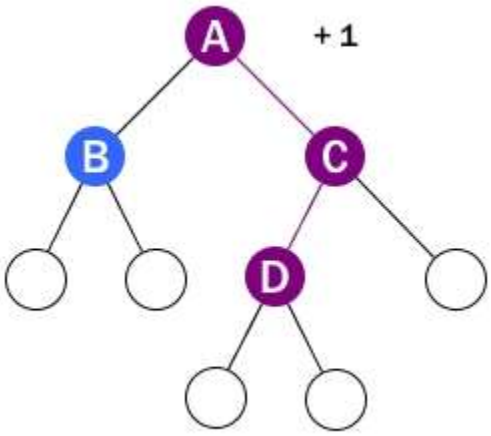
- Root is the topmost node of a tree.
- Path – a sequence of nodes and edges connecting a node with a descendant.
- Height of a node – The height of a node is the number of edges on the longest downward path between that node and a leaf. In a tree, height of all leaf nodes is '0'.



- Depth of a node –The depth of a node is the number of edges from the node to the tree's root node. In a tree, depth of the root node is '0'.

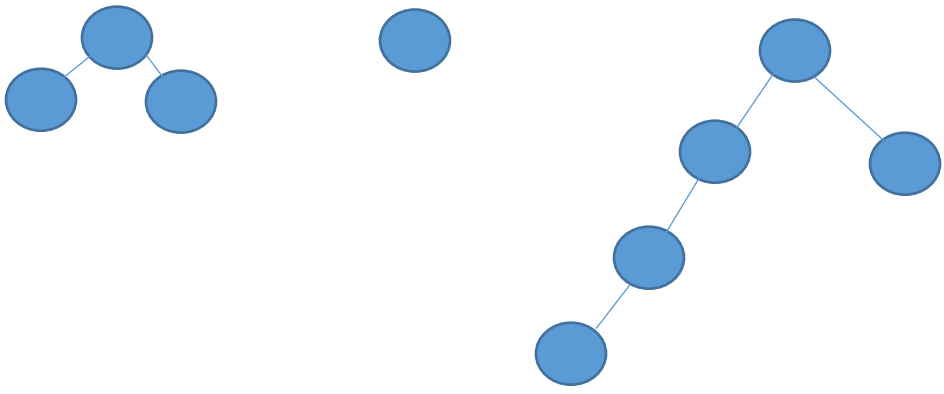


- Level – The level of a node is defined by 1 + the number of connections between the node and the root. Simply, Level = Depth + 1
  The important thing to remember is when talking about level, it starts from 1 and the level of the root is 1.



Binary Tree

A tree whose elements have at most 2 children (0, 1, 2 but not more than 2) is called a binary tree.



If there are k levels in a binary tree, then the maximum number of nodes in the tree is as follows:

$n = 2^k - 1$

For example, if k = 3, then the maximum number of nodes (n) is 7 and for k = 4, n = 15 and so on.

When the maximum number of nodes is known, number of levels can be calculated as-

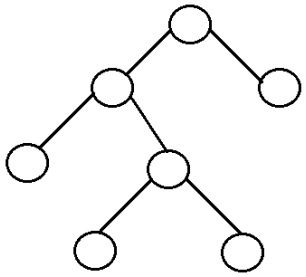
$k = \lceil \log_2 (n + 1) \rceil$ // ⌈ x ⌉ means ceiling of x to the next integer

For example,

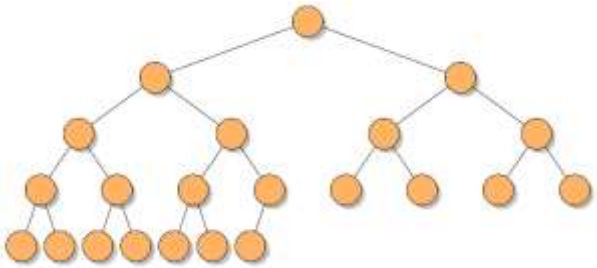
when n = 17; $k = \lceil \log_2 (17 + 1) \rceil = 5$

when n = 34; $k = \lceil \log_2 (34 + 1) \rceil = 6$; etc.

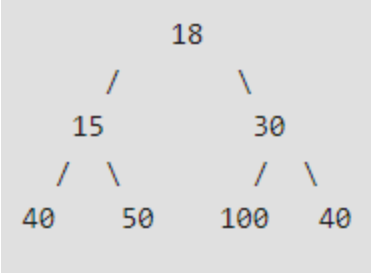
Full / Proper / Strict binary tree: It is a tree in which every node in the tree has either 0 or 2 children.



