

## Group A

### Assignment No:

**2**

**Title of the Assignment:** Write a program to implement Huffman Encoding using a greedy strategy.

**Objective of the Assignment:** Students should be able to understand and solve Huffman Encoding using greedy method

**Prerequisite:**

1. Basic of Python or Java Programming
  2. Concept of Greedy method
  3. Huffman Encoding concept
- 

**Contents for Theory:**

1. Greedy Method
  2. Huffman Encoding
  3. Example solved using huffman encoding
-



## What is a Greedy Method?

- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.
- The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.
- This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

## Advantages of Greedy Approach

- The algorithm is **easier to describe**.
- This algorithm can **perform better** than other algorithms (but, not in all cases).

## Drawback of Greedy Approach

- As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm
- For example, suppose we want to find the longest path in the graph below from root to leaf.

## Greedy Algorithm

1. To begin with, the solution set (containing answers) is empty.
2. At each step, an item is added to the solution set until a solution is reached.
3. If the solution set is feasible, the current item is kept.
4. Else, the item is rejected and never considered again.

## Huffman Encoding

- Huffman Coding is a technique of compressing data 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.

- Huffman Coding is a famous Greedy Algorithm.
- It is used for the lossless compression of data.
- It uses variable length encoding.
- It assigns variable length code to all the characters.
- The code length of a character depends on how frequently it occurs in the given text.
- The character which occurs most frequently gets the smallest code.
- The character which occurs least frequently gets the largest code.
- It is also known as **Huffman Encoding**.

### Prefix Rule-

- Huffman Coding implements a rule known as a prefix rule.
- This is to prevent the ambiguities while decoding.
- It ensures that the code assigned to any character is not a prefix of the code assigned to any other character

### Major Steps in Huffman Coding-

There are two major steps in Huffman Coding-

1. Building a Huffman Tree from the input characters.
2. Assigning code to the characters by traversing the Huffman Tree.

### How does Huffman Coding work?

Suppose the string below is to be sent over a network.



- Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of  $8 * 15 = 120$  bits are required to send this string.
- Using the Huffman Coding technique, we can compress the string to a smaller size.

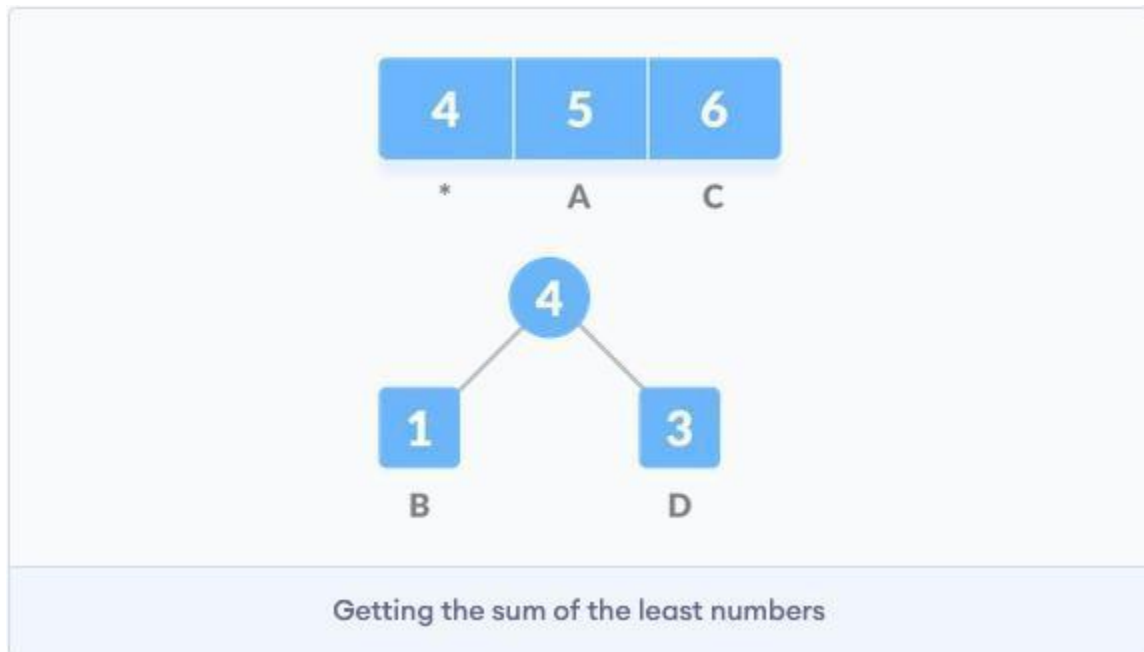
- Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.
  - Once the data is encoded, it has to be decoded. Decoding is done using the same tree.
  - Huffman Coding prevents any ambiguity in the decoding process using the concept of **prefix code** ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.
- Huffman coding is done with the help of the following steps.
1. Calculate the frequency of each character in the string.



2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.



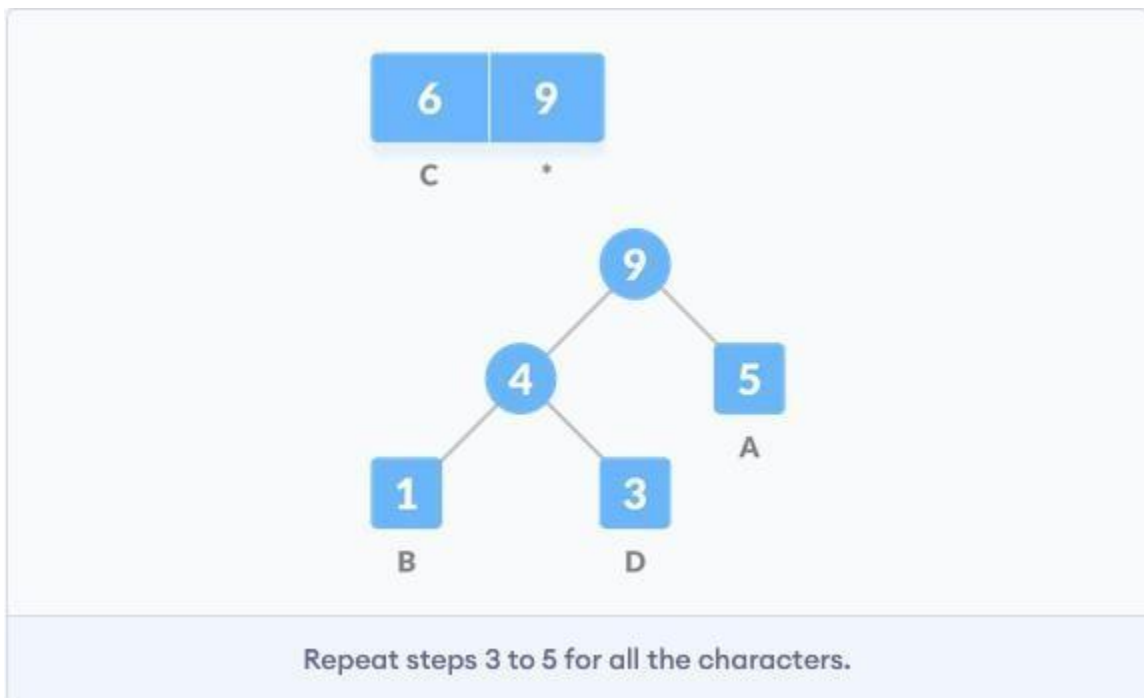
3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