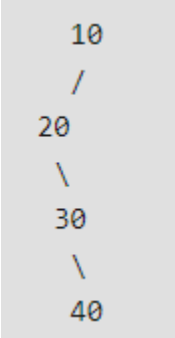
In a **complete binary tree** all levels except the last are completely filled, and in the last level all nodes are to the left as much as possible. This means that all nodes have two children except the nodes at the lowest two levels. At the lowest level the nodes have (by definition) zero children, and at the level above that nodes can have 0, 1 or 2 children.



Perfect Binary Tree: A Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at the same level.



A degenerate (or pathological) tree: A Tree where every internal node has one child.



Traversing Binary Trees

Traversing means to visit all the nodes of the tree. There are three standard methods to traverse the binary trees. These are as follows:

- Preorder Traversal
- Postorder Traversal
- Inorder Traversal

Preorder Traversal: The preorder traversal of a tree is

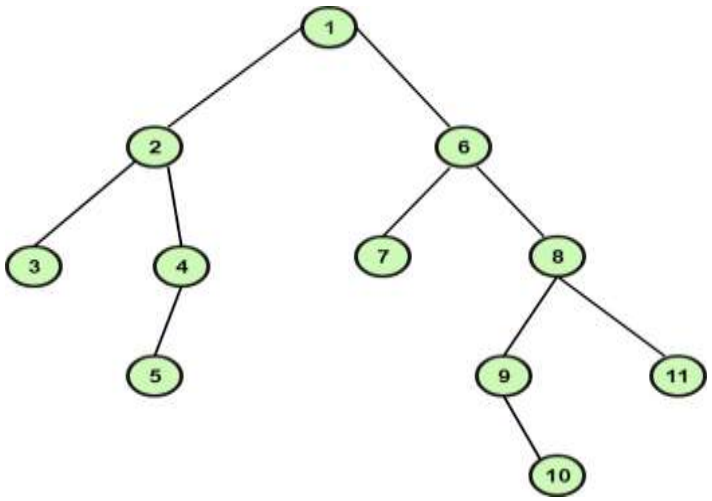
1. Visit the root of the tree.
2. Traverse the left subtree in preorder.
3. Traverse the right subtree in preorder.

Postorder Traversal: The postorder traversal of a tree is

1. Traverse the left subtree in postorder.
2. Traverse the right subtree in postorder.
3. Visit the root of the tree.

Inorder Traversal: The inorder traversal of a tree is

1. Traverse in inorder the left subtree.
2. Visit the root of the tree.
3. Traverse in inorder the right subtree.



Preorder (Root->Left->Right)	1	2	3	4	5	6	7	8	9	10	11
Postorder (Left->Right->Root)	3	5	4	2	7	10	9	11	8	6	1
Inorder (Left->Root->Right)	3	2	5	4	1	7	6	9	10	8	11

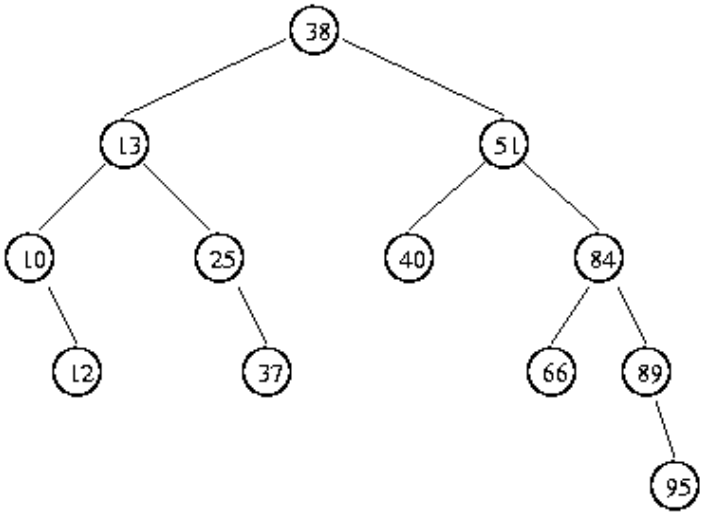
Binary Search Tree (BST)

BST is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

Construct a Binary Search Tree (BST) by inserting the following sequence of numbers:

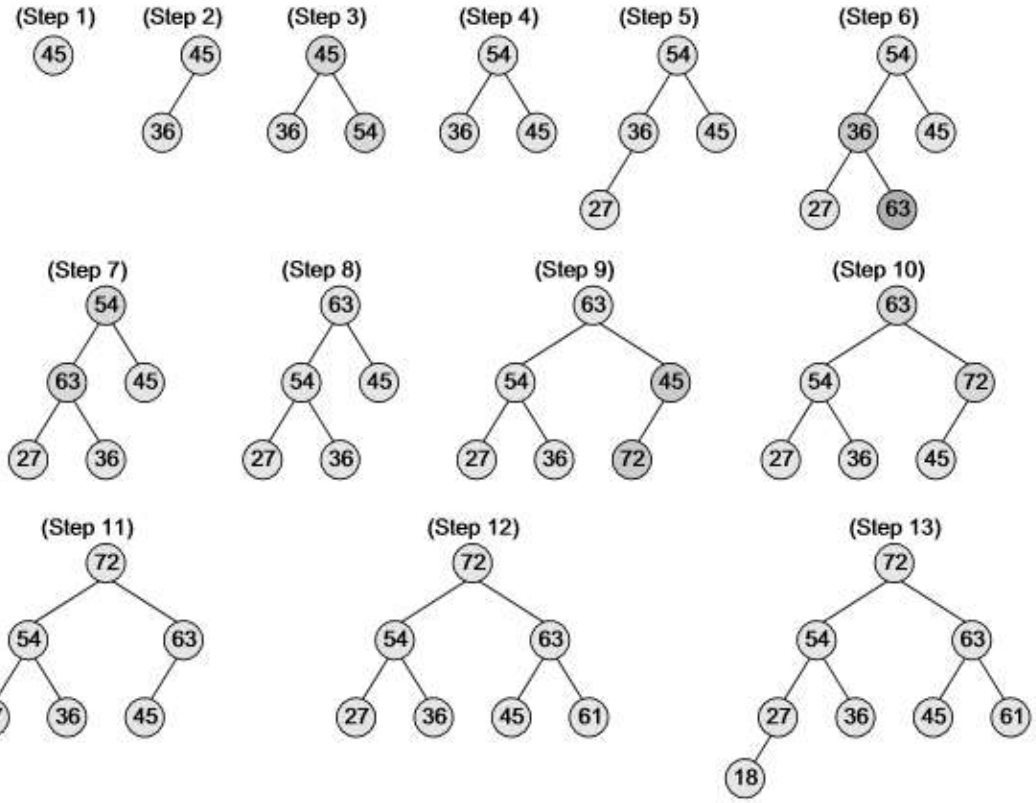
38, 13, 51, 10, 12, 40, 84, 25, 89, 37, 66, 95



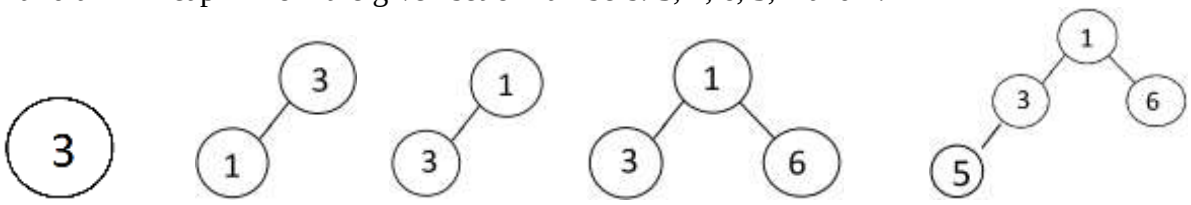
Heap

It is a complete binary tree each of whose node's value is greater (or smaller) than the node value of its children. If the node value is greater than the node value of its children, then the heap is called max heap. Otherwise, the heap is called min heap.

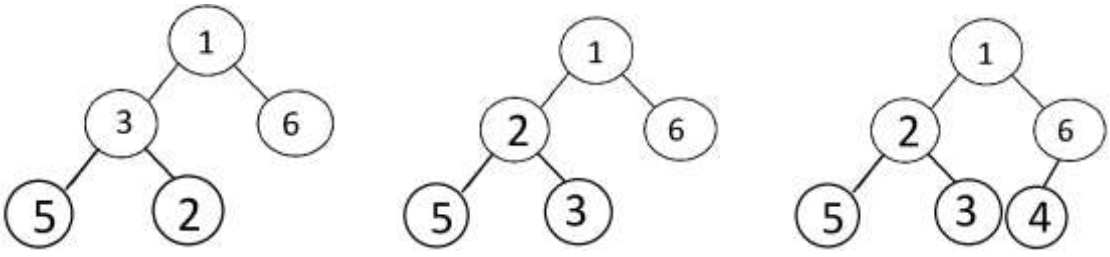
Build a max heap H from the given set of numbers: **45, 36, 54, 27, 63, 72, 61and 18**.