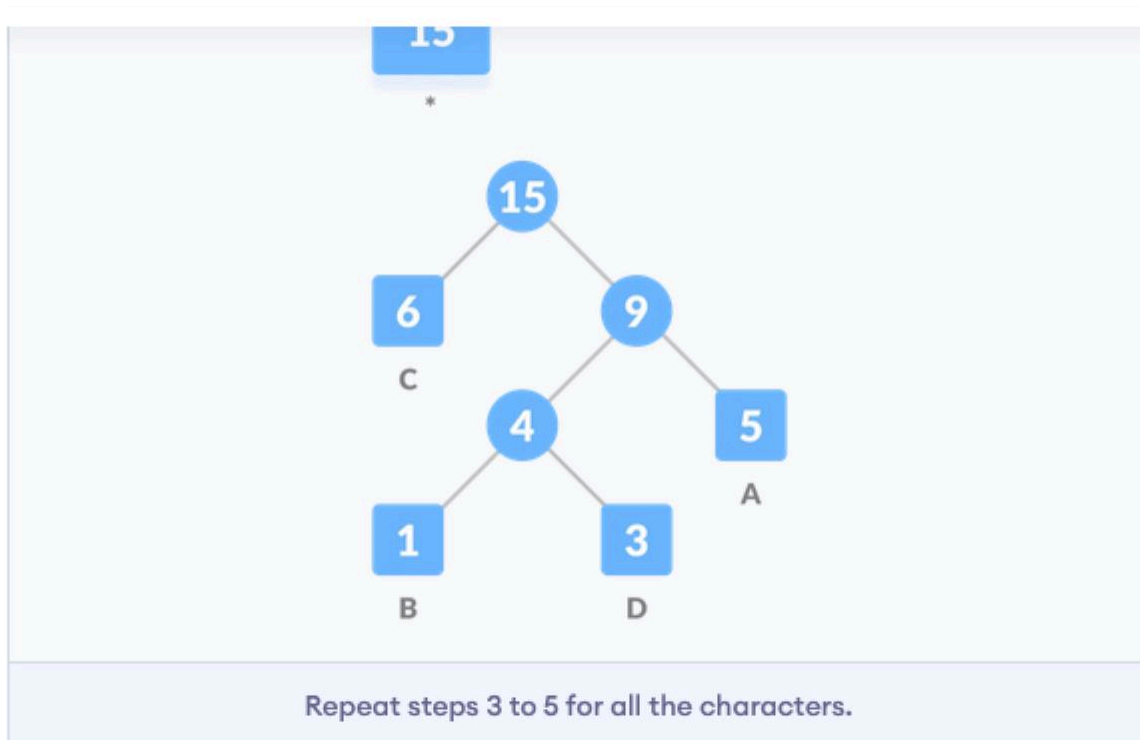


5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (\* denote the internal nodes in the figure above).

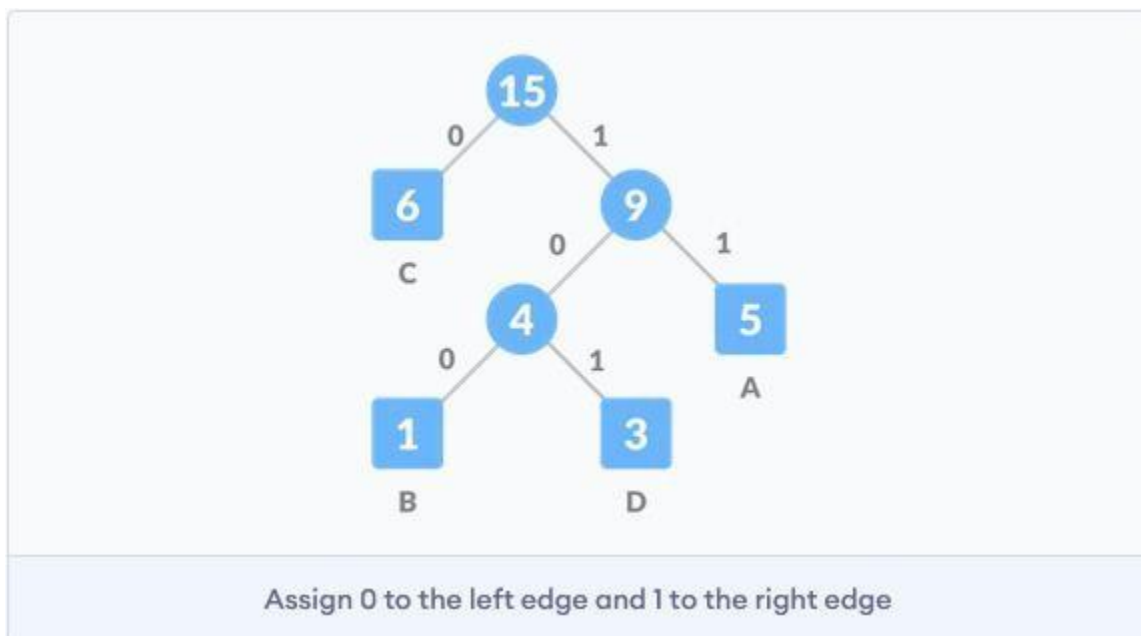
6. Insert node z into the tree.

7. Repeat steps 3 to 5 for all the characters.





8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge



For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

Character	Frequency	Code	Size
A	5	11	$5 \times 2 = 10$
B	1	100	$1 \times 3 = 3$
C	6	0	$6 \times 1 = 6$
D	3	101	$3 \times 3 = 9$
$4 \times 8 = 32 \text{ bits}$	15 bits		28 bits

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to  $32 + 15 + 28 = 75$ .

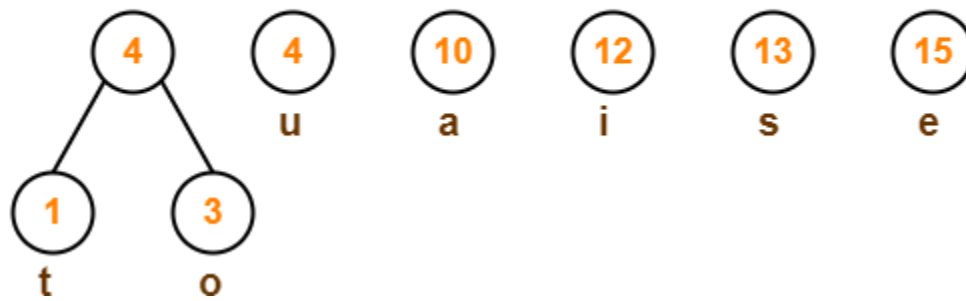
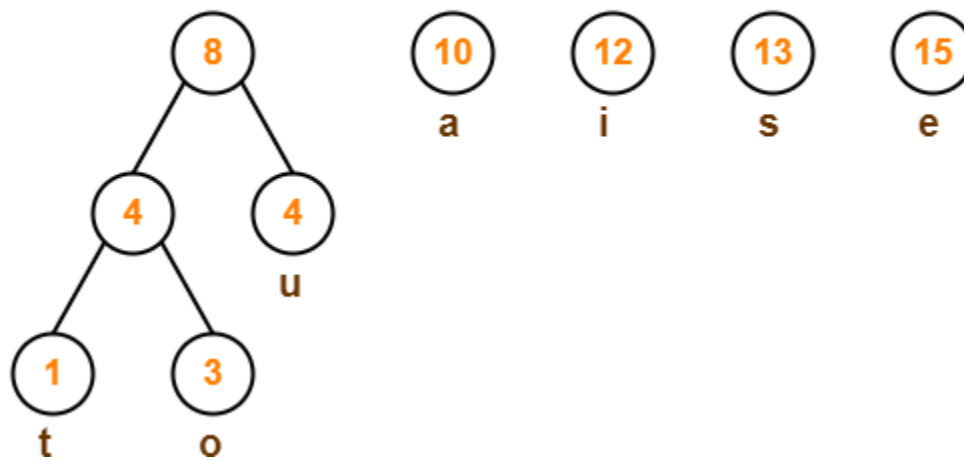
### Example:

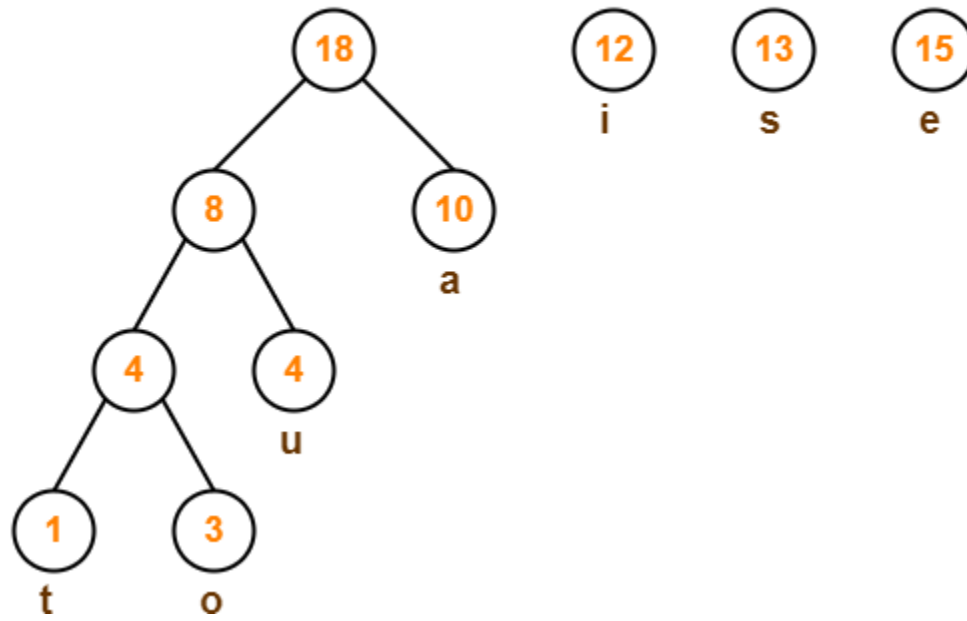
A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