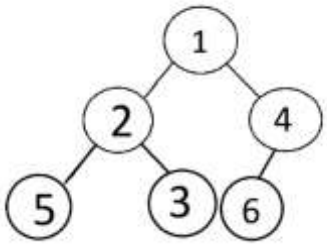
Build a min heap H from the given set of numbers: **3**, **1**, **6**, **5**, **2** and **4**.



Step-01 Step-02 Step-03 Step-04 Step-05



Step-06 Step-07 Step-08



Step-09

Huffman Coding

Huffman Coding is a technique of compressing data so as to reduce its size without losing any of the details. It was first developed by David Huffman. Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.

Algorithm for creating the Huffman Tree-

Step 1- Create a leaf node for each character and build a min heap using all the nodes (The frequency value is used to compare two nodes in min heap)

Step 2- Repeat Steps 3 to 5 while heap has more than one node

Step 3- Extract two nodes, say x and y, with minimum frequency from the heap

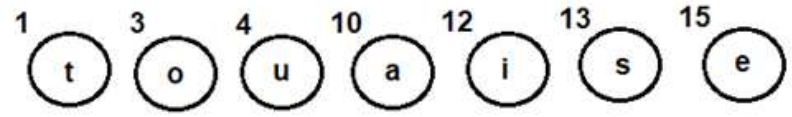
Step 4- Create a new internal node z with x as its left child and y as its right child. Also frequency(z)= frequency(x)+frequency(y)

Step 5- Add z to min heap

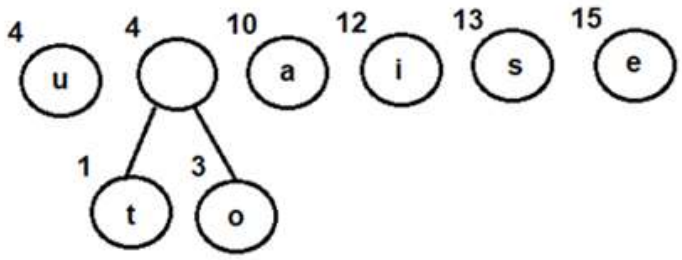
Step 6- Last node in the heap is the root of Huffman tree

Let's try and create Huffman Tree for the following characters along with their frequencies using the above algorithm-

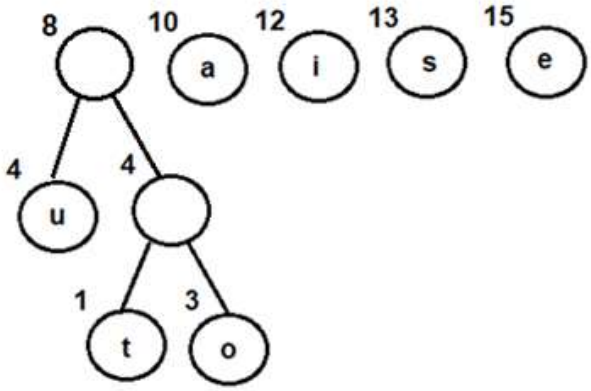
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1