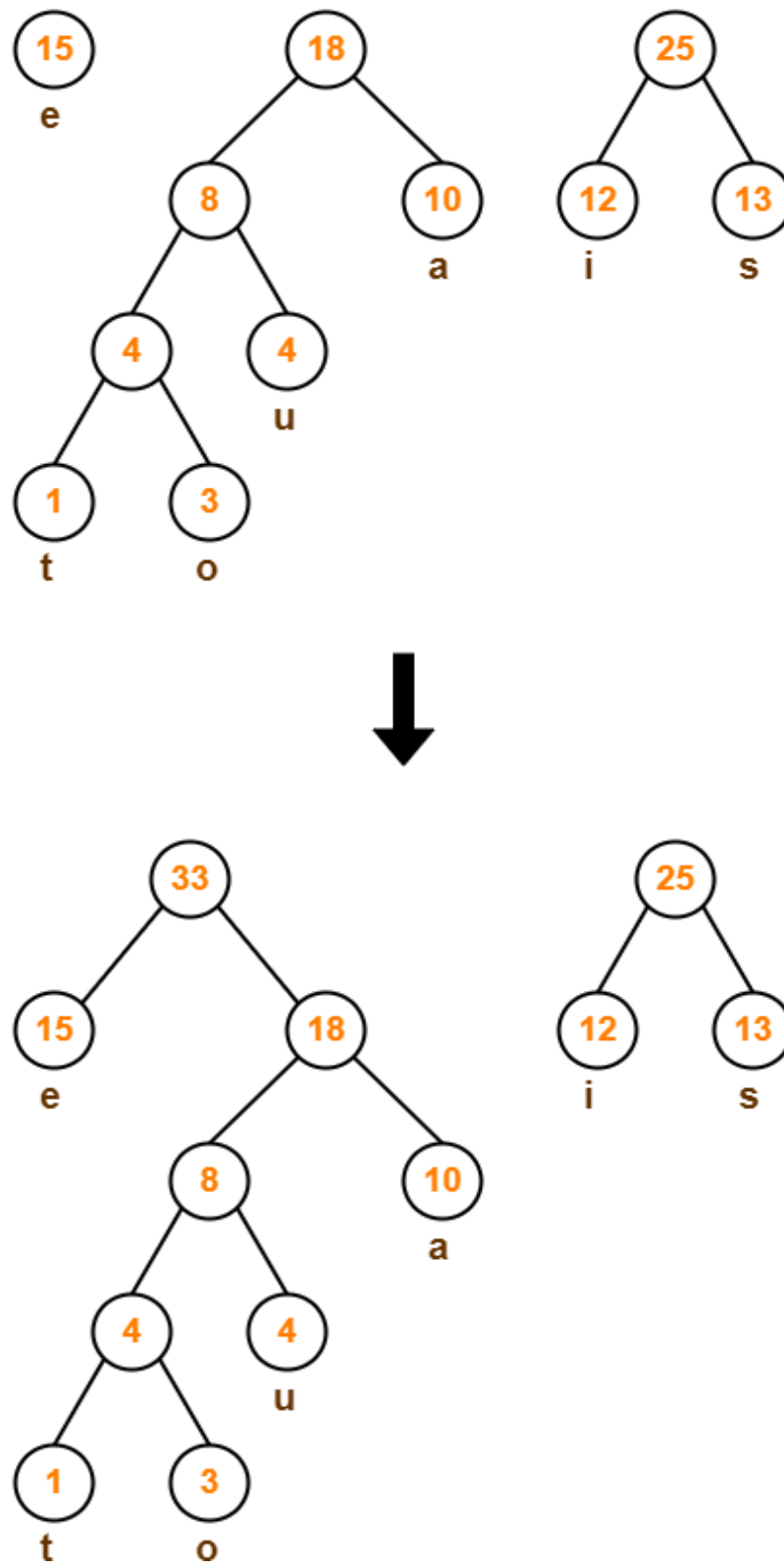
1. Huffman Code for each character
2. Average code length
3. Length of Huffman encoded message (in bits)

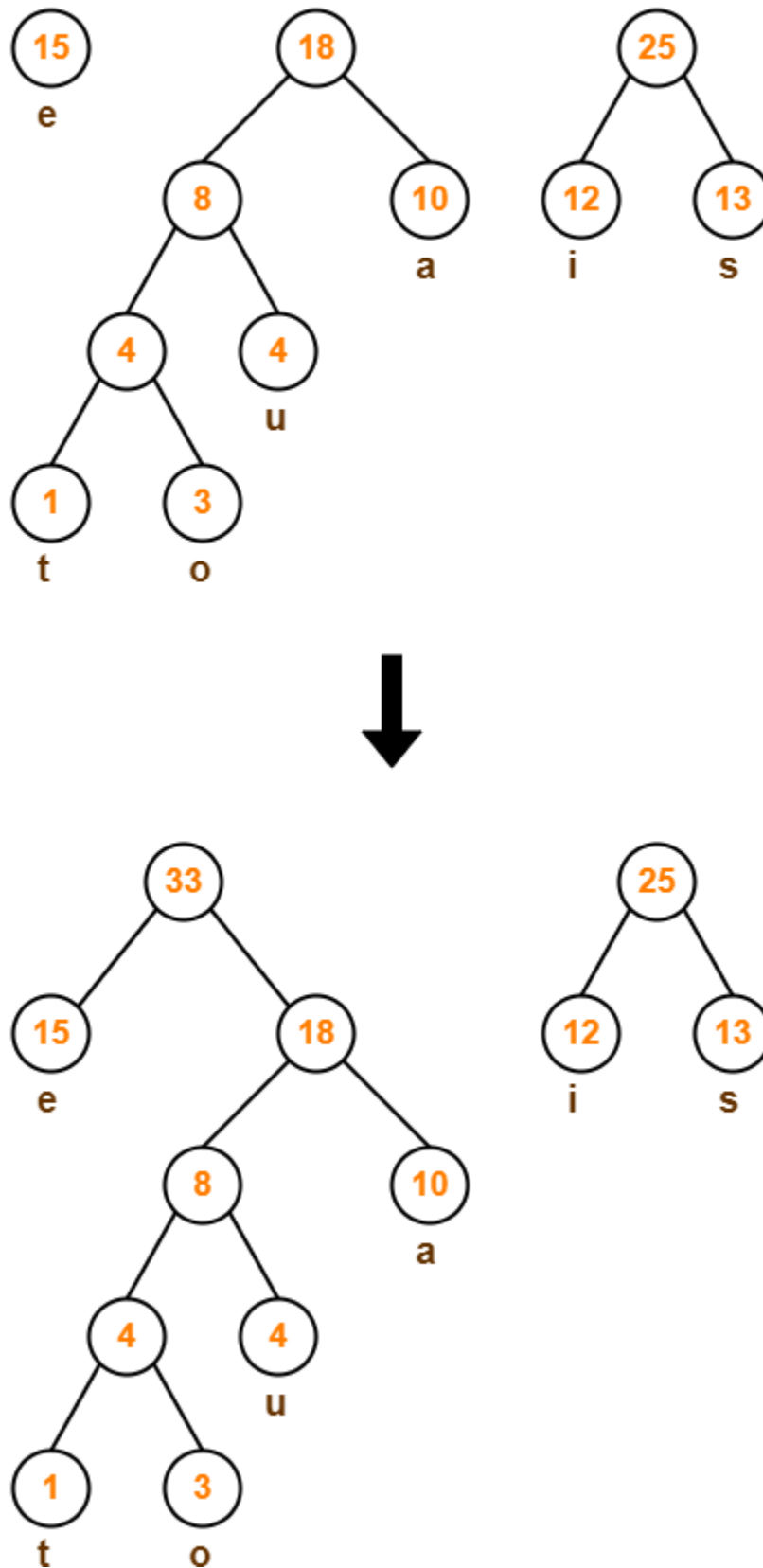
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