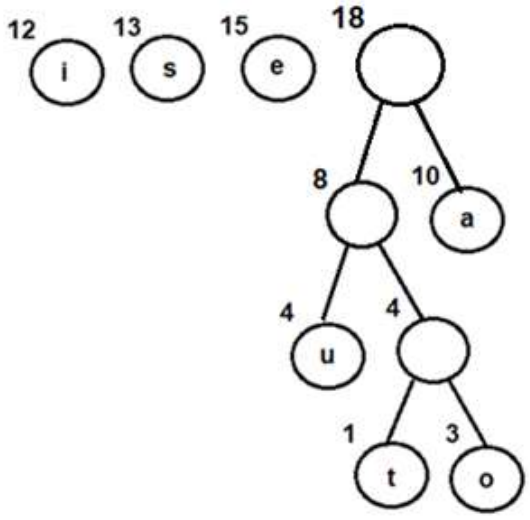
Step-01



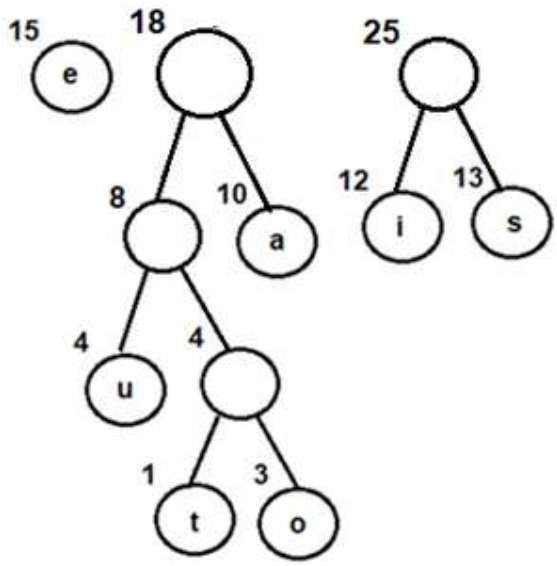
Step-02



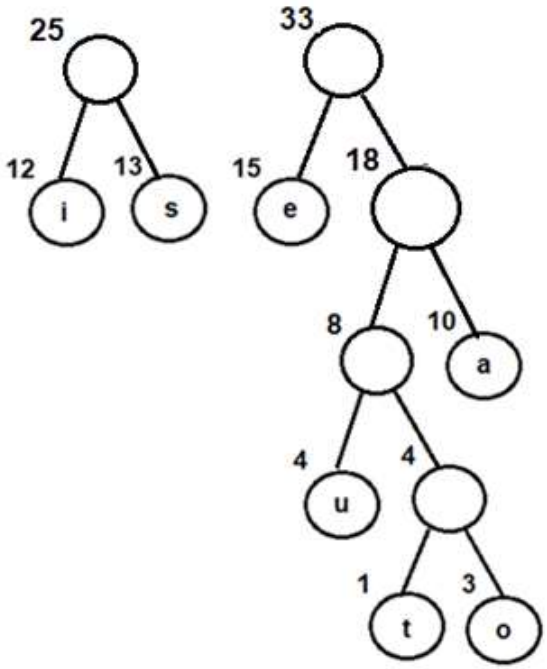
Step-03



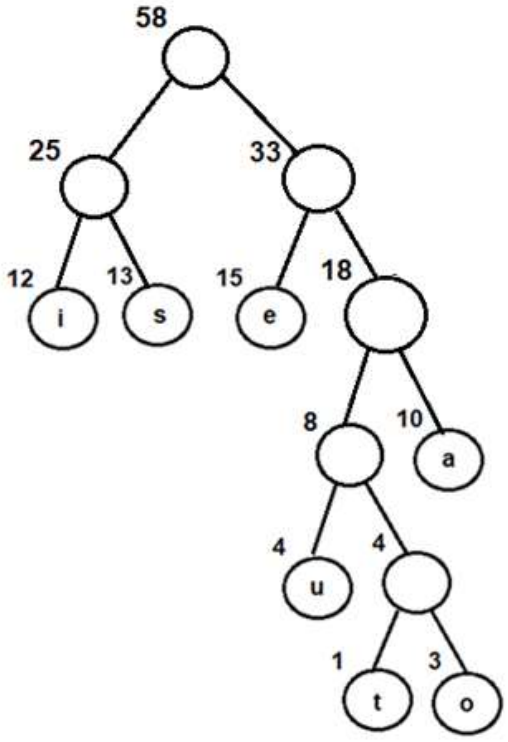
Step-04



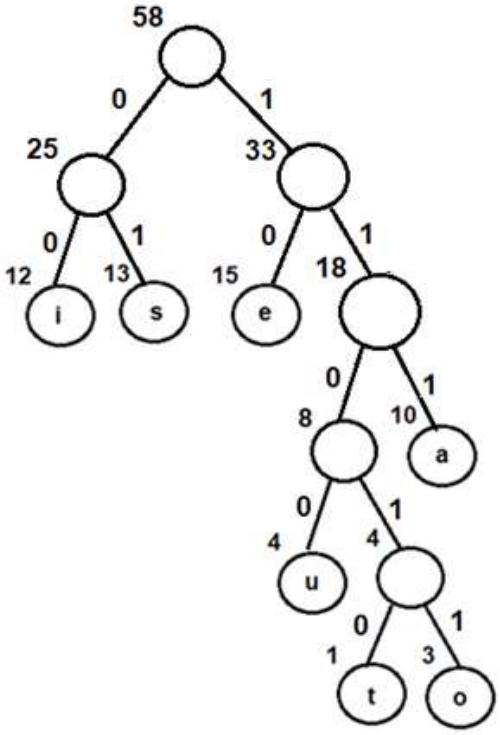
Step-05



Step-06



Step-07



Step-08

Characters	Binary Codes
i	00
s	01
e	10
u	1100
t	11010
o	11011
a	111