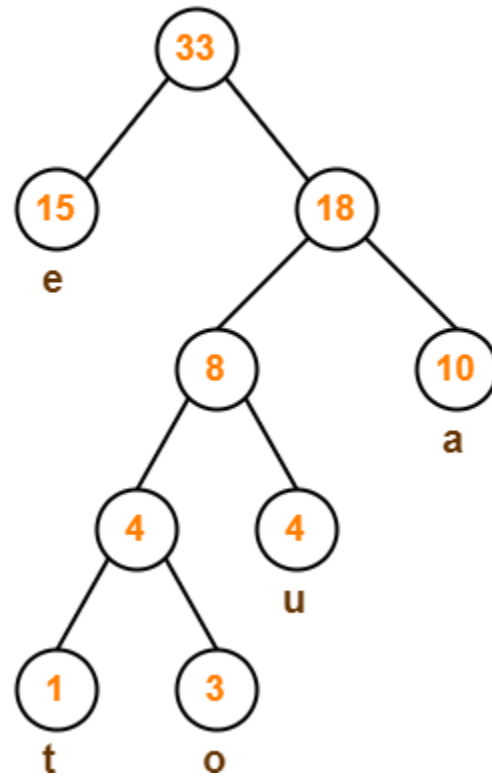
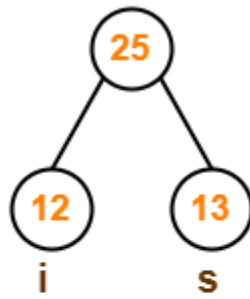


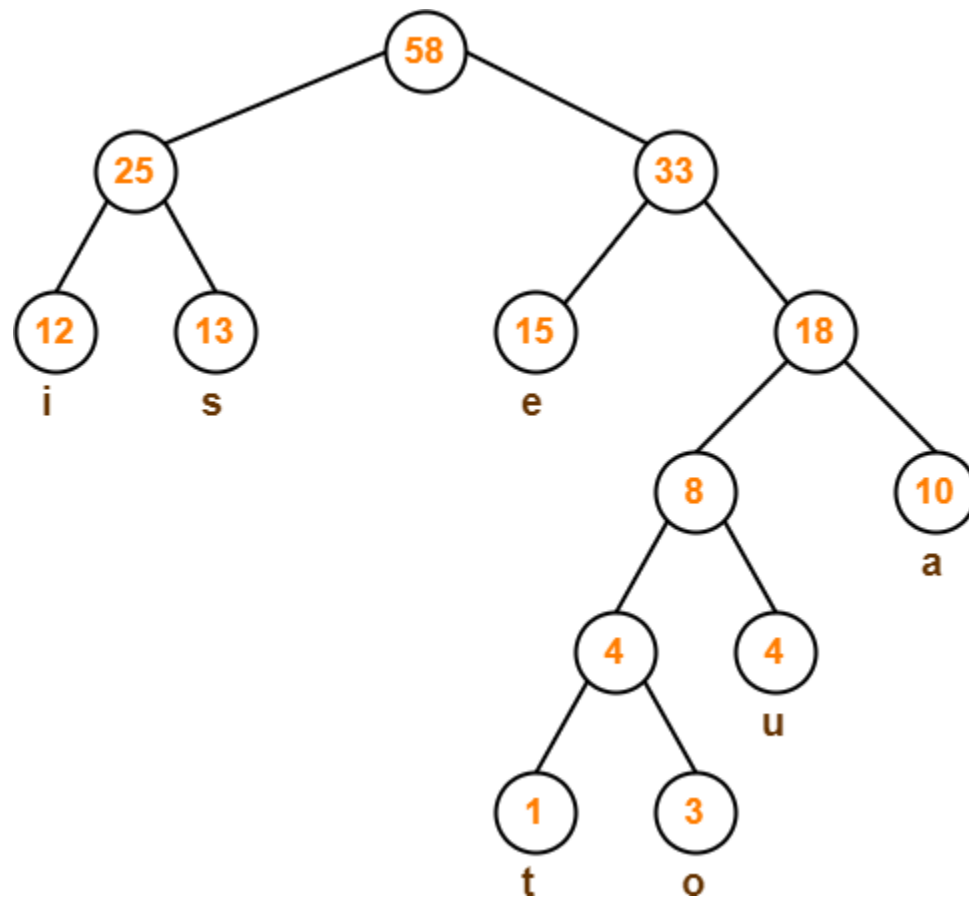
**Step-01:****Step-02:****Step-03:**

**Step-04:**

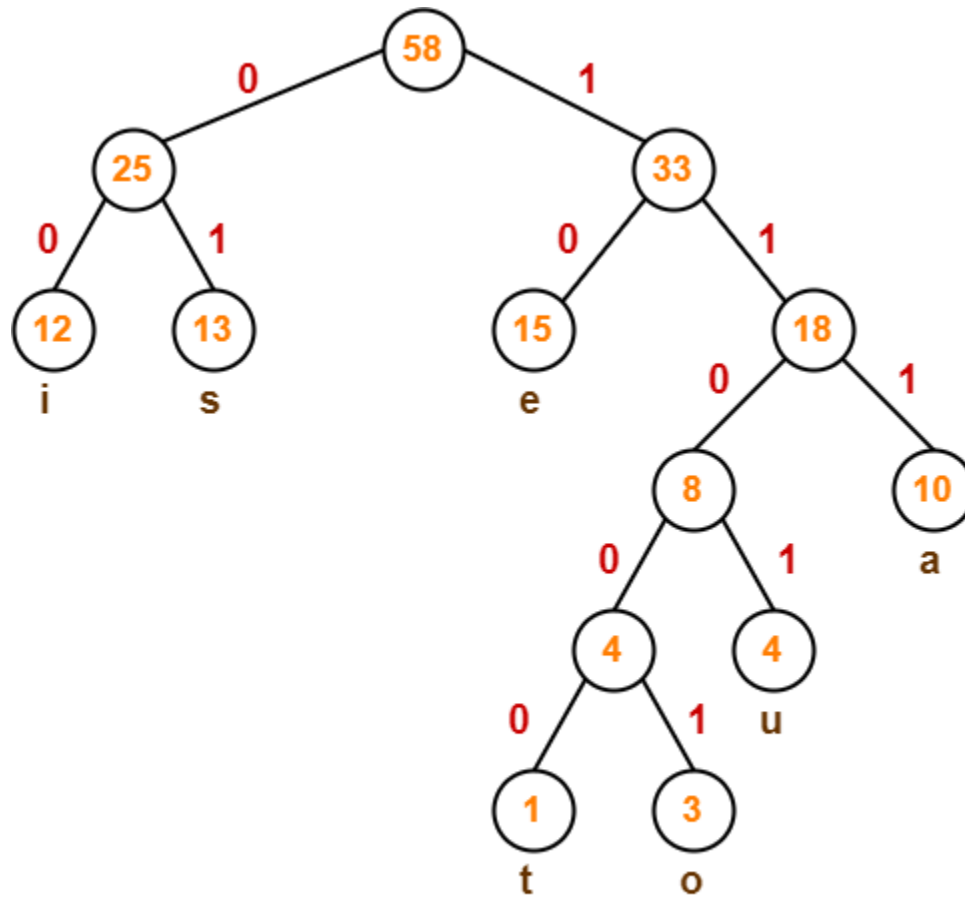
**Step-06:**

**Step-06:**

**Step-07:**



After assigning weight to all the edges, the modified Huffman Tree is-



### Huffman Tree

To write Huffman Code for any character, traverse the Huffman Tree from root node to the leaf node of that character.

Following this rule, the Huffman Code for each character is-

a = 111

e = 10

i = 00

o = 11001

u = 1101

s = 01

t = 11000

**Time Complexity-**

The time complexity analysis of Huffman Coding is as follows-

- `extractMin()` is called  $2 \times (n-1)$  times if there are  $n$  nodes.
- As `extractMin()` calls `minHeapify()`, it takes  $O(\log n)$  time.

Thus, Overall time complexity of Huffman Coding becomes  **$O(n \log n)$** .

**Conclusion-** In this way we have explored Concept of Huffman Encoding using greedy method

**Assignment Question**

1. What is Huffman Encoding?
2. How many bits may be required for encoding the message 'mississippi'?
3. Which tree is used in Huffman encoding? Give one Example
4. Why Huffman coding is lossless compression?

**Reference link**

- <https://towardsdatascience.com/huffman-encoding-python-implementation-8448c3654328>
- <https://www.programiz.com/dsa/huffman-coding#cpp-code>
- <https://www.gatevidyalay.com/tag/huffman-coding-example-ppt/>